TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Vergleich und Analyse von Partitionierungsalgorithmen für Quicksort

Ausgeführt am Institut für

Diskrete Mathematik und Geometrie
der Technischen Universität Wien

unter der Anleitung von Ao.Univ.Prof. Dr. Alois Panholzer

durch

Stephan Beer, BSc
Reinlgasse 44/15, 1140 Wien

....
Datum Unterschrift

Ich widme diese Arbeit meiner Familie und allen Personen, die mich in dieser Zeit unterstützt haben!

Zusammenfassung:
Das Sortieren großer Datenmengen stellt in der Computerwissenschaft seit jeher einen wichtigen Bestandteil dar. Zwei wichtige Eigenschaften werden von Sortieralgorithmen gefordert, Korrektheit und Geschwindigkeit.
Seit Jahren ist die von Hoare 1962 entwickelte Quicksort-Familie das De-facto-Standardverfahren. Im Jahr 2009 sorgte allerdings eine Arbeit von Yaroslavskiy für Aufsehen. In dieser wurde gezeigt, dass eine bis dahin vernachlässigte Variante schneller ist als der alte Standard: Das Sortieren mit Hilfe zweier Pivot-Elemente.
Diese Arbeit hat sich das Ziel gesetzt, die neuesten Entwicklungen innerhalb dieser Familie zu analysieren und mit den alten Verfahren zu vergleichen. Neben einer ausführlichen theoretischen Analyse werden die betrachteten Verfahren auch praktisch umgesetzt und die benötigten Rechenzeiten verglichen.

Schlagwörter: Quicksort, Dual-Pivot, Sortieren

Abstract:
Sorting big data is one of the most important parts in computer science ever since. Correctness and speed are two key properties each alogrithm requires. Since Hoare presented the quicksort family in 1962 it has been one of the most used algorithms to sort big data sets.
In 2009 Vladimir Yaroslavskiy presented a new version of quicksort which completed the task faster than any other version. His work is based on selecting two pivots instead of one, a technique thought to be inferior.
The goal of this thesis is to analyse the latest developments within the quicksort family and compare them with older versions. Besides an exact analysis of each algorithm, the algorithms were also implemented and the needed running times were compared.

Keywords: Quicksort, Dual-Pivot, Sorting

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht. Ich erkläre mich damit einverstanden, dass die Arbeit mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft wird.

. . . .

Ort, Datum Unterschrift

Inhaltsverzeichnis

1 Einleitung

Eines der grundlegenden Probleme der elektronischen Datenverarbeitung ist das Sortieren von gegebenen Datensätzen in angemessener Zeit. Aus der schieren Menge an bekannten Sortierverfahren sticht besonders eines aufgrund seiner Schnelligkeit, Eleganz und doch einfachen Umsetzbarkeit heraus: Quicksort.
Quicksort wurde bereits 1962 von C.A.R. Hoare präsentiert [Hoa62] und stellt bis heute mit seinen vielen Modifikationen den De-facto-Standardsortieralgorithmus dar. Der Algorithmus gehört der Klasse der Divide-and-Conquer Algorithmen an: Durch die Wahl eines ausgezeichneten Pivot-Elements wird das Problem auf zwei einfachere Probleme reduziert.
Zusätzlich zur Basisvariante wird in dieser Arbeit auch untersucht, an welcher Stelle sich geschickt alternative Verfahren einsetzen lassen und wie man die Wahl des Pivot-Elements günstig beeinflussen lassen kann.
Einen wesentlichen Beitrag zur Analyse von Quicksort stellt die Doktorarbeit von R. Sedgewick [Sed75] von 1975 dar. Als einer der ersten analysierte er in dieser eine weitere Modifikation, das Teilen der Eingabe in drei Teilmengen durch die Wahl von zwei Pivot-Elementen. Allerdings stellte sich die von Sedgewick analysierte Version von Dual-Pivot Quicksort als nachteilig heraus und die Idee wurde wieder verworfen.
Erst 30 Jahre später, im Jahr 2009, präsentierte Vladimir Yaroslavskiy [Yar09] eine neue Variante von Dual-Pivot Quicksort, von der gezeigt wurde, dass sie den klassischen Varianten überlegen ist. Die Performance beider Varianten wird in dieser Arbeit im Hinblick auf die erwartete Anzahl von Vertauschungen und Vergleichen analysiert und betrachtet. Zusätzlich werden im Anschluss noch zwei optimale Varianten betrachtet.
Im letzten Teil der Arbeit werden alle betrachteten Varianten in C-Code umgesetzt und Laufzeitanalysen durchgeführt, wobei die erhaltenen Ergebnisse gut zu den erwarteten Laufzeiten passen.

Gegliedert wurde diese Arbeit in folgender Weise: Im zweiten Kapitel werden grundlegende Notationen sowie oft genutzte mathematische Identitäten und Sätze aufgelistet. Weiters wird kurz der Sortieralgorithmus Insertionsort betrachtet.
Das dritte Kapitel widmet sich der klassischen Version von Quicksort und seinen Modifikationen. Gemeinsam ist allen betrachteten Varianten, dass zur Partitionierung jeweils nur ein Pivot-Element benötigt wird. Der Anfang des Kapitels widmet sich kurz Hoare's Analyse bevor im Anschluss detaillierte Laufzeitanalysen aufgrund von benötigten Ope-

rationen in Knuths theoretischem Rechner MMIX [Knu98, Knu05] betrachtet werden.
Die beiden Dual-Pivot Quicksort Varianten von Sedgewick und Yaroslavskiy werden im vierten Kapitel betrachtet. Anhand eines allgemeinen Falles und eines Spezialfalles wird die Güte beider Verfahren aufgrund der erwarteten Anzahl von Vertauschungen und Vergleichen bestimmt. Danach werden noch optimale Varianten besprochen.
Im fünften Kapitel werden schlussendlich die praktischen Umsetzungen aller Varianten und deren Laufzeiten präsentiert.

2 Grundlagen

In diesem Kapitel werden einige grundlegende mathematische Definitionen und Identitäten angeben, die im Laufe dieser Diplomarbeit immer wieder auftreten.
Manche der hier angeführten Punkte werden dem vertrauten Leser trivial oder zumindest vertraut erscheinen, aus Gründen der Vollständigkeit werden diese nichtsdestotrotz hier angeführt.

2.1 Mathematische Grundlagen und Identitäten

- Logarithmus:
  Der Logarithmus zur Basis b von a ist die Umkehrfunktion der Exponentialrechnung. Es gilt:
  $$\log_b(a) = x \Leftrightarrow b^x = a.$$
  Für den Logarithmus zur Basis e, genannt der natürliche Logarithmus, verwenden wir die Notation $\ln(a) := \log_e(a)$. Die Konstante e heißt Eulersche Zahl, ihr numerischer Wert beträgt $2,7182818\ldots$.

- Die Landau-Symbole:
  Aus der Reihe der Landau-Symbole wird in dieser Arbeit nur das Symbol $\mathcal{O}$ verwendet. Die Definition lautet: Seien f und g Folgen reeller Zahlen, $n \in \mathbb{N}$. Dann gilt
  $$f \in \mathcal{O}(g) \Leftrightarrow \limsup_{n\to\infty} \left|\frac{f_n}{g_n}\right| < \infty. \qquad \text{[Knu76]}$$

- Die Harmonische Reihe:
  Mit $\mathcal{H}_\infty$ bezeichnen wir die unendliche Reihe $\sum_{i=1} \frac{1}{i}$ und mit $\mathcal{H}_n = \sum_{i=1}^{n} \frac{1}{i}$ die n-te Partialsumme. Die Harmonische Reihe ist divergent, für die Partialsummen gilt die Näherung
  $$\mathcal{H}_n = \ln(n) + \gamma + \mathcal{O}(n^{-1}).$$
  Die Konstante γ heißt Euler-Mascheroni-Konstante, ihr numerischer Wert beträgt $0,577215\ldots$.[GKP94]

- Satz der totalen Wahrscheinlichkeit:
  Ist A ein beliebiges Ereignis und $(H_i)_{i\in I}$ ein vollständiges Ereignissystem auf einem

Wahrscheinlichkeitsraum (Ω, Σ, P), so gilt

$$P(A) = \sum_{i \in I} P(H_i) P(A|H_i). \qquad \text{[Kus11]}$$

- Der Rückwärtsdifferenzenoperator ∇:
  Für $\nabla^r f(n)$ gilt:

$$\begin{aligned} \nabla^r f(n) &:= \nabla^{r-1} f(n) - \nabla^{r-1} f(n-1) \quad r = 1, 2, \ldots, \quad \text{mit} \\ \nabla^0 f(n) &= f(n). \end{aligned} \tag{2.1}$$

  Insbesondere gilt:

$$\begin{aligned} \nabla f(n) &= f(n) - f(n-1), \\ \nabla^2 f(n) &= f(n) - 2f(n-1) + f(n-2). \end{aligned}$$

- Die folgenden Identitäten werden gehäuft in dieser Arbeit angewandt:
  - Vandermondesche Identität [GKP94, Tabelle 174 auf Seite 174]

$$\sum_{k=0}^{n} \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}, \qquad n \in \mathbb{Z}, \tag{2.2}$$

  - [GKP94, Gleichung 5.22 auf Seite 169]

$$\sum_{k} \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}, \qquad m, n \in \mathbb{Z}, \tag{2.3}$$

$$\sum_{k=0}^{l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}, \qquad l, m, n, q \in \mathbb{N},\ n \geq q, \tag{2.4}$$

  - [GKP94, Gleichung 6.70 auf Seite 280]

$$\sum_{0 \leq k < n} \binom{k}{m} \mathcal{H}_k = \binom{n}{m+1} \left(\mathcal{H}_n - \frac{1}{m+1} \right), \qquad m, n \in \mathbb{N}, \tag{2.5}$$

    Diese Gleichung taucht oft für den Fall $m = 1$ auf und reduziert sich damit

zu

$$\sum_{0\le k<n} k\mathcal{H}_k = \frac{n(n-1)}{2}\left(\mathcal{H}_n - \frac{1}{2}\right), \qquad n \in \mathbb{N}, \tag{2.6}$$

– [GKP94, Tabelle 174 auf Seite 174]

$$\sum_{0\le k\le n} \binom{k}{m} = \binom{n+1}{m+1}, \qquad m, n \in \mathbb{N}. \tag{2.7}$$

2.2 Erzeugende Funktionen

Die Methode der Erzeugenden Funktionen ist ein Verfahren aus der abzählenden Kombinatorik. Dabei werden die gesuchten Zahlenwerte $a_0, a_1, \ldots$ als Koeffizienten einer formalen Potenzreihe $A(z) = \sum_{n\ge0} a_n z^n$ aufgefasst, die man als Erzeugende Funktion der Folge (a_n) bezeichnet.

Auf der Menge der formalen Potenzreihe $R[[z]] := \{\sum_{n\ge0} a_n z^n : a_n \in R\}$ können Rechenoperationen folgendermaßen definiert werden:

$$\begin{aligned} +: &\quad A(z) + B(z) = \sum_{n\ge0}(a_n + b_n)z^n, \\ *: &\quad A(z) * B(z) = \sum_{n\ge0}\left(\sum_{k=0}^{n} a_k b_{n-k}\right) z^n. \end{aligned}$$

Ist R ein Ring oder ein Integritätsbereich, übertragen sich die strukturellen Eigenschaften auf die Struktur $(R[[z]], +, *)$. Falls R ein Körper ist, erhält man dennoch keinen Körper, da nur im Fall $a_0 \neq 0$ ein multiplikatives Inverses existiert [Wil94].

2.3 Insertionsort

Insertionsort ist ein einfaches, stabiles Sortierverfahren und folgt dem intuitiven Ansatz, eine Folge zu sortieren. Hierbei wird einer unsortierten Folge ein Element entnommen und an die richtige Stelle einer anfangs leeren Ausgabefolge gesetzt.

Im Average-Case und im Worst-Case weist Insertionsort eine Laufzeit von $\mathcal{O}(n^2)$ auf, wobei n die Anzahl der zu sortierenden Elemente bezeichnet. Im Fall einer bereits sortierten Folge tritt der Best-Case ein. In diesem Fall besitzt Insertionsort eine lineare Laufzeit $\mathcal{O}(n)$ und ist somit in diesem Fall auch weiter entwickelten Verfahren überlegen.
Für kleine Listen ist es ein sehr effizientes Verfahren und wird deshalb zum Sortieren kleiner Teillisten bei Quicksort verwendet.

Der folgenden Pseudocode zeigt eine effiziente Programmierung von Insertionsort von Sedgewick [Sed75, Programm 1.3 auf Seite 6]:

Algorithm 1 Insertionsort

```
1:  procedure INSERTIONSORT(A)
2:      A[0] = −∞;
3:      for i = 2 to length(A) do
4:          if A[i]<A[i-1] then
5:              key := A[i];
6:              j := i − 1;
7:              while A[j − 1] > key do
8:                  A[j + 1] := A[j];
9:                  j := j − 1;
10:             end while
11:             A[j + 1] := key;
12:         end if
13:     end for
```

Unter der Annahme, dass die Eingabe aus verschiedenen, zufälligen Elementen besteht, beträgt die Laufzeit für Insertionsort

$$L(n) = 3D_n + 8E_n + 7n + 6, \tag{2.8}$$

Zeiteinheiten [Sed75, Seite 12]. Wir sehen also, dass die Laufzeit von zwei Größen abhängt, nämlich von D_n, der Anzahl der Einschübe, und von E_n, der Anzahl der Schlüsselbewegungen während der Einschübe. Durch die Bestimmung der Erwartungswerte dieser beiden Größen sind wir in der Lage, die erwartete Laufzeit anzugeben.

Wir beginnen mit D_n. Die Größe D_n beschreibt die Anzahl der Einschübe bei Insertionsort. Dies entspricht genau der Anzahl an Elementen in A, die zumindest ein größeres Element links von ihnen besitzen.
Für jedes Element $A[i]$ der Eingabefolge bezeichnet nun V_i die genaue Anzahl an solchen Elementen, die zwar einen kleineren Index als $A[i]$ besitzen aber einen größeren numerischen Wert aufweisen. Diese V_i, die die Anzahl der Fehlstellungen beschreiben, können offensichtlich die folgenden Werte annehmen:

$$V_1 = 0, 0 \leq V_2 \leq 1, 0 \leq V_3 \leq 2, \ldots, 0 \leq V_n \leq n-1.$$

Unsere gesuchte Größe D_n entspricht nun genau der Anzahl der V_i, für welche $V_i \neq 0$ gilt. Da jede der $n!$ möglichen Permutationen der Eingabe gleich wahrscheinlich ist, sind auch alle Werte für die einzelnen V_i gleich wahrscheinlich und somit gilt $\mathbb{P}[V_i \neq 0] = \left(1 - \frac{1}{i}\right)$. Daraus folgt für D_n:

$$D_n = (1-1) + \left(1 - \frac{1}{2}\right) + \left(1 - \frac{1}{3}\right) + \cdots + \left(1 - \frac{1}{n}\right) = n - \mathcal{H}_n.$$

Kommen wir nun zu E_n. Die Anzahl der Schlüsselbewegungen E_n bei Insertionsort entspricht genau der Anzahl der Fehlstellungen. Bevor wir E_n berechnen können benötigen wir zuerst zwei wichtige Feststellungen. Erstens, für die maximale Anzahl an Fehlstellungen gilt: $\sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2}$. Zweitens, besitzt $A[1], A[2], \ldots, A[n]$ genau k Fehlstellungen, dann besitzt die reverse Permutation $A[n], A[n-1], \ldots, A[1]$ genau $\frac{n(n-1)}{2} - k$ Fehlstellungen. Wir bezeichnen nun mit e_{nk} die Wahrscheinlichkeit, dass A genau k Fehlstellungen aufweist und mit e'_{nk} die Wahrscheinlichkeit, dass die zugehörige reverse Permutation genau $\frac{n(n-1)}{2} - k$ Fehlstellungen besitzt. Es gilt nun für E_n:

$$\begin{aligned} E_n = \sum_{0 \leq k \leq \frac{n(n-1)}{2}} k \cdot e_{nk} &= \sum_{0 \leq k \leq \frac{n(n-1)}{2}} \left(\frac{n(n-1)}{2} - k\right) e'_{nk} \\ &= \sum_{0 \leq k \leq \frac{n(n-1)}{2}} \left(\frac{n(n-1)}{2} - k\right) e_{nk}, \end{aligned}$$

und somit:

$$\begin{aligned} 2E_n &= \sum_{0 \le k \le \frac{n(n-1)}{2}} \left(k + \frac{n(n-1)}{2} - k \right) e_{nk} \\ &= \frac{n(n-1)}{2} \underbrace{\sum_{0 \le k \le \frac{n(n-1)}{2}} e_{nk}}_{=1} \\ &= \frac{n(n-1)}{2}, \\ E_n &= \frac{n(n-1)}{4}. \end{aligned}$$

Setzen wir nun die erwarteten Werte für D_n und E_n in unsere Formel für die Laufzeit (2.8) ein, erhalten wir für die durchschnittliche Laufzeit von Insertionsort:

$$2n^2 + 8n - 3\mathcal{H}_n - 6$$

Zeiteinheiten.
Knuth [Knu98] gibt die Dauer einer Zeiteinheit auf einem damals schnellen Computer mit 10 Nanosekunden an. Der Leser muss aber bedenken, dass Knuth's Werk bereits 1998 erschienen ist und moderne Computer entsprechend schneller sind.

3 Quicksort:
Das Original von Hoare und erste Verbesserungen

„I think Quicksort is the only really interesting algorithm that I've ever developed."

- Tony Hoare

„This method combines elegance and efficiency, and it remains today the most useful general-purpose sorting method for computers."

- Robert Sedgewick

3.1 Allgemeines über Quicksort

Im Jahr 1960 stellte Charles Antony Richard Hoare das Verfahren Quicksort vor, um Listen von Elementen zu sortieren. Aufgrund seiner Effizienz, einfachen Programmierbarkeit und vielen Verbesserungen stellt Quicksort bis heute den De-facto Standardsortieralgorithmus dar.

Quicksort gehört zur Klasse der „Divide-and-Conquer" Algorithmen und arbeitet in folgender Weise:
Es sei eine Liste A mit n Elementen gegeben. Wähle aus dieser Liste ein sogenanntes Pivot-Element p und sortiere die Liste in der Form um, dass alle Elemente kleiner als p links von p stehen und alle größeren Elemente rechts. Das Element p befindet sich nun bereits an der richtigen Position und muss deswegen in den weiteren Schritten nicht mehr beachtet werden. Wendet man dieses Verfahren nun rekursiv auf die beiden entstandenen Teillisten an, sortiert dies die gesamte Liste.
Da die linke und rechte Teilliste zusammen nur mehr $n - 1$ Elemente beinhalten, ist garantiert, dass das Verfahren nach endlich vielen Aufrufen terminiert.

Implementiert wird Quicksort als In-place-Algorithmus. Das bedeutet, dass die Elemente der zu sortierenden Liste nicht in zusätzliche Speicher kopiert werden, sondern ausschließlich innerhalb der Liste vertauscht werden.

3.2 Die ursprünglichen Listings

Nachfolgend sind die Algorithmen „partition“ und „quicksort“ angegeben, wie sie von Hoare in der Zeitschrift *Communications of the ACM* publiziert wurden. [Hoa61]

Algorithm 2 Partition

```
1:  procedure PARTITION(A,M,N,I,J)
2:      real X; integer F;
3:      F := random(M, N); X = A[F];
4:      I := M; J := N;
5:  up:
6:      for I = I step 1 until N do
7:          if X < A[I] then go to down;
8:      I := N;
9:  down:
10:     for J := J step −1 until M do
11:         if A[J] < X then go to change;
12:     J := M;
13: change
14:     if I < J then
15:         exchange(A[I], A[J]);
16:         I := I + 1; J := J − 1;
17:         go to up;
18:     else if I < F then
19:         exchange(A[I], A[F]);
20:         I := I + 1;
21:     else if F < J then
22:         exchange(A[F], A[J]);
23:         J := J − 1;
```

Algorithm 3 Quicksort

```
1: procedure QUICKSORT(A,M,N)
2:     integer I,J;
3:     if M < N then
4:         partition(A, M, N, I, J);
5:         quicksort(A, M, J);
6:         quicksort(A, I, N);
7:     return A;
```

3.3 Das Flussdiagramm von Quicksort

Aus dem folgenden Flussdiagramm ist der Ablauf von Quicksort sehr gut ersichtlich. Besitzt die eingegebene Folge mehr als ein Element, wird zuerst die Teilungsprozedur 'partition' aufgerufen. In dieser wird ein Pivot-Element gewählt und die Folge umsortiert. Danach folgen rekursive Aufrufe für die beiden kleineren Teilfolgen.

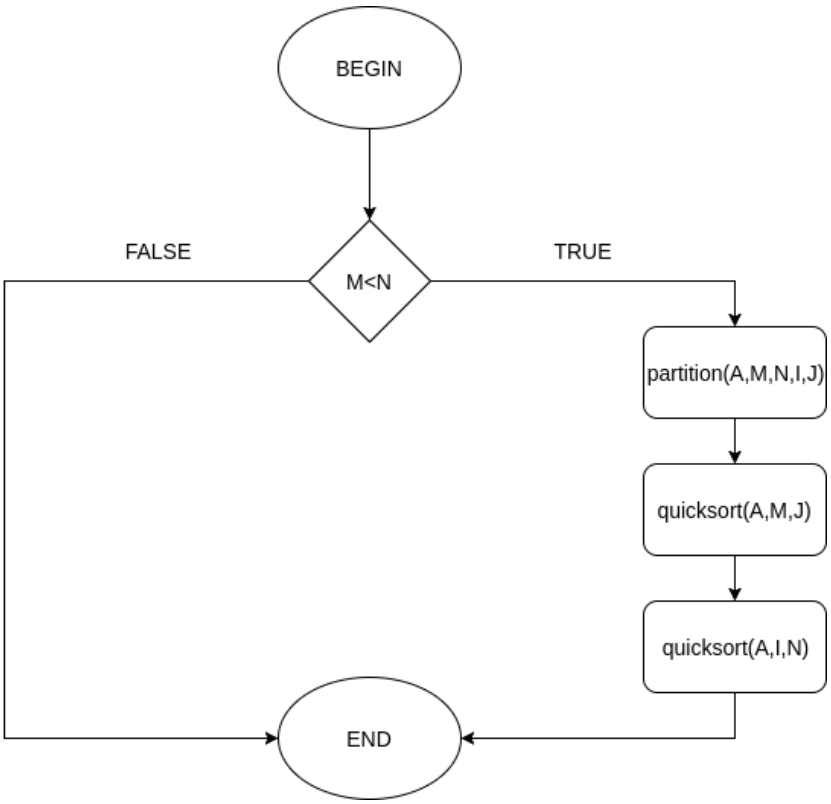


Abbildung 1: Flussdiagramm Quicksort

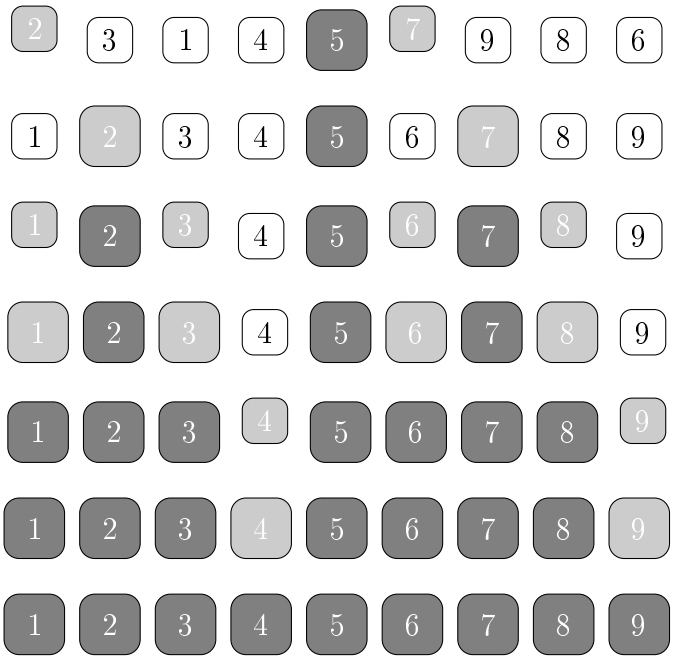
3.4 Ein Beispiel

Die Vorgehensweise von Quicksort wird nun Anhand eines kleinen Beispiels demonstriert. Als Pivot-Element wird in diesem Beispiel das erste Element der Liste ausgewählt, der restliche Algorithmus folgt der Partitionierung von Hoare:

Schritt 1: Wähle ein Pivot-Element.

Schritt 2: Alle Elemente die kleiner als das Pivot-Element sind kommen in die linke Teilliste, die die größer sind in die Rechte.

Schritt 3: Das Pivot-Element aus Schritt 1 ist nun an der richtigen Stelle. Wiederhole Schritt 1 und Schritt 2 mit beiden Teillisten.



3.5 Das Eingabemodell

Für alle in dieser Arbeit betrachteten Varianten von Quicksort gilt, dass die Eingabe dem folgenden Modell entspricht: Die zu sortierenden Schlüssel sind alle verschieden, sind Elemente der Menge $\{1, \ldots, n\}$ und für jede Permutation gilt, dass diese mit der Wahrscheinlichkeit $1/n!$ auftritt, also jede Permutation gleich wahrscheinlich ist.

3.6 Durchschnittliche Anzahl an Operationen

Bevor wir uns im späteren Verlauf dieses Kapitels mit den genauen Laufzeiten verschiedener Quicksort Varianten auseinandersetzen, die natürlich von den konkreten Implementierungen und der zugrunde liegenden Maschine abhängen, wird an dieser Stelle erarbeitet, welche Anzahl an Operationen, Vergleichen und Vertauschungen man für das Sortieren von n Elementen mit Hilfe des ursprünglichen Verfahrens erwarten kann. Dieses Kapitel orientiert sich an der ursprünglichen Analyse durch Hoare [Hoa62]. Daher können sich manche kleine Unterschiede im Vergleich zu späteren Analysen ergeben. Weiters wurden auf Details und Zwischenschritte während der Umformungen verzichtet, da im späteren Verlauf der Arbeit die Herleitung der Ergebnisse ohnehin detailliert beschrieben werden.

Die genaue Anzahl an Vergleichen, um eine Sequenz der Länge n im ersten Partitionierungsschritt zu teilen, hängt von der genauen Implementierung ab. In jedem Fall liegen die Vergleiche in der Größenordnung von n.

Die Anzahl der Vertauschungen unterscheidet sich bei jedem Aufruf und kann aus diesem Grund nur als bedingter Erwartungswert angegeben werden.

Denken wir an die Situation am Ende des ersten Teilungsprozesses. Angenommen das gewählte Pivot-Element p ist das r-kleinste innerhalb des Sequenz. Nach der letzten Vertauschung befindet sich das Pivot-Element nun an der r-ten Stelle, die $r-1$ Elemente, die kleiner als p sind, befinden sich links davon auf den Positionen 1 bis $r-1$.
Die Wahrscheinlichkeit eines Elements, größer als das Pivot-Element zu sein, beträgt $\frac{n-r}{n}$. Es gibt nun exakt $r-1$ Stellen, auf denen Elemente, die größer als das Pivot-Element sind, stehen können und folglich während der Partitionierung getauscht werden müssen. Daher ist die erwartete Anzahl der Elemente, die zu tauschen sind, gleich:

$$\frac{(n-r)(r-1)}{n}.$$

Summiert man nun über alle r auf, dividiert durch n, um die Wahrscheinlichkeit $\frac{1}{n}$, dass die Wahl des Pivot-Elements auf p fällt, zu berücksichtigen und addiert eine zusätzliche Vertauschung, um das Pivot-Element am Schluss in die richtige Position zu bringen, erhält man für die erwartete Anzahl an Vertauschungen gleich

$$\begin{aligned}\frac{\sum_{r=1}^{n} \frac{(n-r)(r-1)}{n}}{n} + 1 &= \frac{1}{n^2}\sum_{r=1}^{n}(nr - n - r^2 + r) + 1 \\ &= \frac{n}{6} + \frac{1}{2} + \frac{1}{3n}.\end{aligned}$$

Nachdem wir nun die Werte für die Anzahl der Vergleiche und die erwartete Anzahl an Vertauschungen im ersten Partitionierungsschritt haben, können wir nun berechnen, wie viele Vergleiche und Vertauschungen wir während des gesamten Teilungsprozesses erwarten. Die Anzahl der Vergleiche und Vertauschungen, die im ersten Partitionierungsschritt benötigt werden, haben allgemein die folgende Form:

$$an + b + \frac{c}{n},$$

die Koeffizienten werden, je nachdem was berechnet werden soll, einfach abgelesen.

Angenommen das gewählte Pivot-Element ist das r-größte in der Folge. Die Anzahl an Operationen, die benötigt werden, um die Folge zu sortieren, wir schreiben diese hier mit T_n an und unterscheiden nicht zwischen Vertauschungen und Vergleichen, setzt sich aus den folgenden Teilen zusammen: Die Operationen, die benötigt werden, um die beiden Teilfolgen zu sortieren plus die Zeit um die Folge zu partitionieren. Mathematisch ergibt sich die folgende Rekursionsgleichung für die erwartete Anzahl an Operationen:

$$T_n = T_{r-1} + T_{n-r} + an + b + \frac{c}{n},$$

für den Fall, dass das gewählte Pivot-Element das r-kleinste in der Sequenz ist.
Summation über alle r und Division durch n, die Wahrscheinlichkeit für das Pivot-

Element das r-kleinste zu sein, liefert die Rekursion

$$T_n = \frac{2}{n}\sum_{r=1}^{n-1} T_r + an + b + \frac{c}{n}, \quad n \geq 1.$$

Bereits 1962 verwendete Hoare eine Verbesserung für Quicksort, die im Detail später betrachtet wird und hier nur am Rande erwähnt werden soll. Nämlich die Feststellung, dass Quicksort seine Schnelligkeit erst bei einer gewissen Größe der zu sortierenden Sequenz ausspielt. Für kleine Sequenzen A mit Größe $n < M$ wird ein anderes, nicht rekursives, Verfahren verwendet.
Somit ist nun die Rekursion

$$T_n = \frac{2}{n}\sum_{r=1}^{n-1} T_r + an + b + \frac{c}{n}, \quad n \geq M,$$

zu lösen und die exakte Lösung lautet:

$$\begin{aligned} T_n =& \frac{2(n+1)}{M(M+1)}\sum_{r=1}^{M-1} T_r + \frac{(n+1)c}{M(M+1)} \\ & - \left[\frac{2(n+1)}{M+1} - 1\right] b \\ & - \left[2(n+1)\sum_{M+1}^{n}\frac{1}{r} - \frac{4(n+1)}{M+1} + n + 4\right] a. \end{aligned}$$

Die Korrektheit der Lösung kann leicht durch das Einsetzen in die ursprüngliche Gleichung gezeigt werden.

Wir haben nun eine Formel für die Anzahl an Operationen,die Quicksort benötigt, um eine Liste mit n Elementen zu sortieren. An dieser Stelle stellt sich die Frage, wie gut Quicksort tatsächlich ist. Einen Anhaltspunkt, um diese Frage beantworten zu können, liefert der Vergleich von der von Quicksort benötigten Anzahl an Vergleichen mit dem theoretischen Minimum an Vergleichen, die benötigt werden um eine Folge zu sortieren. Bei großem n erhält man die Anzahl an Vergleichen für Quicksort, indem wir in der Formel für T_n die folgenden Werte einsetzen: Wir betrachten den Fall $M = 1$ und $a = 1$

und $b = c = T_1 = 0$. Wir ignorieren die kleineren Terme und erhalten

$$2n \sum_{1}^{n} \frac{1}{r} \sim 2n \ln(n).$$

Das theoretische Minimum [CSRL09, Seite 193] liegt bei

$$\frac{-\log(n!)}{-\log(2)} = \log_2(n!) \sim n \log_2(n).$$

Die durchschnittliche Anzahl der Vergleiche liegt also um den Faktor $2 \ln(2) \sim 1,4$ höher als das theoretische Minimum.

Für die folgende Analyse des Verfahrens werden wir allerdings eine Prozedur betrachten, die beide Algorithmen, sowohl Quicksort als auch die Partitionierung, gleichzeitig beinhaltet und in dieser Form von R. Sedgewick präsentiert wurde [Sed77, Seite 329].

Algorithm 4 Quicksort nach Sedgewick

```
1:  procedure QUICKSORT(integer l,r)
2:      //The array A is declared to be A[1 : n + 1], with A[n + 1] = ∞
3:      if r − l ≥ M then
4:          i := l; j := r + 1; v := A[l];
5:          do
6:              do
7:                  i = i + 1;
8:              while A[i] < v
9:              do
10:                 j = j − 1;
11:             while A[j] > v
12:             exchange(A[i], A[j]);
13:         while i < j
14:         exchange(A[l], A[j]);
15:         if j − l < r − i + 1 then
16:             quicksort(l, j − 1); quicksort(i, r);
17:         else
18:             quicksort(i, r); quicksort(l, j − 1);
19:         end if
20:     end if
```

Beide vorgestellten Quicksort-Varianten verwenden die sogenannte 'Crossing-Pointers-Technik' [Hoa62, Seite 10], um den späteren Platz des Pivot-Elements in der sortierten Folge zu finden.
Ausgehend von den Grenzen des im Augenblick betrachteten Arrays zählt der erste Pointer solange aufwärts bis ein Element größer als das Pivot-Element erreicht wird. Danach startet der zweite Pointer und zählt solange abwärts bis ein Element kleiner als das Pivot-Element erreicht wird. Diese ausgewählten Elemente werden getauscht und die Pointer zählen weiter. Sobald sich die Pointer überkreuzen, bricht die Schleife ab und das Pivot-Element wird an den richtigen Platz getauscht.

Beachtenswert ist auch, dass in den beiden Algorithmen das Pivot-Element unterschiedlich gewählt wird. Während Hoare den Zufall entscheiden lässt, wählt Sedgewick das erste Element der Folge.

Da wir von einer zufälligen Eingabe ausgehen, bleibt die Zufälligkeit bei beiden Varianten erhalten. Hoare's Verfahren liefert allerdings bessere Ergebnisse in der Praxis, da die Wahrscheinlichkeit des Auftretens eines worst-case, bei Sedgewick's Partitionierung wäre dies eine bereits sortierte Folge, geringer ist. Durch Randomisieren, d.h. das Tauschen des ersten Elements mit einem zufälligen Element im ersten Schritt des Algorithmus, kann dieses Problem einer sortierten Folge vermieden werden.

Im Fall, dass $M = 1$ gilt funktioniert das Verfahren genauso wie oben besprochen. Da Quicksort seine Effektivität erst bei größeren Sequenzen ausspielt, werden kleinere zu sortierende Folgen mittels anderer Verfahren sortiert. Es bietet sich zum Beispiel Insertionsort an, das kurze Folgen erwiesenermaßen schnell sortiert. Somit stellt der Parameter M eine Schranke dar, ab der Quicksort verwendet werden soll. Die richtige Wahl von M ist somit auch ein wesentlicher Faktor für die Geschwindigkeit des Algorithmus.

Insertionsort entnimmt der unsortierten Eingabefolge ein beliebiges Element und fügt es an richtiger Stelle in die Ausgabenfolge ein. Eine detailliertere Betrachtung wurde bereits in Kapitel 2 vorgenommen. Der Vollständigkeit halber ist an dieser Stelle nochmals der Algorithmus in Pseudocode angegeben.

Algorithm 5 Insertionsort

```
1:  procedure INSERTIONSORT(A)
2:      A[0] = −∞;
3:      for i = 2 to length(A) do
4:          if A[i]<A[i-1] then
5:              key := A[i];
6:              j := i − 1;
7:              while A[j − 1] > key do
8:                  A[j + 1] := A[j];
9:                  j := j − 1;
10:             end while
11:             A[j + 1] := key;
12:         end if
13:     end for
```

Es gibt nun zwei Möglichkeiten, Insertionsort bei der Verwendung von Quicksort aufzurufen. Die erste offensichtliche Möglichkeit besteht darin, das letzte "**end if** " in Algo-

rithmus 4 durch "**else** insertionsort(l,r) **end if**" zu ersetzten. Jede Teilliste mit weniger als M Elementen würde somit während des Partitionierungsschrittes direkt sortiert werden.
Besser ist es allerdings, die kleinen Teillisten während des Partitionierungsprozesses zu ignorieren und die kleinen Teillisten durch einen einzigen Insertionsort Aufruf über die gesamte Eingabe zu sortieren.
Der Aufruf der gesamten Prozedur schaut demnach folgendermaßen aus:

if $n > M$ **then** $quicksort(1, n)$ **end if**;
$insertionsort(1, n)$;

[Sed77, Seite 330].

3.7 Average Case - Processor Cycle Counts

Die gesamte Laufzeit von Quicksort kann anhand der folgenden 5 Kenngrößen berechnet werden:

A - Tiefe der Rekursion

B - Anzahl der Vertauschungen

C - Anzahl der Vergleiche

D - Anzahl der Einschübe bei Insertionsort

E - Anzahl an Schlüsselbewegungen während der Einschübe.

Je nach der konkreten Implementierung können noch weitere Größen von Bedeutung sein. Wird zum Beispiel auf den rekursiven Aufruf verzichtet und der Algorithmus iterativ mit Hilfe eines Stacks abgearbeitet, stellt die Anzahl der Stack-Pushes S eine weitere solche Größe dar. Bei dieser Realisierung werden nur Teillisten mit mehr als M Elementen auf den Stack gelegt, daher gilt die berechnete Anzahl an Stack-Pushes nur für $n > 2M + 1$. Knuth [Knu98, Seite 118] gibt für den obigen Algorithmus eine Gesamtlaufzeit von

$$L(n) = 24A_n + 11B_n + 4C_n + 9S_n + 3D_n + 8E_n + 7n \tag{3.1}$$

Zeiteinheiten an, wobei $3D_n + 8E_n + 7n - 6$ auf die Verwendung von Insertionsort fallen [Sed77, Seite 332]. Durch eine eingehende Untersuchung dieser Werte werden wir in der

Lage sein, eine günstige Wahl für M zu treffen. Wir nehmen an, dass die Elemente der zu sortierenden Folge paarweise verschieden sind. Die Gleichheit einzelner Werte stört das Verfahren im wesentlichen nicht [Knu98, Seite 119].

Bei der Analyse des Best-Case und des Worst-Case wird die auftretende Rekursion nur in einem sehr allgemeinen Fall gelöst. Der interessantere Average-Case wird detaillierter betrachtet.

3.7.1 Best Case

Die allgemeine Rekursionsgleichung kann leicht aus dem Pseudocode ersehen werden. Sie setzt sich aus der Arbeit für die Partitionierung und dem Sortieren der beiden entstanden Teillisten zusammen und lautet:

$$T(n) = T(K) + T(n - K - 1) + O(n).$$

Im günstigsten Fall wird die zu sortierende Folge durch die Wahl des Pivot-Elements in zwei etwa gleich große Teilfolgen mit jeweils $\frac{n-1}{2}$ Elemente geteilt.
Wir betrachten die Rekursion nun für den Fall, dass $n = 2^k$ gilt und sich die Kosten für die Partitionierung von n Elemente auf cn, mit $c \in \mathbb{N}$, belaufen. Sollte n nicht die angegebene Darstellung aufweisen können wir ohne Beeinträchtigung der Allgemeinheit auf das nächst größere n dieser Art aufrunden. Für die Anzahl an Operationen im Best-

Case ergibt sich somit:

$$\begin{aligned}
T(n) &= 2T\left(\frac{n-1}{2}\right) + cn \\
&= 2T\left(\frac{2^k-1}{2}\right) + c2^k \\
&= 2T\left(2^{k-1} - \frac{1}{2}\right) + c2^k \\
&\leq 2T(2^{k-1}) + c2^k \\
&= 2\left[2T(2^{k-2} - \frac{1}{2}) + c(2^{k-1})\right] + c2^k \\
&= 4T(2^{k-2}) + 2c2^k \\
&\leq 8T(2^{k-3}) + 2c2^k \\
&\vdots \\
&\leq 2^k T(1) + kc(2^k) \\
&= \log_2(n)T(1) + c\log_2(n)n = \mathcal{O}(n\log(n)).
\end{aligned}$$

3.7.2 Worst Case

Der Worst-Case tritt auf, wenn in der Partitionierungsphase das Pivot-Element das größte oder kleinste Element in der gerade betrachteten Folge darstellt. In diesem Fall wird die Größe des Teilfeldes nur um eins reduziert. Werden für die Partitionierungskosten für n Elemente erneut cn angenommen, erhält man für die Anzahl an Operationen:

$$\begin{aligned}
T(n) &= T(0) + T(n-1) + cn \\
&= T(n-1) + cn \\
&= T(n-2) + c(n-1) + cn \\
&\vdots \\
&= T(1) + c\sum_{i=2}^{n} i \\
&= c\left(\frac{n(n+1)}{2} - 1\right) = \mathcal{O}(n^2).
\end{aligned}$$

Eine gute Wahl für das Pivot-Element ist somit von essentieller Bedeutung. Bei den beiden vorangegangen Programmen wurde dies auf unterschiedliche Weise gelöst. Hoare wählt das Pivot-Element zufällig, in Sedgewicks Version wird immer das Element am

linken Rand gewählt.
Der Worst-Case für Sedgewicks Algorithmus wäre somit eine bereits sortierte Folge. Aber auch bei Sedgewick kann dieses Problem durch Randomisieren gelöst werden: Man tauscht im ersten Schritt ein zufälliges Element an die erste Stelle und führt danach die Partitionierung durch.

3.7.3 Average Case

Im letzten Abschnitt wurde festgestellt, dass Quicksort im schlechtesten Fall eine quadratische Laufzeit aufweist. Da dieser Fall im Allgemeinen sehr selten auftritt, folgt nun eine Average-Case Analyse, welche zeigen wird, dass Quicksort im Allgemeinen sehr schnell ist.
Wir versuchen nun eine Formel anzugeben, welche für gegebenes n die zu erwartende Anzahl an Operationen während des Teilungsprozesses liefert. Da die unterschiedlichen Größen, die wir betrachten, alle derselben Rekursion unterliegen und sich nur in den Kosten während des ersten Partitionierungsschrittes unterscheiden, genügt es, die Rekursion einmal zu lösen. Die Kostenfunktion für das Sortieren mit Quicksort werden wir in der folgenden Berechnung F_n nennen, mit f_n bezeichnen wir die jeweiligen Partitionierungskosten. Wir nehmen an, dass das zu sortierende Feld in zufälliger Ordnung vorliegt. Im Fall $n \leq M$ gilt $F_n = 0$, hier kommt Insertionsort zum Einsatz.
Im Fall $n > M$ gilt, dass nach der Partitionierung alle Elemente, die kleiner als das Pivot-Element sind, sich in der gleichen Teilliste befinden. Damit setzen sich die Kosten für die Kenngrößen D_n und E_n von Insertionsort für die gesamten Liste aus den Kosten der Teillisten zusammen. Daher hält die Rekursion, die wir im folgenden aufstellen und lösen, auch für die Größen D_n und E_n mit $f_n = 0$. Weiters ist es genau diese Eigenschaft die es rechtfertigt einen einzigen Aufruf von Insertionsort am Ende des Verfahren durchzuführen statt die Teilliste mit Länge $< M$ sofort zu sortieren.

Bei der folgenden Lösung der Rekursion und der Bestimmung der einzelnen Kenngrößen wird im wesentlich [Sed77, ab Seite 332] gefolgt. Aus der Tatsache, dass jedes gewählte Pivot-Element k, $1 \leq k \leq n$, mit der gleichen Wahrscheinlichkeit $\frac{1}{n}$ auftritt, folgt aus der Struktur des Algorithmus,

$$F_n = \frac{1}{n} \sum_{1 \leq k \leq n} (f_n + F_{k-1} + F_{n-k}),$$

bzw.

$$F_n = f_n + \frac{2}{n} \sum_{0 \le k < n} F_k, \quad \text{für } n > M. \tag{3.2}$$

Um die rekursive Gleichung (3.2) zu lösen, eliminiert man als erstes die auftretende Summe, indem man die Rekursion für $n-1$ von der Rekursion für n subtrahiert. Es gilt:

$$nF_n = nf_n + 2F_{n-1} + 2 \sum_{0 \le k < n} F_k \qquad \text{und}$$
$$(n-1)F_{n-1} = (n-1)f_{n-1} + 2 \sum_{0 \le k < n-1} F_k.$$

Subtraktion dieser beiden Gleichungen liefert:

$$nF_n - (n-1)F_{n-1} = nf_n - (n-1)f_{n-1} + 2F_{n-1}$$

oder

$$nF_n - (n+1)F_{n-1} = nf_n - (n-1)f_{n-1}.$$

Mit Hilfe des Rückwärtsdifferenzenoperators (2.1) kann die Rekursion eleganter angeschrieben werden:

$$nF_n - (n+1)F_{n-1} = \nabla n f_n.$$

Dies ist eine lineare Rekursion 1. Ordnung, welche man mit der sogenannten Methode des summierenden Faktors lösen kann. Division durch $n(n+1)$ liefert für F_n nämlich die folgende Rekursion:

$$\frac{F_n}{n+1} = \frac{F_{n-1}}{n} + \frac{nf_n - (n-1)f_{n-1}}{n(n+1)}, \qquad \text{für } n > M+1,$$

aus welcher man durch Iteration sofort folgende Formel generiert:

$$F_n = (n+1)\left(\frac{F_{M+1}}{M+2} + \sum_{M+2 \le k \le n} \frac{\nabla k f_k}{k(k+1)}\right), \qquad \text{für } n > M. \tag{3.3}$$

Indem wir uns die Summe genauer ansehen, können wir das Ergebnis sogar weiter vereinfachen:

$$\begin{aligned}\sum_{M+2\le k\le n}\frac{\nabla k f_k}{k(k+1)} &= \sum_{M+2\le k\le n}\frac{kf_k-(k-1)f_{k-1}}{k(k+1)}\\ &= \sum_{M+2\le k\le n}\frac{f_k-f_{k-1}}{k+1}+\sum_{M+2\le k\le n}\frac{f_{k-1}}{k(k+1)}\\ &= \sum_{M+2\le k\le n}\frac{\nabla f_k}{k+1}+\sum_{M+2\le k\le n}\frac{f_k}{k}-\sum_{M+2\le k\le n}\frac{f_k}{k+1}\\ &= 2\sum_{M+2\le k\le n}\frac{\nabla f_k}{k+1}+\frac{f_{M+1}}{M+2}-\frac{f_n}{n+1}.\end{aligned}$$

Eingesetzt in (3.3) erhält man als alternative Lösung für die Rekursion (3.2):

$$F_n = 2(n+1)\sum_{M+2\le k\le n}\frac{\nabla f_k}{k+1}+\frac{n+1}{M+2}(f_{M+1}+F_{M+1})-f_n, \qquad \text{für } n>M. \tag{3.4}$$

Mit diesen Resultaten ist es uns nun möglich, die erwartete Anzahl der wesentlichen Kenngrößen zu berechnen, indem wir jeweils für f_n geeignet einsetzen.

Partitionierungsschritte:

Für die erwartete Anzahl an Partitionierungsschritten setzen wir $f_k = 1, M+1 \leq k \leq n$, und $F_{M+1} = 1$. Daraus ergibt sich $\nabla f_k = 0$ und wir erhalten direkt

$$A_n = 2\frac{n+1}{M+2} - 1.$$

Vergleiche:

Die Anzahl an Vergleichen während der Partitionierung beträgt $f_k = k+1$, für $M+1 \leq k \leq n$. Dies folgt direkt aus Algorithmus 2.4: Jedes Mal, wenn der Pointer i um 1 erhöht oder der Pointer j um 1 verringert wird, wird ein Vergleich durchgeführt. Der Pointer i startet bei Index 1, der Pointer j bei Index $n+1$. Gestoppt werden beide Pointer sobald $i - 1 = j = s$ für ein Element s aus dem Interall $[1; n]$. Somit wird i genau s-mal erhöht und j genau $(n - s + 1)$-mal verringert. Für die Anzahl an Vergleichen während der Partitionierung von k Elementen erhält man somit $f_k = s + (k - s + 1) = k + 1$.
Daraus ergibt sich $\nabla f_k = 1$, $f_{M+1} = F_{M+1} = M + 2$ und wir erhalten als Lösung

$$C_n = (n+1)(2\mathcal{H}_{n+1} - 2\mathcal{H}_{M+2} + 1).$$

Vertauschungen:

Bevor wir die erwartete Anzahl an Vertauschungen angeben können, muss erst bestimmt werden, mit wie vielen Vertauschungen wir in jedem Partitionierungssschritt rechnen müssen.
Wenn wir annehmen, dass wir als das Pivot-Element das erste Element der Folge A nehmen und $A[1]$ das k-kleinste Element der gesamten Folge ist, dann beschreibt die Anzahl an Vertauschungen genau die Anzahl an Elementen von $A[2], \ldots, A[k]$ die größer als $A[1]$ sind.
Die Wahrscheinlichkeit, dass die Anzahl dieser Elemente, bedingt nach k, mit $1 \leq k \leq n$, genau t beträgt ist hypergeometrisch verteilt und beträgt $\frac{\binom{n-k}{t}\binom{k-1}{k-1-t}}{\binom{n-1}{k-1}}$.
Bilden des Erwartungswertes und Anwenden des Gesetzes der totalen Wahrscheinlich-

keit, um eine unbedingte Wahrscheinlichkeit zu erhalten, liefert:

$$\frac{1}{n}\sum_{1\le k\le n}\sum_{0\le t\le k-1} t\frac{\binom{n-k}{t}\binom{k-1}{k-1-t}}{\binom{n-1}{k-1}}$$
$$= \frac{1}{n}\sum_{1\le k\le n}\frac{n-k}{\binom{n-1}{k-1}}\sum_{0\le t\le k-1}\binom{n-k-1}{t-1}\binom{k-1}{k-1-t}.$$

Durch die Anwendung der Vandermondeschen Identität (2.3) können wir den Term weiter vereinfachen:

$$\begin{aligned}\frac{1}{n}\sum_{1\le k\le n}\frac{n-k}{\binom{n-1}{k-1}}\sum_{0\le t\le k-1}\binom{n-k-1}{t-1}\binom{k-1}{k-1-t} &= \frac{1}{n}\sum_{1\le k\le n}\frac{n-k}{\binom{n-1}{k-1}}\binom{n-2}{k-2}\\ &= \frac{1}{n(n-1)}\sum_{1\le k\le n}(n-k)(k-1)\\ &= \frac{1}{n(n-1)}\left(\frac{1}{6}n(n+1)(2n+1)\right)\\ &= \frac{n-2}{6}.\end{aligned}$$

Auf Grund der Linearität der Rekursionen gilt $B_n = \frac{1}{6}C_n - \frac{1}{2}A_n$ und somit folgt für die erwartete Anzahl an Vertauschungen:

$$B_n = \frac{1}{6}(n+1)\left(2\mathcal{H}_{n+1} - 2\mathcal{H}_{M+2} + 1 - \frac{6}{M+2}\right) + \frac{1}{2}.$$

Einschübe:
Für die Berechnung der Anzahl der Einschübe von Insertionsort verwenden wir, dass $f_k = 0$ für $M+1 \le k \le n$ gilt. Dies liefert somit $D_n = \frac{n+1}{M+2}D_{M+1}$. Den Wert für D_{M+1} berechnet man aus (3.2) durch Einsetzen des Wertes für zufällige Folgen $D_n = n - \mathcal{H}_n,\ n \le M$. Man erhält:

$$D_M = \frac{2}{M+1}\sum_{1\le k\le M+1}(k-1-\mathcal{H}_{k-1}) = M+1-2\mathcal{H}_{M+1},$$

und daraus folgt schlussendlich

$$D_n = (n+1)\left(1 - 2\frac{\mathcal{H}_{M+1}}{M+2}\right).$$

Schlüsselbewegungen während der Einschübe:
Wieder gilt $f_k = 0$ für $M+1 \leq k \leq n$ und somit $E_n = \frac{n+1}{M+2} E_{M+1}$. E_{M+1} berechnet sich erneut aus (3.2) und dem Einsetzen des Wertes für zufällige Folgen. Man erhält:

$$E_{M+1} = \frac{2}{M+1} \sum_{1 \leq k \leq M+1} \frac{(k-1)(k-2)}{4} = \frac{M(M-1)}{6},$$

und daraus folgt schlussendlich

$$E_n = (n+1)\frac{M(M-1)}{6(M+2)}.$$

Stackpushes:
Im Fall $n \leq 2M+2$ wird keine Teilliste auf den Stack gelegt, da eine der beiden Listen garantiert weniger als $M+1$ Elemente enthält. Auch im Fall $n > 2M+2$ findet nur dann ein Stackpush statt, wenn das Pivot-Element im Bereich $M+2$ bis $n-M-1$ liegt. Daraus ergibt sich für die erwartete Anzahl an Stackpushes $\frac{1}{n}(n-2M-2)$. Für $\nabla k f_k$ gilt somit $\nabla k f_k = 1$ und aus (3.3) folgt schlussendlich

$$S_n = \frac{n+1}{2M+3} - 1, \qquad \text{für } n > 2M+1.$$

Die berechneten Werte sind in der folgenden Tabelle zusammengefasst.

$$\begin{aligned}
A_n &= 2\frac{n+1}{M+2} - 1, \\
B_n &= \frac{1}{6}(n+1)\left(2\mathcal{H}_{n+1} - 2\mathcal{H}_{M+2} + 1 - \frac{6}{M+2}\right) + \frac{1}{2}, \\
C_n &= (n+1)(2\mathcal{H}_{n+1} - 2\mathcal{H}_{M+2} + 1), \\
D_n &= (n+1)\left(1 - 2\frac{\mathcal{H}_{M+1}}{M+2}\right), \\
E_n &= (n+1)\frac{M(M+1)}{6(M+2)}, \\
S_n &= \frac{n+1}{2M+3} - 1, \qquad \text{für } n > 2M+1.
\end{aligned}$$

Aus diesen Größen kann man nun auch den Grund für die Schnelligkeit von Quicksort

ableiten. Betrachtet man erneut die Formel (3.1) für die Gesamtkosten:

$$L(n) = 24A_n + 11B_n + 4C_n + 9S_n + 3D_n + 8E_n + 7n,$$

sieht man, dass die Größen mit sehr hohen Erwartungswerten, nämlich B_n und C_n, sehr niedrige Koeffizienten besitzen. Diese Beobachtung liefert eine Erklärung für die relativ kurze Laufzeit. Wie groß die tatsächliche Laufzeit ist, wird in dem folgenden Kapitel angegeben werden, nachdem wir eine optimale Wahl für M getroffen haben.

3.8 Erste Verbesserungen

In diesem Unterkapitel beschäftigen wir uns mit Möglichkeiten, um die Performance von Quicksort direkt zu verbessern ohne den zugrunde liegenden Algorithmus zu verändern. Der Großteil des Kapitels entfällt dabei auf die Analyse der sogenannten Median-of-Three-Verbesserung, einer Heuristik für eine vorteilhaftere Auswahl des Pivot-Elements.

3.8.1 Die richtige Wahl für M

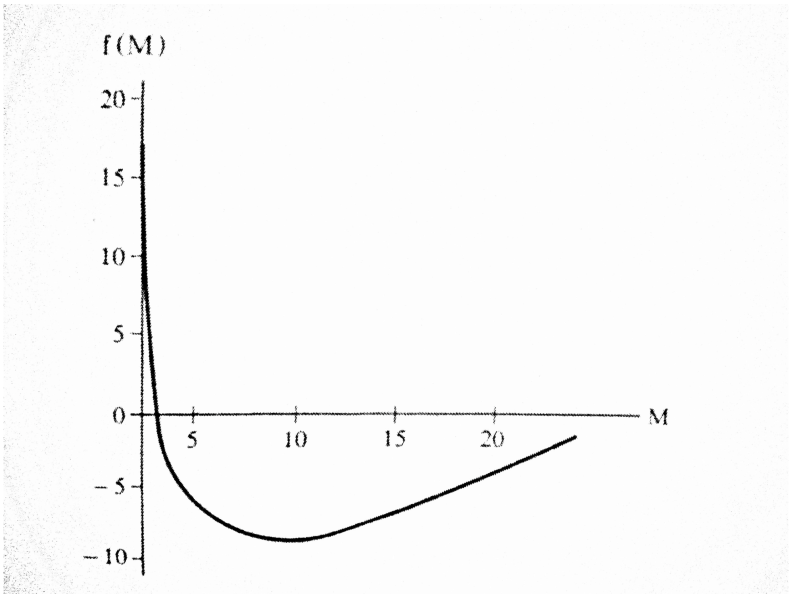
Setzt man die durchschnittlichen Werte nun in die von Knuth bestimmte Gleichung (3.1) für die Laufzeit von Quicksort ein, erhält man für die gesamte durchschnittliche Laufzeit

$$\frac{35}{3}(n+1)\mathcal{H}_{n+1} - \frac{69}{2} + \frac{1}{6}(n+1)\left(8M + 71 - 70\mathcal{H}_{M+2} + \frac{270}{M+2} + \frac{54}{2M+3} - 36\frac{\mathcal{H}_{M+1}}{M+2}\right)$$

Zeiteinheiten.
Um die gesamte Laufzeit mittels M zu minimieren sucht man einfach das lokale Minimum der Gleichung

$$f(M) = \frac{1}{6}(n+1)\left(8M + 71 - 70\mathcal{H}_{M+2} + \frac{270}{M+2} + \frac{54}{2M+3} - 36\frac{\mathcal{H}_{M+1}}{M+2}\right).$$

Die Berechnung mittels geeigneter Software zeigt, dass das gesuchte Minimum bei $M \approx 8,9994$ erreicht wird. Wir werden daher für M den Wert $M = 9$ wählen. Allerdings zeigt der folgende Graph, dass jeder Wert im Intervall $[5; 15]$ durchaus eine akzeptable Wahl für M darstellt.



Für $M = 9$ weist Quicksort eine durchschnittliche Laufzeit von

$$L(n) = 16,6667(n+1)\ln(n) - 1,743n - 19$$

Zeiteinheiten auf [Sed77, Seite 336].

3.8.2 Die Wahl des Pivot-Elements

Da laut unserem Eingabemodell alle Permutationen mit der gleichen Wahrscheinlichkeit auftreten, kann das Pivot-Element beliebig gewählt werden.

3.8.3 Median-of-Three

Eine weitere Möglichkeit, Quicksort zu verbessern, besteht darin, in jedem Rekursionsschritt das Pivot-Element als den Median von drei Elementen zu wählen. Dieses Verhalten liefert eine bessere Ausgewogenheit während des Teilungsprozesses und beeinflusst somit die durchschnittliche Laufzeit positiv.
Das Auffinden des Medians muss in Algorithmus 4 direkt nach der if-Abfrage über die Eingabegröße in Zeile 3 erfolgen. Eine mögliche Implementierung wird von Sedgewick

[Sed77, Seite 336] folgendermaßen angegeben:

if $A[l] > A[r]$ **then** $exchange(A[l], A[r])$ **endif**;
if $A[(l+r)/2] > A[r]$ **then** $exchange(A[(l+r)/2], A[r])$ **endif**;
if $A[l] > A[(l+r)/2]$ **then** $exchange(A[l], A[(l+r)/2])$ **endif**;
$exchange(A[l+1], A[(l+r)/2])$;

Als Median erhält man $A[l+1]$. Als zusätzliche kleine Verbesserung können im Algorithmus 4 die Pointer auf $i := l+1$, $j := r$ gesetzt werden.
In der Praxis ist es durchaus von Vorteil, das Element $A[(l+r)/2]$ statt eines beliebigen zu verwenden. Dieser Schritt verringert im Fall von teilsortierten Folgen die Wahrscheinlichkeiten für den Worst-Case und das Verfahren liefert auch in diesen Situationen gute Ergebnisse.

Mit dieser Verbesserung ergibt sich der folgende Pseudocode:

Algorithm 6 Quicksort with Median-of-Three

```
1:  procedure QUICKSORT(integer l,r)
2:      if r − l ≥ M then
3:          if A[l] > A[r] then exchange(A[l], A[r]); end if
4:          if A[(l + r)/2] > A[r] then exchange(A[(l + r)/2], A[r]); end if
5:          if A[l] > A[(l + r)/2] then exchange(A[l], A[(l + r)/2]); end if
6:          exchange(A[l + 1], A[(l + r)/2]);
7:          i := l + 1; j := r; v := A[l + 1];
8:          do
9:              do
10:                 i = i + 1;
11:             while A[i] < v
12:             do
13:                 j = j − 1;
14:             while A[j] > v
15:             exchange(A[i], A[j]);
16:         while i < j
17:         exchange(A[l], A[j]);
18:         if j − l < r − i + 1 then
19:             quicksort(l, j − 1); quicksort(i, r);
20:         else
21:             quicksort(i, r); quicksort(l, j − 1);
22:         end if
23:     end if
```

3.8.4 Analyse der Median-of-Three Verbesserung

Die laufzeitbestimmenden Größen der Median-of-Three Verbesserung sind selbstverständlich dieselben Größen wie bei der Analyse der Grundform von Quicksort. Da das Finden des Medians in jedem Partitionierungs-Schritt sehr aufwändig ist, wächst hier der Koeffizient von A in der Berechnung der totalen Laufzeit von 24 auf durchschnittlich $53,5$ [Sed77, Seite 337]. Die totale Laufzeit beträgt somit

$$L(n) = 53,5A_n + 11B_n + 4C_n + 9S_n + 3D_n + 8E_n + 7n$$

Zeiteinheiten. Es wird nun gezeigt, dass sich dieser zusätzliche Aufwand für das Finden des Medians auszahlt. Dabei folgen wir der Analyse von [Sed77, ab Seite 337].

Die Wahrscheinlichkeit für das k-kleinste Element, das Pivot-Element zu sein, beträgt $(n-k)(k-1)/\binom{n}{3}$. Daher genügen sämtliche Kenngrößen der folgenden Rekursion:

$$F_n = f_n + \sum_{1\le k\le n} \frac{(n-k)(k-1)}{\binom{n}{3}}(F_{k-1}+F_{n-k}), \quad \text{für } n > M \ge 3. \tag{3.5}$$

Hier bezeichnet f_n wieder den durchschnittlichen Wert der jeweiligen Kenngröße im ersten Teilungsschritt und F_n den erwarteten Wert einer zufälligen n-elementigen, zu sortierenden Folge.

Aus der Beobachtung, dass sich der Faktor von F_k folgendermaßen ergibt: $(n-k-1)k+k(n-k-1) = 2(n-k-1)k$, erreicht man nach der Multiplikation mit $\binom{n}{3}$ folgende vereinfachte Darstellung der Rekursion (3.5):

$$\binom{n}{3}F_n = \binom{n}{3}f_n + 2\sum_{1\le k<n}(n-k-1)kF_k, \quad \text{für } n > M \ge 3. \tag{3.6}$$

Im Fall $n \le M$ nehmen wir für Quicksort wieder $F_n = 0$ an, für die Berechnung der Größen D_n und E_n gilt erneut $f_n = 0$.
Daraus ergeben sich unterschiedliche Rekursionen für die Kenngrößen von Quicksort und Insertionsort, nämlich

$$\binom{n}{3}F_n = \binom{n}{3}f_n + 2\sum_{M<k<n}(n-k-1)kF_k, \quad \text{für } n > M \ge 3,$$

für Quicksort und

$$\binom{n}{3}F_n = 2\sum_{1\le k<n}(n-k-1)kF_k, \quad \text{für } n > M \ge 3,$$

für Insertionsort.
Definieren wir nun f_n für die Kenngrößen von Insertionsort als $\binom{n}{3}f_n = 2\sum_{0\le k\le M}(n-$

$k-1)kF_k$, erhalten wir eine Rekursion, die für alle Kenngrößen gültig ist:

$$\binom{n}{3}F_n = \binom{n}{3}f_n + 2\sum_{M<k<n}(n-k-1)kF_k, \quad \text{für } n > M \geq 3. \tag{3.7}$$

Um die Rekursion (3.7) zu lösen, wird die Methode der erzeugenden Funktionen verwendet. Zuvor wird der Binomialkoeffizient und die Gleichung mit dem daraus resultierenden Nenner erweitert. Man erhält

$$n(n-1)(n-2)F_n = n(n-1)(n-2)f_n + 12\sum_{M<k<n}(n-k-1)kF_k.$$

Multiplikation auf beiden Seiten mit z^{n-3} und Summation über n führt zu eine Differentialgleichung 3. Ordnung:

$$F'''(z) = f'''(z) + 12\frac{F'(z)}{(1-z)^2},$$

wobei $F(z) = \sum_{n>M} F_n z^n$ und $f(z) = \sum_{n>M} f_n z^n$ gilt. Um diese Gleichung zu lösen, werden beide Seiten der Gleichung mit dem Faktor $(1-z)^3$ erweitert. Man erhält

$$(1-z)^3F'''(z) = (1-z)^3f'''(z) + 12(1-z)F'(z). \tag{3.8}$$

Für die vorliegende Differentialgleichung (3.8) gilt nun, dass in jedem Term der Grad und die Ordnung übereinstimmen. Diese Arten von Gleichungen werden in der Literatur Euler-Differentialgleichungen genannt und können auf verschiedene Arten gelöst werden. Einerseits durch geeignete Substitution oder aber durch das Einführen eines neues Operators, der sowohl multipliziert als auch differenziert.
In diesem konkreten Fall bietet sich letztere Methode an, deswegen führen wir einen Operator θ ein, für den folgendes gilt:

$$\theta F(z) := -(1-z)F'(z)$$

Angewandt auf unsere Differentialgleichung ergibt sich

$$-\theta(\theta-1)(\theta-2)F(z) = -12\theta F(z) + (1-z)^3f'''(z),$$

was sich zu

$$-\theta(-2-\theta)(5-\theta)F(z) = (1-z)^3f'''(z)$$

vereinfachen lässt. Man erhält nun eine Lösung für $F(z)$ und F_n indem man die folgenden Differentialgleichungen erster Ordnung löst:

$$\begin{aligned} -\theta U(z) &= (1-z)^3 f'''(z), \\ (-2-\theta)T(z) &= U(z), \\ (5-\theta)F(z) &= T(z). \end{aligned}$$

Definiert man nun $U(z) = \sum_{n>M} U_n z^n$ und $T(z) = \sum_{n>M} T_n z^n$ wieder als erzeugende Funktionen, ergibt sich für die erste Gleichung

$$\begin{aligned} -\theta U(z) &= (1-z)^3 f'''(z), \\ (1-z)\left(\sum_{n>M} U_n z^n\right)' &= (1-3z+3z^2-z^3)\left(\sum_{n>M} f_n z^n\right)''', \\ (1-z)\left(\sum_{n>M+1} n U_n z^{n-1}\right) &= (1-3z+3z^2-z^3)\left(\sum_{n>M+3} n(n-1)(n-2) f_n z^{n-3}\right), \\ \sum_{n>M}(n+1)U_{n+1}z^n - \sum_{n>M} n U_n z^n &= 6\sum_{n>m} \nabla^3 f_{n+3}\binom{n+3}{3} z^n. \end{aligned}$$

Ablesen der Koeffizienten liefert nun folgende Rekursion für U_n:

$$(n+1)U_{n+1} = nU_n + 6\nabla^3 f_{n+3}\binom{n+3}{3}. \tag{3.9}$$

Hierbei bezeichnet ∇ abermals den Rückwärtsdifferenzenoperator.

Die verbleibenden Differentialgleichungen lassen sich auf die gleiche Art lösen und man erhält als Ergebnis

$$(n+1)T_{n+1} = (n+2)T_n + U_n, \tag{3.10}$$

$$(n+1)F_{n+1} = (n-5)F_n + T_n, \quad n > M. \tag{3.11}$$

Einsetzen für n und die Bedingung, dass $f_n = 0$ für $n \leq M$ gilt, liefert für die erste Rekursion (3.9) folgende Vereinfachung:

$$nU_n = 6\nabla^2 f_{n+2}\binom{n+2}{3}.$$

Um eine Lösung für die zweite Rekursion (3.10) zu erhalten, wird zuerst durch den Faktor $(n+1)(n+2)$ dividiert. Man erhält danach auf gleichem Weg wie zuvor

$$\frac{T_n}{n+1} = \frac{T_{M+1}}{M+2} + \sum_{M+3\le k\le n+1} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}}.$$

Für $n > 5$ vereinfacht sich die Rekursion (3.11) für F_n nach Multiplikation mit $\frac{1}{6}\binom{n}{5}$ zu

$$\begin{aligned}\binom{n}{6} F_n =& \binom{M+1}{6} F_{M+1} + \frac{T_{M+1}}{M+2} \sum_{M+2\le j\le n} \binom{j}{6} \\ &+ \sum_{M+2\le j\le n} \binom{j}{6} \sum_{M+3\le k\le j} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}}.\end{aligned}$$

Die Summe im zweiten Term vereinfacht sich durch die Anwendung von (2.7) zu $\binom{n+1}{7} - \binom{M+2}{7}$, die Werte von F_{M+1} und T_{M+1} erhält man durch Einsetzen in (3.7). Das Anwenden dieser Vereinfachungen liefert

$$\begin{aligned}\binom{n}{6} F_n =& \binom{M+1}{6} f_{M+1} + \left(\nabla f_{M+2} + \frac{6}{M+2} f_{M+1}\right)\left(\binom{n+1}{7} - \binom{M+2}{7}\right) \\ &+ \sum_{M+3\le k\le n} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}} \left(\binom{n+1}{7} - \binom{k}{7}\right), \quad n > \max(M, 5). \end{aligned} \tag{3.12}$$

Durch Aufteilen der Summe und unter Berücksichtigung von $f_n = 0$, für $n \leq M$, erreicht man

$$\begin{aligned}
&\sum_{M+3\leq k\leq n} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}} \left(\binom{n+1}{7} - \binom{k}{7}\right) \\
&= \overbrace{\sum_{M+1\leq k\leq n} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}} \left(\binom{n+1}{7} - \binom{k}{7}\right)}^{=:\rho} - \frac{\nabla^2 f_{M+2} \binom{M+2}{3}}{\binom{M+2}{3}} \left(\binom{n+1}{7} - \binom{M+2}{7}\right) \\
&\quad - \frac{\nabla^2 f_{M+1} \binom{M+1}{3}}{\binom{M+1}{3}} \left(\binom{n+1}{7} - \binom{M+1}{7}\right) \\
&= \rho - \left(f_{M+2} - 2f_{M+1}\frac{M-1}{M+2}\right)\left(\binom{n+1}{7} - \binom{M+2}{7}\right) - f_{M+1}\left(\binom{n+1}{7} - \binom{M+1}{7}\right) \\
&= \rho - \left(\nabla f_{M+2} + \frac{6f_{M+1}}{M+2}\right)\left(\binom{n+1}{7} - \binom{M+2}{7}\right) \\
&\quad + f_{M+1}\left(\binom{n+1}{7} - \binom{M+2}{7}\right) - f_{M+1}\left(\binom{n+1}{7} - \binom{M+1}{7}\right) \\
&= \rho - \left(\nabla f_{M+2} + \frac{6f_{M+1}}{M+2}\right)\left(\binom{n+1}{7} - \binom{M+2}{7}\right) - \binom{M+1}{6} f_{M+1}.
\end{aligned}$$

Einsetzen in (3.12) liefert

$$\binom{n}{6} F_n = \sum_{M+1\leq k\leq n} \frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}} \left(\binom{n+1}{7} - \binom{k}{7}\right).$$

Man kann diesen Term nun noch weiter vereinfachen, indem man den Rückwärtsdifferenzenoperator auflöst und den Koeffizienten von f_k direkt angibt.
Wir betrachten $\frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}}$. Auflösen des Rückwärtsdifferenzenoperators liefert:

$$\begin{aligned}
\frac{\nabla^2 f_k \binom{k}{3}}{\binom{k}{3}} &= \frac{f_k \binom{k}{3}}{\binom{k}{3}} - 2\frac{f_{k-1}\binom{k-1}{3}}{\binom{k}{3}} + \frac{f_{k-2}\binom{k-2}{3}}{\binom{k}{3}} \\
&= f_k - 2\frac{f_{k-1}\binom{k-1}{3}}{\binom{k}{3}} + \frac{f_{k-2}\binom{k-2}{3}}{\binom{k}{3}}.
\end{aligned}$$

Hier wird klar, dass sowohl $\nabla^2 f_{k+2}$ als auch $\nabla^2 f_{k-1}$ einen Beitrag zu dem Koeffizienten von f_k liefern.

Bezeichnet man nun diesen Koeffizienten mit a_k, ergibt sich für diesen:

$$\begin{aligned} a_k =& \left(\binom{n+1}{7} - \binom{k}{7}\right) - 2\frac{k-2}{k+1}\left(\binom{n+1}{7} - \binom{k+1}{7}\right) \\ &+ \frac{(k-1)(k-2)}{(k+2)(k+1)}\left(\binom{n+1}{7} - \binom{k+2}{7}\right) \\ =& \frac{12}{(k+2)(k+1)}\left(\binom{n+1}{7} - \binom{k+2}{7}\right). \end{aligned}$$

Man erhält somit die folgende, erstaunlich einfache, Gleichung:

$$F_n = f_n + \frac{1}{\binom{n}{6}} \sum_{M<k<n} \frac{12 f_k}{(k+2)(k+1)}\left(\binom{n+1}{7} - \binom{k+2}{7}\right), \quad n > \max(M, 5). \tag{3.13}$$

Das Aufteilen der Summe ergibt für den ersten Term:

$$\begin{aligned} \frac{1}{\binom{n}{6}} \sum_{M<k<n} \frac{12 f_k}{(k+2)(k+1)}\binom{n+1}{7} &= \frac{12}{7}(n+1)\left(\sum_{M<k<n} \frac{f_k}{k+1} - \frac{f_k}{k+2}\right) \\ &= \frac{12}{7}(n+1)\left(\frac{f_{M+1}}{M+2} + \sum_{M<k<n} \frac{f_{k+1}}{k+2} - \frac{f_n}{n+1} - \sum_{M<k<n} \frac{f_k}{k+2}\right) \\ &= \frac{12}{7}(n+1)\left(\frac{f_{M+1}}{M+2} - \frac{f_n}{n+1} + \sum_{M<k<n} \frac{f_{k+1} - f_k}{k+2}\right) \\ &= \frac{12}{7}(n+1)\left(\frac{f_{M+1}}{M+2} - \frac{f_n}{n+1} + \sum_{M+1<k<n+1} \frac{\nabla f_k}{k+1}\right). \end{aligned}$$

Für den zweiten Term folgt sofort:

$$\frac{1}{\binom{n}{6}} \sum_{M<k<n} \frac{12 f_k}{(k+2)(k+1)}\left(\binom{k+2}{7}\right) = \frac{2}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} f_k \binom{k}{5}.$$

Einsetzen in die Rekursion (3.7) liefert nun

$$\begin{aligned} F_n =& \frac{12}{7}\frac{n+1}{M+2} f_{M+1} - \frac{5}{7} f_n + \frac{12}{7}(n+1) \sum_{M+1<k<n+1} \frac{\nabla f_k}{k+1} \\ &- \frac{2}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} f_k \binom{k}{5}. \end{aligned} \tag{3.14}$$

Mit dieser Lösung für die Rekursion (3.7) können nun durch Einsetzen für f_n die Werte für die durchschnittliche Anzahl der Partitionierungsschritte, Vertauschungen, Vergleiche, Einschübe, Bewegungen während der Einschübe und Stackpushes bestimmt werden.

Partitionierungsschritte:
Für die durchschnittliche Anzahl der Partitionierungsschritte setzt man $f_k = 1$, $M+1 \leq k \leq n$, und erhält

$$\begin{aligned} A_n &= \frac{12}{7}\frac{n+1}{M+2} - \frac{5}{7} - \frac{2}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} \binom{k}{5} \\ &= \frac{12}{7}\frac{n+1}{M+2} - \frac{5}{7} - \frac{2}{7}\frac{1}{\binom{n}{6}} \left(\binom{n}{6} - \binom{M+1}{6} \right) \\ &= \frac{12}{7}\frac{n+1}{M+2} - 1 + O(n^{-6}). \end{aligned}$$

Vergleiche:
Für die durchschnittliche Anzahl an Vergleichen setzt man $f_n = n - 1$ und für ∇f_k ergibt sich somit $\nabla f_k = 1$. Der Grund, dass man an dieser Stelle $f_n = n - 1$ setzt und nicht wieder $f_n = n + 1$, wie bei der ursprünglichen Variante, ist der, dass sich durch das Auffinden des Medians die Elemente $A[1]$ und $A[n]$ bereits in den jeweils richtigen Hälften der Eingabe befinden und deswegen, während der Partitionierung, nicht mehr berücksichtigt werden müssen.
Die Berechnung kann allerdings wieder vereinfacht werden, indem man die Linearität der Rekursion ausnützt. Zu diesem Zweck unterscheiden wir zwischen C_n, der Anzahl an Vergleichen während der Partitionierung, und einem C_n^* mit $f_n^* = n + 1$. Man erhält

C_n danach durch $C_n = C_n^* - 2A_n$. Dies liefert:

$$\begin{aligned}
C_n =& \frac{12}{7}(n+1) - \frac{5}{7}(n+1) + \frac{12}{7}(n+1) \sum_{M+1<k<n+1} \frac{1}{k+1} - \frac{2}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} (k+1)\binom{k}{5} \\
& - \frac{24}{7}\frac{n+1}{M+2} + 2 \\
=& (n+1) + \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) - \frac{12}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} \binom{k+1}{6} - \frac{24}{7}\frac{n+1}{M+2} + 2 \\
=& \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + (n+1) - \frac{12}{7}\frac{1}{\binom{n}{6}} \sum_{M<k<n} \binom{k+2}{7} - \binom{k+1}{7} - \frac{24}{7}\frac{n+1}{M+2} + 2 \\
=& \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + (n+1) - \frac{12}{7}\frac{1}{\binom{n}{6}} \left(\binom{n+1}{7} - \binom{M+2}{7} \right) - \frac{24}{7}\frac{n+1}{M+2} + 2 \\
=& \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \frac{37}{49}(n+1) + \frac{12}{7}\frac{\binom{M+2}{7}}{\binom{n}{6}} - \frac{24}{7}\frac{n+1}{M+2} + 2 \\
=& \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{\mathcal{M}+\in}) + \frac{37}{49}(n+1) - \frac{24}{7}\frac{n+1}{M+2} + 2 + \mathcal{O}(n^{-6}).
\end{aligned}$$

Vertauschungen:

Für die durchschnittliche Anzahl an Vertauschungen muss zuerst die durchschnittliche Anzahl von Vertauschungen während des ersten Partitionierungsschrittes berechnet werden.

Unter der Annahme, dass das zweite Element $A[2]$ der Liste als Pivot-Element gewählt wird, $A[1]$ kleiner als $A[2]$ ist und $A[2]$ das k-kleinste Element ist, ergibt sich folgende Berechnung.

Die gesuchte Anzahl an Vertauschungen ist genau die Anzahl der Elemente aus $A[3], \ldots, A[k]$, die größer als $A[2]$ sind. Von diesen Elementen existieren genau t mit der Wahrscheinlichkeit

$$\frac{\binom{n-k-1}{t}\binom{k-2}{k-2-t}}{\binom{n-3}{k-2}}.$$

Für die durchschnittliche Anzahl an Vertauschungen ergibt sich somit

$$\begin{aligned}
\sum_{0\le t\le k-2} t\frac{\binom{n-k-1}{t}\binom{k-2}{k-2-t}}{\binom{n-3}{k-2}} &= \frac{1}{\binom{n-3}{k-2}} \sum_{t=0}^{k-2} (n-k-1)\binom{n-k-2}{t-1}\binom{k-2}{k-2-t} \\
&= \frac{(n-k-1)}{\binom{n-3}{k-2}} \sum_{t=0}^{k-3} \binom{n-k-2}{t}\binom{k-2}{k-3-t}.
\end{aligned}$$

Die Anwendung der Vandermondeschen Identität (2.2) führt schließlich zu

$$\frac{(n-k-1)}{\binom{n-3}{k-2}}\binom{n-4}{k-3} = \frac{(n-k-1)(k-2)}{n-3}.$$

Bildet man nun den Durchschnitt über alle möglichen Pivot-Elemente, erhält man für die erwartete Anzahl an Vertauschungen im ersten Partitionierungsschritt

$$\begin{aligned}\sum_{k=1}^{n} \frac{(k-1)(n-k)}{\binom{n}{3}} \frac{(n-k-1)(k-2)}{n-3} &= \frac{1}{\binom{n}{4}} \sum_{k=1}^{n} \frac{(n-k)(n-k-1)}{2} \frac{(k-1)(k-2)}{2} \\ &= \frac{1}{\binom{n}{4}} \sum_{k=1}^{n} \binom{n-k}{2}\binom{k-1}{2} \\ &= \frac{\binom{n}{5}}{\binom{n}{4}} = \frac{n-4}{5}.\end{aligned}$$

Die vorletzte Gleichheit folgt direkt aus (2.4). Indem man die Linearität der Rekursion ausnützt, ergibt sich aus

$$\frac{n-4}{5} = \frac{1}{5}(n-1) - \frac{3}{5}1,$$

für B_n:

$$B_n = \frac{1}{5}C_n - \frac{3}{5}A_n$$

und weiter:

$$B_n = \frac{12}{35}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \frac{37}{245}(n+1) - \frac{12}{7}\frac{n+1}{M+2} + 1.$$

Einschübe:
Die Anzahl der Einschübe, die Insertionsort benötigt, lässt sich am besten mit Hilfe von (3.12) berechnen da, in diesem Fall $\nabla^2 f_k\binom{k}{3} = 0$ gilt.
Es folgt somit direkt nach der Division von $\binom{n}{6}$:

$$D_n = \left(\nabla f_{M+2} + \frac{6}{M+2} f_{M+1}\right)\frac{n+1}{7} + \mathcal{O}(n^{-6}).$$

Dasselbe gilt später auch für E_n.
Die Werte für f_{M+2} und f_{M+1} folgen direkt aus $\binom{n}{3} f_n = 2\sum_{0\le k\le M}(n-k-1)kF_k$. Somit gilt für f_{M+1} und f_{M+2}:

$$f_{M+1} = \frac{2}{\binom{M+1}{3}} \sum_{0\le k\le M} (M-k)k(k-\mathcal{H}_k) = M+1-2\mathcal{H}_{M+1}+\frac{2}{3},$$
$$f_{M+2} = M+2-2\mathcal{H}_{M+2}+\frac{2}{3}.$$

Daraus ergibt sich sofort $\nabla f_{M+2} = 1-\frac{2}{M+2}$ und wir erhalten:

$$D_n = n+1-\frac{4}{7}\frac{n+1}{M+2}(3\mathcal{H}_{M+1}-1).$$

Schlüsselbewegungen während der Einschübe:
Wir wissen bereits, dass

$$E_n = \left(\nabla f_{M+2} + \frac{6}{M+2} f_{M+1}\right)\frac{n+1}{7} + \mathcal{O}(n^{-6})$$

gilt. Für f_{M+1} und f_{M+2} folgt auf demselben Weg wie zuvor:

$$f_{M+1} = \frac{1}{20}(3M^2-5M-2),$$
$$f_{M+2} = \frac{1}{20}(3M^2+M-4).$$

Daraus ergibt sich sofort $\nabla f_{M+2} = \frac{1}{20}(6M-2)$ und somit

$$E_n = \frac{1}{35}(n+1)(M-17)+\frac{6}{7}\frac{n+1}{M+2}.$$

Stackpushes:
Die durchschnittliche Anzahl an Stackpushes während des ersten Partitionierungsschrittes ist genau die Wahrscheinlichkeit, dass das gewählte Pivot-Element zwischen $M+2$ und $n-M-1$ liegt:

$$f_n = \frac{1}{\binom{n}{3}} \sum_{M+2\le k\le n-M-1} (k-1)(n-k).$$

Die Rekursion (3.7) ist gültig für $n > 2M+1$, daher ersetzen wir in Rekursion (3.7) alle M mit $2M+1$. Da $\nabla^2\binom{k}{3} f_k = k-2$ gilt, verwenden wir wieder (3.12) und erhalten,

nachdem wir $f_{2M+2} = 0$ und $f_{2M+3} = \frac{(M+1)^2}{\binom{2M+3}{3}}$ eingesetzt haben, direkt die Lösung

$$S_n = \frac{3}{7}(n+1)\frac{5M+3}{(2M+3)(2M+1)} - 1.$$

Die berechneten Werte sind in der folgenden Tabelle nochmals zusammengefasst:

$$\begin{aligned}
A_n &= \frac{12}{7}\frac{n+1}{M+2} - 1,\\
B_n &= \frac{12}{35}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \frac{37}{245}(n+1) - \frac{12}{7}\frac{n+1}{M+2} + 1,\\
C_n &= \frac{12}{7}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \frac{37}{49}(n+1) - \frac{24}{7}\frac{n+1}{M+2} + 2,\\
D_n &= (n+1) - \frac{4}{7}\frac{n+1}{M+2}(3\mathcal{H}_{M+1} + 1),\\
E_n &= \frac{1}{35}(n+1)(M-17) + \frac{6}{7}\frac{n+1}{M+2},\\
S_n &= \frac{3}{7}(n+1)\frac{5M+3}{(2M+3)(2M+1)} - 1.
\end{aligned}$$

Bemerkenswert an dieser Stelle ist, dass, bis auf die Werte der Partitionierungsschritte und Vergleiche, die anderen Werte alle höher als bei der ursprünglichen Quicksort Variante sind.

Für die durchschnittliche totale Laufzeit haben wir zu Beginn $53,5A_n + 11B_n + 4C_n + 8D_n + 8E_n + 9S_n + 7n$ angegeben. Setzt man nun die gerade erhaltenen Werte ein, erhält man folgende exakte Formel für die Laufzeit $L(n)$:

$$\begin{aligned}
L(n) = {} & \frac{372}{35}(n+1)\mathcal{H}_{n+1} - \frac{111}{2}\\
& + \frac{1}{7}(n+1)\left(\frac{48}{5}M + \frac{529}{7} - \frac{372}{5}\mathcal{H}_{M+2} + \frac{450}{M+2} + \frac{27(5M+3)}{(2M+3)(2M+1)} - \frac{36\mathcal{H}_{M+2}}{M+2}\right).
\end{aligned}$$

Dies gilt für alle $n > 2M+1$, für $M < n \leq 2M-1$ muss $9S_n$ subtrahiert werden. Die ideale Wahl für M ist, wie man leicht zeigen kann, erneut 9, und für dieses M ergibt sich eine totale Laufzeit von

$$10,6286(n+1)\ln n + 2,116n - 71$$

Zeiteinheiten [Sed77, Seite 341]. Vergleicht man diese Laufzeit mit $11,6667(n+1)\ln n + 1,743n - 19$, der Laufzeit des klassischen Quicksort, ergibt sich für $n = 10000$ eine Verbesserung von etwa 5% und asymptotisch, für $n \to \infty$ in etwa $8,8\%$.
Obwohl also die Verbesserung für sehr große n theoretisch bis zu $8,8\%$ betragen kann, beträgt sie aber in der Praxis nur etwa 6%.

3.8.5 Größere Stichproben

Man könnte nun annehmen, dass der Median einer größeren Stichprobe weitere Verbesserungen liefert. Erweitert man das Verfahren auf $(2t+1)$-elementige Stichproben, dann muss für die Analyse der Anzahl der Vergleiche folgende Rekursion gelöst werden:

$$F_n = f_n + \sum_{1 \le k \le n} \frac{\binom{n-k}{t}\binom{k-1}{t}}{\binom{n}{2t+1}}(F_{k-1} + F_{n-k}), \quad \text{für } n > M \ge 2t+1.$$

Im Fall, dass $f_n = n + \mathcal{O}(1)$ gilt, erhält man als Lösung:

$$F_n = \frac{1}{\mathcal{H}_{2t+2} - \mathcal{H}_{t+1}}(n+1)\mathcal{H}_{n+1} + \mathcal{O}(n).$$

Dieses Resultat wurde als erstes von van Emden in [vE70] erzielt. Die durchschnittliche Anzahl an Vertauschungen im ersten Partitionierungs-Schritt beträgt $\frac{t+1}{4t+6}n + \mathcal{O}(1)$ [Sed77](Seite 341).

Für die Laufzeit einer Median-of-$(2t+1)$ Implementierung von Quicksort erhält man

$$L(n) = \frac{4 + 11\left(\frac{t+1}{4t+6}\right)}{H_{2t+2} - H_{t+1}} n \ln n + \mathcal{O}(n),$$

[Sed77](Seite 341). Ausgewertet ergibt der Koeffizient gerundet $11,67; 10,66; 10,31; 10,12$ für $t = 0, 1, 2, 3$ und nähert sich langsam dem Grenzwert $\frac{27}{4}\ln 2 \approx 9,73$ an.

Die größte Verbesserung ergibt sich somit für $t = 1$. Größere Stichproben sind höchstwahrscheinlich nicht rentabel, da der Aufwand zum Finden des Medians, versteckt im Term $\mathcal{O}(n)$, die geringe Verbesserung kompensiert.

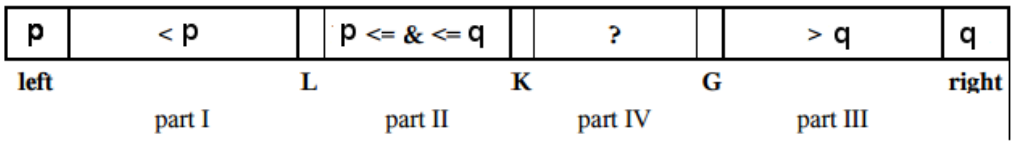
3.8.6 Multi-Pivot-Quicksort

Die klassische Variante von Quicksort mit der Wahl eines Pivot-Elements und der rekursiven Anwendung auf die zwei entstanden Teillisten, kann leicht verallgemeinert werden. Durch die Wahl von $s-1$ Pivot-Elementen entstehen s Teillisten, auf die das Vorgehen wieder angewendet werden kann. Für besonderes Aufsehen sorgte im Jahr 2009 Yaroslavskiy mit der Publikation einer Quicksortvariante mit zwei Pivot-Elementen [Yar09], welche deutlich schneller als die klassische Variante ist.
Eine Quicksort-Variante welche 3 Pivot-Elemente wählt, wurde Ende 2013 von [KLOMQ14] analysiert, wird aber in dieser Arbeit nicht berücksichtigt.

4 Average-Case Analyse von Dual-Pivot Quicksort

Dual-Pivot Quicksort ist eine Familie von Sortieralgorithmen, die sehr nahe mit der klassischen Quicksort-Familie verwandt ist. Um eine Sequenz von paarweise verschiedenen Zahlen in zufälliger Reihenfolge zu sortieren, wird folgendermaßen vorgegangen.
Gilt für die Anzahl n der Elemente $n \leq 1$ sind wir fertig. Für $n \geq 2$ werden zufällig zwei Elemente p und q als Pivot-Elemente gewählt. Sei o.B.d.A. $p < q$.
Als nächstes kommt der Partitionierungs-Schritt. Während diesem Schritt wird die Sequenz in die folgenden drei Kategorien zerlegt:

- Die kleinen Elemente (kleiner als p),
- Die mittleren Elemente (größer als p aber kleiner als q),
- Die großen Elemente (größer als q).



Nach dieser Einteilung erfolgt ein rekursiver Aufruf für jede der entstandenen Teilmengen. Die Pivot-Elemente p und q befinden sich wie bei der klassischen Quicksort-Variante bereits an der richtigen Position.

In dieser Arbeit werden zwei Partitionierungsvarianten von Dual-Pivot Quicksort betrachtet. Als erstes Yaroslavskiy's Version aus dem Jahre 2009 [Yar09] und danach eine Variante von Sedgewick [Sed75, Programm 5.1 auf Seite 151] die bereits 1975 publiziert wurde.
Beide Algorithmen werden nun kurz in Pseudocode präsentiert, bevor anschließend die Rekursion für das Verfahren gelöst wird.

4.1 Die Algorithmen

4.1.1 Yaroslavskiy's Algorithmus in Pseudocode

Algorithm 7 Dual Pivot Quicksort mit Yaroslavskiys Partitionierung

```
1:  procedure DualPivotQuicksortYaroslavskiy(A,left,right)
2:      if right − left < M then
3:          Insertionsort(A,left, right);
4:      else
5:          p := A[left]; q := A[right];
6:          if p>q then
7:              exchange(p, q);
8:          end if
9:          l := left + 1; g := right − 1; k := l;
10:         while k ≤ g do
11:             if A[k] < p then
12:                 exchange(A[k], A[l]);
13:                 l := l + 1;
14:             else
15:                 if A[k] ≥ q then
16:                     while A[g] > q and k < g do
17:                         g := g − 1;
18:                     end while
19:                     exchange(A[k], A[l]);
20:                     g := g − 1;
21:                     if A[k] < p then
22:                         exchange(A[k], A[l]);
23:                         l := l + 1;
24:                     end if
25:                 end if
26:             end if
27:             k := k + 1;
28:         end while
29:         l := l − 1; g := g + 1;
30:         A[left] := A[l]; A[l] := p; //Swap pivot elements to final position
31:         A[right] := A[g]; A[g] := q;
32:         DualPivotQuicksortYaroslavskiy(A, left, l − 1);
33:         DualPivotQuicksortYaroslavskiy(A, l + 1, g − 1);
34:         DualPivotQuicksortYaroslavskiy(A, g + 1, right);
35:     end if
```

4.1.2 Sedgewicks Algorithmus in Pseudocode

Algorithm 8 Dual-Pivot Quicksort mit Sedgewicks Partitionierung

```
1:  procedure DualPivotQuicksortSedgewick(A,links,rechts)
2:      if right − left ≥ M then
3:          i := left; i_1 := left;
4:          j := right; j_1 := right;
5:          if A[left] > A[right] then exchange(A[left], A[right]);
6:          p := A[left]; q := A[right];
7:          while true do
8:              i := i + 1
9:              while A[i] ≤ q do
10:                 if i ≥ j then break outer while //pointers crossed
11:                 if A[i] < p then
12:                     A[i_1] := A[i];
13:                     i_1 := i_1 + 1 ;
14:                     A[i] := A[i_1];
15:                 end if
16:                 i := i + 1;
17:             end while
18:             j := j − 1;
19:             while A[j] ≥ p do
20:                 if A[j] > q then
21:                     A[j_1] := A[j];
22:                     j = j + 1;
23:                     A[j] := A[j + 1];
24:                 end if
25:                 if i ≥ j then break outer while //pointers crossed
26:                 j := j − 1;
27:             end while
28:             A[i_1] := A[j]; A[j_1] := A[i];
29:             i_1 := i_1 + 1; j_1 := j_1 − 1;
30:             A[i] := A[i_1]; A[j] := A[j_1];
31:         end while
32:         A[i_1] := p
33:         A[j_1] := q
34:         DualPivotQuicksortSedgewick(A, left, i_1 − 1);
35:         DualPivotQuicksortSedgewick(A, i_1 + 1, j_1 − 1);
36:         DualPivotQuicksortSedgewick(A, j_1 + 1, right);
37:     end if
```

4.2 Aufstellen der Rekursion

In diesem Abschnitt wird eine allgemeine Lösung für die Rekursion von Dual-Pivot Quicksort erarbeitet. Mit F_n werden die erwarteten Kosten für das Sortieren einer zufälligen Permutation von $1, \ldots, n$ bezeichnet.
Es gilt die folgende wichtige Eigenschaft:

> Jeder Vergleich beinhaltet ein Pivot-Element des aktuellen Teilungsschrittes.

Hennequin hat gezeigt, dass diese Eigenschaft genügt, um die angenommene Zufälligkeit auch in den resultierenden Teilproblemen garantieren zu können [Hen89, Seite 318/319]. Diese Eigenschaft erlaubt es uns, die folgende Rekursion aufzustellen, da jedes Teilproblem mit k Elementen dieselben erwarteten Kosten verursacht. F_n setzt sich dabei folgendermaßen zusammen:

$F_n =$ Kosten für den ersten Teilungsschritt + Kosten für das Sortieren der Teillisten.

Da für zwei Elemente des Feldes die Wahrscheinlichkeit als die Pivot-Elemente gewählt zu werden gleich $1/\binom{n}{2}$ beträgt, ergeben sich die folgenden zu erwartenden Kosten:

$$
\begin{aligned}
F_0 &= 0, \\
F_1 &= 1, \\
F_n &= \sum_{1 \le p < q \le n} \mathbb{P}[\text{pivots}(p,q)] * (\text{Teilungskosten} + \text{rekursiver Aufwand}) \\
&= \sum_{1 \le p < q \le n} \frac{1}{\binom{n}{2}} (f_n + F_{p-1} + F_{q-p-1} + F_{n-q}) \\
&= f_n + \frac{1}{\binom{n}{2}} \sum_{1 \le p < q \le n} (F_{p-1} + F_{q-p-1} + F_{n-q}).
\end{aligned}
$$

Die Notation pivots(p, q) beschreibt hier den Fall, dass die Elemente p und q als die Pivot-Elemente gewählt wurden.
Berücksichtigt man die Anwendung von Insertionsort für ein Teilproblem mit $n \le M$ Elementen, ergibt sich:

$$
F_n = \begin{cases} f_n + 1/\binom{n}{2} \sum_{1 \le p < q \le n} (F_{p-1} + F_{q-p-1} + F_{n-q}), & \text{für } n > M, \\ F_n^{IS}, & \text{für } n \le M. \end{cases} \tag{4.1}
$$

Hierbei bezeichnet F_n^{IS} die Kosten für das Anwenden von Insertionsort für kleine Listen und f_n die Kosten für das einmalige Aufteilen der Rekursion.
Durch das Anpassen von f_n an die zu berechnende Größe kann dieselbe Rekursion für alle Größen verwendet werden und braucht somit nur einmal gelöst werden.

Indem wir Symmetrien der Summe in (4.1) ausnützen, ergibt sich

$$\begin{aligned}
\sum_{1\le p<q\le n}(F_{p-1}+F_{q-p-1}+F_{n-q}) &= \sum_{1\le p<q\le n}F_{p-1}+\sum_{1\le p<q\le n}F_{q-p-1}+\sum_{1\le p<q\le n}F_{n-q}\\
&= \sum_{p=1}^{n-1}F_{p-1}\sum_{q=p+1}^{n}1+\sum_{k=0}^{n-2}F_k\sum_{p=1}^{n-1-k}1+\sum_{q=2}^{n}F_{n-1}\sum_{p=1}^{q-1}1\\
&= \sum_{p=1}^{n-1}(n-p)F_{p-1}+\sum_{k=0}^{n-2}(n-1-k)F_k+\sum_{q=2}^{n}(q-1)F_{n-q}\\
&= \sum_{k=0}^{n-2}(n-1-k)F_k+\sum_{k=0}^{n-2}(n-1-k)F_k+\sum_{k=0}^{n-2}(n-1-k)F_k\\
&= 3\sum_{k=0}^{n-2}(n-k-1)F_k.
\end{aligned}$$

Unsere Rekursion (4.1) vereinfacht sich somit zu

$$F_n = f_n + \frac{3}{\binom{n}{2}}\sum_{k=0}^{n-2}(n-k-1)F_k, \quad \text{für } n > M, \tag{4.2}$$

oder alternativ zu

$$F_n = f_n + \frac{6}{n(n-1)}\sum_{k=0}^{n-2}(n-k-1)F_k, \quad \text{für } n > M. \tag{4.3}$$

4.2.1 Lösen der Rekursion

Die Rekursion (4.3) wird nun auf zwei unterschiedliche Arten gelöst werden. Als erstes durch elementare Umformungen und danach durch die Methode der Erzeugenden Funktionen.
Interpretiert man F_n als eine Funktion von f_n, werden wir weiters feststellen, dass F_n in f_n linear ist.

4.2.2 Elementare Lösung

Der elementare Lösungsweg folgt im wesentlichen [WNN15, Appendix A].
Als erstes werden wir den Faktor vor der Summe eliminieren. Wir definieren aus diesem Grund $A_n := \binom{n+1}{2} F_{n+1} - \binom{n}{2} F_n$ und erhalten für $n > M + 2$:

$$\begin{aligned} A_n &= \overbrace{\binom{n+1}{2} f_{n+1} - \binom{n}{2} f_n}^{=:a(n)} \\ &\quad + \frac{(n+1)n}{2} \frac{6}{(n+1)n} \sum_{k=0}^{n+1} (n-k) F_k - \frac{n(n-1)}{2} \frac{6}{n(n-1)} \sum_{k=0}^{n-2} (n-k-1) F_k \\ &= a(n) + 3 \sum_{k=0}^{n-1} F_k. \end{aligned}$$

Durch einfache Subtraktion eliminiert man die Summe in der auftretenden Rekursion und erhält:

$$B_n := A_{n-1} - A_n = a(n+1) - a(n) + 3F_n, \quad \text{für } n \geq M + 2.$$

Durch leichtes Umformen und Division durch den Faktor $\binom{n+2}{2}$ erhält man nach Einsetzen für A_n:

$$\begin{aligned} (B_n - 3F_n) / \binom{n+2}{2} &= (A_{n+1} - A_n - 3F_n) / \binom{n+2}{2} \\ &= \left(\binom{n+2}{2} F_{n+2} - \binom{n+1}{2} F_{n+1} - \left(\binom{n+1}{2} F_{n+1} - \binom{n}{2} F_n \right) - 3F_n \right) \Big/ \binom{n+2}{2} \\ &= F_{n+2} - \frac{2n}{n+2} F_{n+1} + \frac{\frac{1}{2} n(n-1) - 3}{\frac{1}{2}(n+2)(n+1)} F_n \\ &= F_{n+2} - \frac{2n}{n+2} F_{n+1} + \frac{\frac{1}{2}(n-3)(n+2)}{\frac{1}{2}(n+2)(n+1)} F_n \\ &= F_{n+2} - \frac{2n}{n+2} F_{n+1} + \frac{n-3}{n+1} F_n. \end{aligned}$$

Betrachtet man nun $C_n := F_n - \left(\frac{n-4}{n}\right) F_{n-1}$ und berechnet

$$\begin{aligned} C_{n+2} - C_{n+1} &= F_{n+2} - \frac{n-2}{n+2} F_{n+1} - \left(F_{n+1} - \frac{n-3}{n+1} F_n \right) \\ &= F_{n+2} - \frac{2n}{n+2} F_{n+1} + \frac{n-3}{n+1} F_n, \end{aligned}$$

kann man danach die letzten beiden Ergebnisse gleichsetzen und erhält:

$$C_{n+2} - C_{n+1} = (B_n - 3F_n)/\tbinom{n+2}{2} = \underbrace{(a(n+1) - a(n))/\tbinom{n+2}{2}}_{=:c(n)}, \quad \text{für } n \geq M+4.$$

Diese letzte Gleichung kann nun einfach über n aufsummiert werden:

$$C_n = \underbrace{\sum_{i=M+4}^{n} c(i-2) + C_{M+3}}_{=:d(n)}, \qquad \text{für } n \geq M+4.$$

Durch Einsetzen in die Definition von $C_n = F_n - \left(\frac{n-4}{n}\right) F_{n-1}$ erhält man

$$F_n = \frac{n-4}{n} F_{n+1} + d(n), \qquad \text{für } n \geq M+4.$$

Multipliziert man diese Gleichung mit $\binom{n}{4}$ und verwendet $\binom{n}{4} \cdot \left(\frac{n-4}{n}\right) = \binom{n-1}{4}$, erhält man folgende Rekursion für $D_n := \binom{n}{4} F_n$:

$$\begin{aligned}
D_n &= D_{n-1} + \binom{n}{4} d(n) && (4.4)\\
&= \sum_{i=M+4}^{n} \binom{i}{4} d(i) + D_{M+3} && (4.5)\\
&= \sum_{i=1}^{n} \binom{i}{4} d(i) + D_{M+3} - \sum_{i=1}^{M+3} \binom{i}{4} \left(\overbrace{\sum_{j=M+4}^{i} c(j-2)}^{=0} + C_{M+3} \right) && (4.6)\\
&= \sum_{i=1}^{n} \binom{i}{4} d(i) + D_{M+3} - \binom{M+4}{5} C_{M+3}. && (4.7)
\end{aligned}$$

Die letzte Gleichheit folgt aus (2.7).
Einsetzen der Definitionen von $d(n)$, $c(n)$ und $a(n)$ in $\sum_{i=1}^{n}\binom{i}{4}d(i)$ liefert:

$$\begin{aligned}\sum_{i=1}^{n}\binom{i}{4}d(i) &= \sum_{i=1}^{n}\binom{i}{4}\left(C_{M+3}+\sum_{j=M+2}^{i-2}\frac{a(j+1)-a(j)}{\binom{j+2}{2}}\right)\\ &= \binom{n+1}{5}C_{M+3}+\sum_{i=M+4}^{n}\binom{i}{4}\sum_{j=M+2}^{i-2}\left(f_{j+2}-\frac{2j}{j+2}f_{j+1}+\frac{\binom{j}{2}}{\binom{j+2}{2}}f_j\right).\end{aligned} \tag{4.8}$$

Setzt man nun das Ergebnis (4.8) in die Gleichung (4.4) ein und verwendet weiters die Definitionen von C_n und D_n, ergibt sich schlussendlich:

$$\begin{aligned}F_n =& \frac{\binom{n+1}{5}}{\binom{n}{4}}C_{M+3}+\frac{1}{\binom{n}{4}}\sum_{i=M+4}^{n}\binom{i}{4}\sum_{j=M+2}^{i-2}\left(f_{j+2}-\frac{2j}{j+2}f_{j+1}+\frac{\binom{j}{2}}{\binom{j+2}{2}}f_j\right)\\ &+\frac{D_{M+3}-\binom{M+4}{5}C_{M+3}}{\binom{n}{4}}\\ =&\frac{1}{\binom{n}{4}}\sum_{i=M+4}^{n}\binom{i}{4}\sum_{j=M+2}^{i-2}\left(f_{j+2}-\frac{2j}{j+2}f_{j+1}+\frac{\binom{j}{2}}{\binom{j+2}{2}}f_j\right)\\ &+\left(\frac{n+1}{5}+\frac{\binom{M+3}{4}-\binom{M+4}{5}}{\binom{n}{4}}\right)F_{M+3}-\frac{M-1}{M+3}\left(\frac{n+1}{5}-\frac{\binom{M+4}{5}}{\binom{n}{4}}\right)F_{M+2}.\end{aligned} \tag{4.9}$$

An dieser Stelle sieht man, dass F_n in f_n linear ist. Wir nehmen nun an, dass unsere Kostenfunktion f_n linear in n ist und können f_n somit folgendermaßen darstellen:

$$f_n = an+b, \qquad \text{für } a\in\mathbb{N}, b\in\mathbb{Z}, n\geq M+1. \tag{4.10}$$

Durch Einsetzen von (4.10) in (4.9) werden wir eine explizite Lösung für F_n gewinnen können.

Wir starten mit der geschlossenen Darstellung der Rekursion:

$$F_n = \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \binom{i}{4} \sum_{j=M+2}^{i-2} \overbrace{\left(f_{j+2} - \frac{2j}{j+2} f_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} f_j \right)}^{\eta_j} + \underbrace{\left(\frac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}} \right) F_{M+3} - \frac{M-1}{M+3} \left(\frac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}} \right) F_{M+2}}_{\rho} \tag{4.11}$$

Nach Einsetzen von $f_n = an + b$ und Ausführen der Partialbruchzerlegung erhält man

$$\eta_j := \left(f_{j+2} - \frac{2j}{j+2} f_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} f_j \right) = \frac{8a - 2b}{j+2} - \frac{2a - 2b}{j+1}.$$

Mit diesem Ergebnis kann man die innere Summe in (4.11) durch Harmonische Reihen darstellen:

$$\begin{aligned} \sum_{j=M+2}^{i-2} \eta_j &= (8a - 2b)(\mathcal{H}_i - \mathcal{H}_{M+3}) - (2a + 2b)(\mathcal{H}_{i-1} - \mathcal{H}_{M+2}) \\ &= 6a(\mathcal{H}_i - \mathcal{H}_{M+3}) + (2a - 2b) \left(\frac{1}{i} - \frac{1}{M+3} \right). \end{aligned}$$

Durch Verwenden der Identitäten (2.5),(2.7) und $\binom{n}{k} = \binom{n-1}{k-1}\left(\frac{n}{k}\right)$ erhält man:

$$\begin{aligned}
\frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \binom{i}{4} \sum_{j=M+2}^{i-2} \eta_j =& \frac{6a}{\binom{n}{4}} \left(\binom{n+1}{5} \left(\mathcal{H}_{n+1} - \frac{1}{5} \right) - \binom{M+4}{5} \left(\mathcal{H}_{M+4} - \frac{1}{5} \right) \right) \\
&+ \frac{2a-2b}{\binom{n}{4}} \sum_{i=M+3}^{n-1} \left(\frac{1}{4} \binom{i-3}{3} - \frac{1}{M+3} \binom{i}{4} \right) \\
&- \frac{6a}{\binom{n}{4}} \mathcal{H}_{M+3} \left(\binom{n+1}{5} - \binom{M+4}{5} \right) \\
=& \frac{6}{5} a(n+1) \left(\mathcal{H}_{n+1} - \frac{1}{5} \right) - 6a \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\mathcal{H}_{M+4} - \frac{1}{5} \right) \\
&+ \frac{2a-2b}{\binom{n}{4}} \left(\frac{1}{4} \left(\binom{n}{4} - \binom{M+3}{4} \right) - \frac{1}{M+3} \left(\binom{n+1}{5} - \binom{M+4}{5} \right) \right) \\
&- 6a \mathcal{H}_{M+3} \left(\frac{n+1}{5} - \binom{M+4}{5} \Big/ \binom{n}{4} \right) \\
=& \frac{6}{5} a(n+1) \mathcal{H}_{n+1} - \frac{n+1}{5} \left(6a \mathcal{H}_{M+3} + \frac{2a-2b}{M+3} + \frac{6}{5} a \right) + \frac{a-b}{2} \\
&+ \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\frac{6}{5} a - 6a(\mathcal{H}_{M+4} - \mathcal{H}_{M+3}) - \frac{a-b}{2} \frac{5}{M+4} + \frac{2a-2b}{M+3} \right) \\
=& \frac{6}{5} a(n+1) \mathcal{H}_{n+1} - \frac{n+1}{5} \left(6a \mathcal{H}_{M+3} + \frac{2a-2b}{M+3} + \frac{6}{5} a \right) \\
&+ \frac{a-b}{2} + \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\frac{6}{5} a + \frac{2a-2b}{M+3} + \frac{5b-17a}{2(M+4)} \right). \qquad (4.12)
\end{aligned}$$

Als nächstes betrachten wir

$$\rho = \left(\frac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}} \right) F_{M+3} - \frac{M-1}{M+3} \left(\frac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}} \right) F_{M+2}. \qquad (4.13)$$

Wir beginnen, indem wir unsere ursprüngliche Rekursion (4.2) zweimal einsetzen, einmal für F_{M+3} und einmal für F_{M+2}. Weiters verwenden wir $F_n = F_n^{IS}$ für $n \leq M$.

$$\begin{aligned}
F_{M+3} &= f_{M+3} + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M+1} (M+3-k) F_k \\
&= f_{M+3} + \frac{3}{\binom{M+3}{2}} \left(f_{M+1} + \frac{3}{\binom{M+1}{2}} \sum_{k=0}^{M-1} (M-k) F_k^{IS} \right) + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M} (M+2-k) F_k^{IS} \\
&= f_{M+3} + \frac{3}{\binom{M+3}{2}} f_{M+1} + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M} \left((M+2-k) + \frac{3(M-k)}{\binom{M+1}{2}} F_k^{IS} \right).
\end{aligned}$$

Einsetzen für F_{M+2} liefert

$$F_{M+2} = f_{M+2} + \frac{3}{\binom{M+2}{2}} \sum_{k=0}^{M} (M+1-k) F_k^{IS}.$$

Die Ergebnisse für F_{M+3} und F_{M+2} werden nun in die Gleichung (4.13) eingesetzt. Gemeinsam mit $f_n = an + b$, für $n \geq M+1$, erhält man:

$$\begin{aligned}
\rho =& \frac{n+1}{5} \left((M+3)a + b + \frac{3}{\binom{M+3}{2}} ((M+1)a + b) - \frac{M-1}{M+3} ((M+2)a + b) \right) \\
&+ \frac{n+1}{5} \sum_{k=0}^{M} \left(\frac{3(M+2-k)}{\binom{M+3}{2}} + \frac{9(M-k)}{\binom{M+1}{2}\binom{M+3}{2}} - \frac{M-1}{M+3} \frac{3(M+1-k)}{\binom{M+2}{2}} \right) F_k^{IS} \\
&+ \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\left(\frac{5}{M+4} - 1 \right) F_{M+3} + \frac{M-1}{M+3} F_{M+2} \right) \\
=& \frac{n+1}{5} \left(5a + \frac{6(b-a)}{M+2} + \frac{2(4a-b)}{M+3} \right) + \frac{n+1}{5} \sum_{k=0}^{M} \frac{3M-2k}{\binom{M+2}{3}} F_k^{IS} \\
&+ \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\frac{M-1}{M+3} F_{M+2} - \frac{M-1}{M+4} F_{M+3} \right).
\end{aligned} \tag{4.14}$$

Addition der Resultate (4.12) und (4.14) liefert nun folgende explizite Formel für F_n:

$$\begin{aligned}
F_n =& \frac{6}{5} a(n+1) \mathcal{H}_{n+1} + \frac{n+1}{5} \left(\frac{19}{5} a + \frac{6(b-a)}{M+2} - 6a\mathcal{H}_{M+2} \right) + \frac{a-b}{2} + \frac{n+1}{5} \sum_{k=0}^{M} \frac{3M-2k}{\binom{M+2}{3}} F_k^{IS} \\
&+ \frac{\binom{M+4}{5}}{\binom{n}{4}} \left(\frac{6}{5} a + \frac{2(a-b)}{M+3} + \frac{5b-17a}{2(M+4)} - \frac{M-1}{M+4} F_{M+3} + \frac{M-1}{M+3} F_{M+2} \right).
\end{aligned} \tag{4.15}$$

4.2.3 Lösung mittels erzeugender Funktionen

Es wird nun gezeigt, wie die Rekursion (4.3) mit Hilfe von erzeugenden Funktionen gelöst werden kann. Zur Vereinfachung gilt für die gesamte Berechnung $M = 1$. Es wird hier im wesentlichen dem Lösungsweg von [Wil12, Kapitel 4.2.2] gefolgt, welcher wiederum an [Hen91] angelehnt ist. Unterschiede ergeben sich durch die leicht abweichenden Kostenfunktionen f_n.

Wir beginnen wieder bei der Rekursion

$$F_n = f_n + \frac{6}{n(n-1)} \sum_{k=0}^{n-2} (n-k-1) F_k.$$

Multiplikation mit $n(n-1)z^{n-2}$ und Summation über alle $n \geq 2$ liefert

$$\sum_{n\geq 2} n(n-1) F_n z^{n-2} = \sum_{n\geq 2} n(n-1) f_n z^{n-2} + 6 \sum_{n\geq 2} \sum_{k=0}^{n-2} (n-k-1) z^{n-2-k} F_k z^k. \quad (4.16)$$

Wir betrachten nun die Doppelsumme. Als erstes erweitern wir die Summationsgrenzen. Dies können wir problemlos machen, da $F_0 = 0$ gilt. Ersetzen von $n-1$ durch n und Darstellen als Produkt von erzeugenden Funktionen liefert danach

$$6 \sum_{n\geq 1} \sum_{k=0}^{n-1} (n-k-1) z^{n-2-k} F_k z^k = 6 \sum_{n\geq 0} \sum_{k=0}^{n} (n-k) z^{n-1-k} F_k z^k = \left(\sum_{n\geq 0} n z^{n-1} \right) \left(\sum_{n\geq 0} F_n z^n \right).$$

Setzt man nun erneut $F(z) = \sum_{n\geq 0} F_n z^n$, erhält man

$$\left(\sum_{n\geq 0} n z^{n-1} \right) F(z) = \left(\sum_{n\geq 0} z^n \right)' F(z) = \left(\frac{1}{1-z} \right)' F(z) = \frac{F(z)}{1-z^2}.$$

Eingesetzt in die obige Gleichung (4.16) ergibt sich, nachdem auch $f(z) = \sum_{n\geq 0} f_n z^n$ gesetzt wurde, somit

$$F''(z) = f''(z) + \frac{6}{1-z^2} F(z),$$

oder umgeformt

$$(1-z^2) F''(z) - 6F(z) = (1-z^2) f''(z). \quad (4.17)$$

Hierbei handelt sich erneut um eine, wie bereits bei der Analyse der Median-of-Three Verbesserung auftauchende, Euler-Differentialgleichung. Wir werden diese Gleichung erneut durch Einführen eines Operators θ lösen, der sowohl multipliziert als auch differenziert.

Setzt man $\theta F(z) = (1-z)F'(z)$, ergibt sich $\theta^2 F(z) = (1-z)(-F'(z)) + (1-z)^2 F''(z)$ und damit kann die vorherige Gleichung (4.17) geschrieben werden als

$$(\theta^2 + \theta - 6)F(z) = (1-z)^2 f''(z).$$

Faktorisieren von $\theta^2 + \theta - 6$ in $(\theta+3)(\theta-2)$ und $D(z) := (\theta+3)F(z)$ liefert eine Differentialgleichung 1. Ordnung:

$$\begin{aligned}(\theta-2)D(z) &= (1-z^2)f''(z),\\ (1-z)D'(z) - 2D(z) &= (1-z^2)f''(z).\end{aligned}$$

Nach Multiplikation mit $(1-z)$ kann die Gleichung als

$$\big((1-z)^2 D(z)\big)' = (1-z)^3 f''(z)$$

geschrieben werden. Integration über z liefert sofort

$$D(z) = \frac{1}{(1-z)^2}\int_0^z (1-s)^3 f''(s)ds + c.$$

Da $D(0) = F'(0) + 3F(0) = 0$ gilt, erhält man als Lösung für $D(z)$

$$D(z) = \frac{1}{(1-z)^2}\int_0^z (1-s)^3 f''(s)ds.$$

Nun muss noch $D(z) = (1-z)F'(z) + 3F(z)$ gelöst werden. Dies geschieht auf ähnliche Weise wie zuvor, man erweitert mit $\frac{1}{(1-z)^4}$ und erhält:

$$\begin{aligned}(1-z)F'(z) + 3F(z) &= D(z),\\ (1-z)^{-3}F'(z) + 3(1-z)^{-4}F(z) &= (1-z)^{-4}D(z),\\ \big((1-z)^{-3}F(z)\big)' &= (1-z)^{-4}D(z),\\ F(z) &= (1-z)^3\int_0^z (1-t)^{-4}D(t)dt.\end{aligned}$$

Die letzte Gleichheit gilt wieder wegen $F(0) = 0$. Setzt man nun für $D(t)$ ein, erhält man

$$F(z) = (1-z)^3 \int_0^z (1-t)^{-6} \int_0^t (1-s)^3 f''(s) ds dt. \tag{4.18}$$

Wie bei der elementaren Lösung werden wir nun wieder für f_n die spezielle Form $f_n = an + b$ einsetzen und somit eine explizite Lösung für F_n erhalten:

$$\begin{aligned} f(z) &= \sum_{n\geq 0} f_n z^n \\ &= a \sum_{n\geq 0} n z^n + b \sum_{n\geq 0} z^n \\ &= \frac{az}{(1-z)^2} + \frac{b}{1-z}. \end{aligned}$$

Einsetzen in (4.18) ergibt nun:

$$\begin{aligned} F(z) &= (1-z)^3 \int_0^z (1-t)^{-6} \int_0^t (1-s)^3 f''(s) ds dt \\ &= (1-z)^3 \int_0^z (1-t)^{-6} \int_0^t (1-s)^3 \left(\frac{2a(s+2)}{(1-s)^4} + \frac{2b}{(1-s)^3} \right) ds dt \\ &= (1-z)^3 \int_0^z (1-t)^{-6} \left(\int_0^t \frac{2a(s+2)}{1-s} ds + \int_0^t 2b ds \right) dt \\ &= (1-z)^3 \int_0^z (1-t)^{-6} \Big(2a - 2a(t + 3\ln(1-t) - 2b(1-t) + c_1 \Big) dt. \end{aligned}$$

Nach der Integration des zweiten Integrals wurde zusätzlich $2b$ subtrahiert, um nach dem Herausheben den Faktor $(1-t)$ zu erhalten. Diese Subtraktion wird schlussendlich durch die Integrationskonstante c_1 ausgeglichen und beeinflusst unsere Berechnung somit nicht.

Wir fahren fort:

$$\begin{aligned}
F(z) =&(1-z)^3\Big(\int_0^z 2a\frac{1}{(1-t)^6}dt - \int_0^z 2a\frac{t}{(1-t)^6}dt - \int_0^z 2a\frac{3\ln(1-t)}{(1-t)^6}dt \\
&- \int_0^z 2b\frac{1}{(1-t)^5}dt + \int_0^z \frac{c_1}{(1-t)^6}dt\Big) \\
=&(1-z)^3\left(2a\frac{5(1-z)}{20(1-z)^5} - \frac{6}{5}a\frac{\ln(1-z)+\frac{1}{5}}{(1-z)^5} - \frac{1}{2}b\frac{1}{(1-z)^4} + c_1\frac{1}{5(1-z)^5} + c_2\right) \\
=&\frac{6}{5}a\frac{\ln(\frac{1}{1-z})}{(1-z)^2} + \left(\frac{1}{5}c_1 - \frac{6}{25}a\right)\frac{1}{(1-z)^2} + \left(\frac{a-b}{2}\right)\frac{1}{1-z} + (1-z)^3c_2.
\end{aligned}$$

Wir haben somit schließlich eine geschlossene Formel für die erzeugende Funktion $F(z)$ des durchschnittlichen Aufwandes von Dual-Pivot Quicksort bei linearen Partitionierungskosten erhalten.
Man erhält nun die Lösung für die Rekursion (4.3) durch Koeffizientenvergleich von $F_n = [z^n]F(z)$. Die Reihendarstellung des ersten Summanden wird durch das Anwenden der Formel (7.43) von [GKP91,Seite 351] gewonnen:

$$\frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right) = \sum_{n\geq 0}(\mathcal{H}_{m+n} - \mathcal{H}_m)\binom{m+n}{n}z^n.$$

Der letzte Summand kann ignoriert werden, da er nur Grad 3 besitzt. Man erhält also:

$$\begin{aligned}
F_n &= \frac{6}{5}a(\mathcal{H}_{n+1} - \mathcal{H}_1)\binom{n+1}{n} + \left(\frac{1}{5}c_1 - \frac{6}{25}a\right)(n+1) + \left(\frac{a-b}{2}\right) \\
&= \frac{6}{5}a\mathcal{H}_{n+1}(n+1) + \left(\frac{1}{5}c_1 - \frac{6}{25}a - \frac{6}{5}a\right)(n+1) + \left(\frac{a-b}{2}\right) \\
&= \frac{6}{5}a\mathcal{H}_{n+1}(n+1) + \left(\frac{1}{5}c_1 - \frac{36}{25}a\right)(n+1) + \left(\frac{a-b}{2}\right). \qquad (4.19)
\end{aligned}$$

Um die Integrationskonstanten c_1 und c_2 zu bestimmen, setzen wir in die Anfangsbedingungen ein und lösen das daraus resultierende lineare Gleichungssystem

$$\begin{aligned}
0 = C_0 = [z^0]C(0) = C(0) &= \frac{1}{5}c_1 - \frac{6}{25}a + \frac{a-b}{2} + c_2, \\
0 = C_1 = [z^1]C(z) = C'(0) &= \frac{18}{25}a + \frac{2}{5}c_1 + \frac{a-b}{2} - 3c_2.
\end{aligned}$$

Als Lösung erhalten wir:

$$\begin{aligned} c_1 &= 2b - 2a, \\ c_2 &= \frac{7}{50}a + \frac{1}{10}b. \end{aligned}$$

Setzt man die Ergebnisse nun in (4.19) ein, erhält man als Ergebnis für F_n:

$$F_n = \frac{6}{5}a\mathcal{H}_{n+1}(n+1) + \left(-\frac{46}{5}a + 2b\right)\frac{(n+1)}{5} + \left(\frac{a-b}{2}\right). \tag{4.20}$$

Vergleicht man nun die mittels erzeugender Funktionen gewonnene Formel für F_n mit der mittels elementarer Methoden gewonnenen Formel (4.15), sieht man, dass nach Einsetzen von $M = 1$ beide Ergebnisse übereinstimmen:

$$\begin{aligned} F_n =& \frac{6}{5}a(n+1)\mathcal{H}_{n+1} + \frac{n+1}{5}\left(\frac{19}{5}a + \frac{6(b-a)}{3} - 11a\right) + \frac{a-b}{2} + \frac{n+1}{5}\sum_{k=0}^{1}\frac{3-2k}{\binom{3}{3}}C_k^{IS} \\ &+ \frac{\binom{5}{5}}{\binom{n}{4}}\left(\frac{6}{5}a + \frac{2(a-b)}{4} + \frac{5b-17a}{10} - 0C_4 + 0C_3\right) \\ =& \frac{6}{5}a\mathcal{H}_{n+1}(n+1) + \left(-\frac{46}{5}a + 2b\right)\frac{(n+1)}{5} + \left(\frac{a-b}{2}\right). \end{aligned}$$

Im folgenden Abschnitt werden wir nun die erwartete Anzahl an Vergleichen und Vertauschungen während des ersten Partitionierungsschrittes der Algorithmen 7 und 8 berechnen. Die Berechnungen folgen dabei [Wil12], Unterschiede ergeben sich erneut aufgrund der geringfügig abweichenden Kostenfunktion f_n.
Wir gehen von einer zufälligen Permutation von $\{1, \ldots, n\}$ mit $n > M$ aus. Für $n \leq M$ findet keine Partitionierung statt und es gelten die Werte für Insertionsort.

4.3 Average-Case Analyse von Yaroslavskiy's Partitionierung

Der zentrale Part von Yaroslavskiy's Partitionierung sei an dieser Stelle noch einmal angeführt.

```
1:  p := A[left]; q := A[right];
2:  if p>q then
3:      exchange(p, q);
4:  end if
5:  l := left + 1; g := right − 1; k := l;
6:  while k ≤ g do
7:      if A[k] < p then
8:          exchange(A[k], A[l]);
9:          l := l + 1;
10:     else
11:         if A[k] ≥ q then
12:             while A[g] > q and k < g do
13:                 g := g − 1;
14:             end while
15:             exchange(A[k], A[g]);
16:             g := g − 1;
17:             if A[k] < p then
18:                 exchange(A[k], A[l]);
19:                 l := l + 1;
20:             end if
21:         end if
22:     end if
23:     k := k + 1;
24: end while
25: l := l − 1; g := g + 1;
26: A[links] := A[l]; A[l] = p;
27: A[rechts] := A[g]; A[g] = q :
```

Die für unsere Analyse wesentlichen Aktionen finden an den folgenden Stellen statt:

	Vergleich	Tausch
Zeile 2	c_0	
Zeile 3		s_0
Zeile 7	c_1	
Zeile 8		s_1
Zeile 11	c_2	
Zeile 12	c_3	
Zeile 15		s_2
Zeile 17	c_4	
Zeile 18		s_3
Zeile 26		s_4
Zeile 27		s_5

Bevor aber berechnet wird, wie häufig die einzelnen Vergleiche und Vertauschungen durchgeführt werden, muss die Aufmerksamkeit noch auf eine andere Stelle gerichtet werden.

Yaroslavskiy verwendet genauso wie Sedgewick die Crossing-Pointers-Technique für die Partitionierung. Während sich nun die Zeiger k, g aufeinander zubewegen, erreicht einer von ihnen den Punkt, an dem sie sich treffen werden, den Crossing-Point, früher als der andere und muss dort sozusagen „auf den zweiten Zeiger warten“.
Da sich daraus unterschiedliche Werte für die Vergleiche und Vertauschungen ergeben, müssen wir an dieser Stelle zwei Fälle unterscheiden:

- k erreicht den Crossing-Point zuerst:
  g bewegt sich zuletzt, wird in Zeile 16 um 1 verringert, während k in Zeile 23 nochmals um 1 erhöht wird. Es gilt schlussendlich $k = g + 2$.

- g erreicht den Crossing-Point zuerst:
  In diesem Fall verlassen wir die äußere while-Schleife in Zeile 6 immer mit $k = g+1$.

Für Operationen, die für jedes k ausgeführt werden, ergeben sich aus diesem Umstand unterschiedliche Werte. Darum ist das folgende Lemma für eine exakte Analyse unbedingt notwendig.

Lemma 4.1: Crossing-Point Lemma
Sei A eine zufällige Permutation von $\{1, \ldots, n\}$, mit $n \geq 2$. Dann verlässt Algorithmus 7 die äußere while-Schleife in Zeile 6 mit $k = q + \delta = g + 1 + \delta$ mit $\delta \in \{0, 1\}$.
Zusätzlich gilt $\delta = 1$ genau dann, wenn zu Beginn $A[q] > q$ für das größere Pivot-Element $q = \max\{A[1], A[n]\}$ gilt.

Bevor wir aber den Beweis des Crossing-Point Lemmas führen können, benötigen wir ein Hilfslemma:

Lemma 4.2:
Wird im aktuellen Partitionierungsschritt ein Vergleich mit dem Element $A[i]$ durchgeführt, dann wurde $A[i]$ in diesem Partitionierungsschritt noch nicht überschrieben.

Beweis Lemma 4.2: Für den Beweis reicht ein Blick auf den Pseudocode. Bei jeder neuen Iteration der äußeren while-Schleife in Zeile 6 beschreibt k den Index eines Elements, das noch nicht überschrieben wurde. Dies liegt daran, dass alle bisherigen Vertauschungen mit Elementen duchgeführt wurden, deren Indizes kleiner als k oder größer als g sind. Das Gleiche gilt in der inneren while-Schleife in Zeile 12 für g.

□

Nun kann der Beweis des Crossing-Point Lemmas geführt werden.

Beweis Lemma 4.1: Der Teil $k = g + 1 + \delta$ wurde durch die obige Herleitung bereits begründet. Dass auch $k = q + \delta$ gilt, folgt daraus, dass in Zeile 27 q an die Stelle des zuvor um eins erhöhten g getauscht wird. Somit gilt $g + 1 = q$ oder $g = q - 1$.
Für den zweiten Teil zeigen wir beide Implikationen separat. Zuerst nehmen wir $\delta = 1$ an. Der Unterschied zwischen g und k beträgt somit $\delta + 1 = 2$. Dieser Unterschied zwischen k und g kann nur dann erreicht werden, wenn in der letzten Iteration sowohl k inkrementiert als auch g dekrementiert wurde.
Dies kann nur passieren, wenn wir uns während der letzten Iteration innerhalb des else-Teiles in Zeile 11 befinden und die Bedingung $A[k] \geq q$ erfüllt ist. Wegen Lemma 4.2

wissen wir auch, dass $A[k]$ in diesem Partitionierungsschritt noch nicht geändert wurde. Um zu zeigen, dass sogar $A[k] > q$ gilt, führen wir eine Fallunterscheidung für k durch:

- **k<n:** Da wir angenommen haben, dass die eingegebenen Elemente alle verschieden sind, gilt $A[k] \neq A[n] = q$, woraus sofort die Behauptung folgt.
- **k=n:** Nehmen wir an, dass während der letzten Ausführung von Zeile 11 $k = n$ gilt. Der Zeiger g wurde in Zeile 5 mit $g = rechts - 1 = n - 1$ initialisiert und wird nur innerhalb der äußeren while-Schleife dekrementiert, daher $g \leq n - 1$. Die Bedingung dieser while-Schleife lautet $k \leq g$. Setzen wir nun unsere Werte ein, führt uns dies auf einen Widerspruch: $n = k \leq g = n - 1$.

Es gilt somit $A[k] > q$ nachdem das letzte Mal die Zeile 11 ausgeführt wurde.
Da wir $\delta = 1$ angenommen haben, wissen wir aus dem Crossing-Point Lemma, dass nach der while-Schleife $k = q+1$ gilt. Nach dem Vergleich in Zeile 11 wurde k in Zeile 23 noch einmal inkrementiert und es folgt, dass für die ursprüngliche Eingabe $A[q] > q$ gegolten haben muss.

Nehmen wir nun an, dass $A[q] > q$ für die eingegebene Folge gilt. Aus dem Crossing-Point Lemma wissen wir, dass g die äußere while-Schleife mit $g = q - 1$ verlässt. Da g in Zeile 16 dekrementiert wird, muss für g in Zeile 12 $g = q$ gegolten haben. Auf Grund unserer Annahme wissen wir, dass für die Bedingungen der while-Schleife der Teil $k < g$ verletzt sein musste. Daraus folgt, dass $k \geq g = q$ in Zeile 12 gelten muss.
In Zeile 16 wird g nochmals dekrementiert, in Zeile 23 wird k nochmals inkrementiert. Somit gilt nach der äußeren while-Schleife $k \geq q + 2$ und somit $\delta = 1$.

□

Mit Hilfe des Crossing-Point Lemmas kann nun der Erwartungswert von δ berechnet werden. Es wird zuerst die bedingte Erwartung von $\delta = 1$ berechnet werden, um danach über den Satz der totalen Wahrscheinlichkeit die unbedingte Erwartung berechnen zu können.

> **Lemma 4.3:** Sei q das größere Pivot-Element. Die Wahrscheinlichkeit von $\delta = 1$ beträgt $\frac{n-q}{n-2}$, für $n \geq 3$. Daraus folgt weiter, dass $\mathbb{E}_n[\delta|p,q] = \frac{n-q}{n-2}$ gilt.

Beweis Lemma 4.3: Da $\delta \in \{0,1\}$ gilt, erhalten wir $\mathbb{E}_n[\delta|p,q] = \mathbb{P}[\delta = 1|p,q]$. Weiters ergibt sich aus dem Crossing-Point Lemma $\mathbb{P}[\delta = 1|p,q] = \mathbb{P}[A[q] > q|p,q]$. Wir machen

eine Fallunterscheidung:

- $q < n$: $A[q]$ ist somit kein Pivot-Element, da diese an den Stellen $A[1]$ und $A[n]$ liegen. Jedes der $n-2$ Elemente, die keine Pivot-Element sind, kann schlussendlich an die Position $A[q]$ getauscht werden, $n - q$ von ihnen sind größer als q. Die Wahrscheinlichkeit für $A[q] > q$ liegt somit bei $\frac{n-q}{n-2}$.
- $q = n$: In diesem Fall ist q das größte Element der zu sortierenden Liste. Daher kann $A[q] > q$ nie eintreten und es gilt $\mathbb{P}[A[q] > q] = 0 = \frac{n-q}{n-2}$.

□

Für den Beweis des folgenden Lemmas benötigen wir den Erwartungswert des größeren Pivot-Elements q. Da im weiteren Verlauf dieses Kapitels der Erwartungswert von p ebenfalls benötigt wird, berechnen wir an dieser Stelle beide Werte.

Lemma 4.4: Es gilt $\mathbb{E}[p] = \frac{1}{3}(n+1)$ und $\mathbb{E}[q] = \frac{2}{3}(n+1)$.

Beweis Lemma 4.4:

$$\begin{aligned}
\mathbb{E}[p] &= \sum_{1 \le p < q \le n} \frac{1}{\binom{n}{2}} p = \frac{2}{n(n-1)} \sum_{q=2}^{n} 1 * \sum_{p=1}^{q-1} p \\
&= \frac{2}{n(n-1)} \sum_{q=2}^{n} \frac{q^2 - q}{2} = \frac{1}{n(n-1)} \left(\frac{n(n+1)(2n+1)}{6} - 1 - \frac{n(n+1)}{2} + 1 \right) \\
&= \frac{1}{n(n-1)} \left(\frac{n(n+1)(2n+1) - 3n(n+1)}{6} \right) = \frac{1}{(n-1)} \left(\frac{2(n-1)(n+1)}{6} \right) \\
&= \frac{1}{3}(n+1). \\
\mathbb{E}[q] &= \sum_{1 \le p < q \le m} \frac{1}{\binom{n}{2}} q = \frac{2}{n(n-1)} \sum_{q=2}^{n} q \sum_{p=1}^{q-1} 1 = \frac{2}{n(n-1)} \sum_{q=2}^{n} q^2 - q \\
&= \frac{2}{3}(n+1).
\end{aligned}$$

□

Lemma 4.5: Es gilt $\mathbb{E}_n[\delta] = \frac{1}{3}$.

Beweis Lemma 4.5: Aus dem Satz der totalen Wahrscheinlichkeit ergibt sich für den Erwartungswert von δ:

$$\begin{aligned}
\mathbb{E}_n[\delta] &= \sum_{1\le p<q\le n} \mathbb{P}[\text{pivots}(p,q)] * \mathbb{E}[\delta|p,q] \\
&= \sum_{1\le p<q\le n} \frac{1}{\binom{n}{2}} * \frac{n-q}{n-2} = \frac{1}{\binom{n}{2}(n-2)} \left(\sum_{1\le p<q\le n} n - \sum_{1\le p<q\le n} q \right) \\
&\overbrace{=}^{\mathbb{E}[q]} \frac{2}{n(n-1)(n-2)} n\binom{n}{2} - \frac{\frac{2}{3}(n+1)}{n-2} \\
&= \frac{n}{n-2} - \frac{2(n+1)}{3(n-2)} = \frac{3n-2n-2}{3(n-2)} \\
&= \frac{1}{3}.
\end{aligned}$$

□

In den folgenden Kapiteln werden mit S die kleinen, mit M die mittleren und mit L die großen Elemente der Eingabe A bezeichnet. Für die Abkürzungen wurden die englischen Begriffe gewählt, da es ansonsten im späteren Verlauf zu unnötig komplizierter Notation geführt hätte. Es gilt also

$$\begin{aligned}
S &= \{1, \ldots, p-1\}, \text{ mit } |S| = p-1, \\
M &= \{p+1, \ldots, q-1\}, \text{ mit } |M| = q-p-1, \\
L &= \{q+1, \ldots, n\}, \text{ mit } |L| = n-q.
\end{aligned}$$

4.3.1 Durchschnittliche Anzahl an Vergleichen

c_0 in Zeile 2:
Der Vergleich in Zeile 2 wird unabhängig von der zu sortierenden Liste während jedes Partitionierungsschrittes genau einmal ausgeführt. Daher gilt

$$c_0 = 1.$$

c_1 in Zeile 7:
Der Vergleich in Zeile 7 wird für jeden Wert von k einmal durchgeführt. Aus dem Crossing-Point Lemma wissen wir, dass k die while-Schleife mit $k = q + \delta$ verlässt. Da k

am Ende der Schleife nochmals erhöht wird und mit $k = links + 1 = 2$ initialisiert wurde, wird die while-Schleife für alle Werte aus der Positionsmenge $\mathcal{K} = \{2, \ldots, q + \delta - 1\}$ ausgeführt.
Für c_1 ergibt sich somit $c_1 = |\mathcal{K}| = q - 2 + \delta$ und somit wegen Lemma 4.3

$$\begin{aligned}\mathbb{E}_n[c_1|p,q] &= q - 2 + \mathbb{E}_n[\delta|p,q] \\ &= q - 2 + \frac{n-q}{n-2}.\end{aligned}$$

Die unbedingte Wahrscheinlichkeit für c_1 berechnen wir nun wieder über das Gesetz der totalen Wahrscheinlichkeit:

$$\begin{aligned}\mathbb{E}_n[c_1] &= \sum_{1 \le p < q \le n} \mathbb{P}[pivots(p,q)] * \mathbb{E}_n[c_1|p,q] \\ &= \sum_{1 \le p < q \le n} \frac{1}{\binom{n}{2}} \left(q - 2 + \frac{n-q}{n-2} \right) \\ &= \frac{2}{n(n-1)} \sum_{1 \le p < q \le n} q - \frac{2}{n(n-1)} \sum_{1 \le p < q \le n} 2 + \mathbb{E}_n[\delta] \\ &= \frac{2}{3}(n+1) - 2 + \frac{1}{3} \\ &= \frac{2}{3}n - 1.\end{aligned}$$

c_2 in Zeile 11:
Der Vergleich in Zeile 11 wird immer dann ausgeführt, wenn der Vergleich in Zeile 7 fehlschlägt. Aus dieser Tatsache folgt sofort, dass $c_2 \le c_1$ gelten muss.
Aus dem vorhergehenden Abschnitt wissen wir, dass $c_1 = |\mathcal{K}| = |\{2, \ldots, q + \delta - 1\}|$ gilt. Betrachtet man den Pseudocode, stellt man fest, dass die Zeile 11 nur im Fall $A[k] \ge p$ mit $k \in \mathcal{K}$ erreicht wird. Da wir angenommen haben, dass die zu sortierenden Werte alle verschieden sind und da die Indizes der Pivot-Elemtente p und q nicht in $\mathcal{K}$ liegen können, kann der Fall $A[k] \in \{p, q\}$ nicht auftreten und somit muss $c_2 = |\{k \in \mathcal{K} : A[k] \in M \cup L\}|$ gelten.

Dass die Indizes der Pivot-Elemente p und q nicht in $\mathcal{K}$ liegen können, hat den folgenden Grund. Zu Beginn des ersten Partitionierungsschrittes liegen die Pivot-Elemente per Definition an den Stellen $A[1]$ und $A[n]$. Per Definition von $\mathcal{K}$ gilt $1 \notin \mathcal{K}$. Dass $n \notin \mathcal{K}$ gilt, ist leicht zu zeigen. Wir nehmen an, n liegt in $\mathcal{K}$, dann muss per Definition gelten: $n \le q + \delta - 1$. Diese Aussage kann nur im Fall $q = n$ und $\delta = 1$ gelten. Aus dem

Crossing-Point Lemma wissen wir aber, dass $\delta = 1$ nur im Fall $A[q] > q = n$ eintreten kann. Dies ist ein Widerspruch zur Annahme, dass die Eingabe nur Elemente der Menge $\{1, \ldots, n\}$ enthält.

Weiters werden wir in den folgenden Abschnitten die folgende Notation verwenden. Mit $s@\mathcal{K}$ bezeichnen wir die Anzahl der kleinen Elemente an Positionen in $\mathcal{K}$. Formal gilt somit

$$s@\mathcal{K} := |\{i \in \mathcal{K} : A[i] \in S\}|.$$

Die Definitionen für mittlere Elemente m und große Elemente l sowie andere Positionsmengen neben $\mathcal{K}$ seien analog.

Wir können nun c_2 folgendermaßen darstellen.

$$c_2 = m@\mathcal{K} + l@\mathcal{K}.$$

An dieser Stelle ist wichtig zu bemerken, dass hier zwei Zufallsvariablen vorliegen. $\mathcal{K}$ ist eine Zufallsvariable, da die Menge von δ abhängt, $m@\mathcal{K}$ und $l@\mathcal{K}$ sind Zufallsvariablen, da sie einerseits von $\mathcal{K}$ als auch von der Permutation der Eingabefolge abhängen. Da die Zufallsvariablen stochastisch abhängig voneinander sind, können sie nicht einfach durch ihren Erwartungswert ersetzt werden. Durch eine Fallunterscheidung kann dieses Problem allerdings gelöst werden.
Betrachten wir $\delta = 1$: Die Menge $\mathcal{K}$ enthält in diesem Fall die Elemente $\{2, \ldots, q\}$. Aus dem Crossing-Point Lemma wissen wir, dass $A[q] > q$ gilt. Daraus folgt, dass in diesem Fall die Anzahl der großen Elemente mit Index in $\mathcal{K}$, $l@\mathcal{K}$, um eins wächst, für $m@\mathcal{K}$ ändert sich nichts.
Definiert man nun $\mathcal{K}' = \{2, \ldots, q-1\}$, erhält man somit: $l@\mathcal{K} = l@\mathcal{K}' + 1$ und $m@\mathcal{K} = m@\mathcal{K}'$.
Betrachten wir $\delta = 0$: In diesem Fall gilt $\mathcal{K} = \mathcal{K}'$ und es folgt sofort $l@\mathcal{K} = l@\mathcal{K}'$ und $m@\mathcal{K} = m@\mathcal{K}'$.
Mit diesem Wissen kann c_2 nun mit stochastisch unabhängigen Variablen angeschrieben werden:

$$c_2 = m@\mathcal{K}' + l@\mathcal{K}' + \delta.$$

Um nun den Erwartungswert von c_2 zu berechnen, werden die Erwartungswerte von $m@\mathcal{K}'$ und $l@\mathcal{K}'$ benötigt. Die Berechnung wird an dieser Stelle nur für $m@\mathcal{K}'$ durchgeführt, $l@\mathcal{K}'$ kann analog berechnet werden.

Bei gegebenen Pivot-Elementen p und q beschreibt $m@\mathcal{K}'$ die Anzahl der mittleren Elemente mit Positionen in $\mathcal{K}'$. Wenn die Positionen der $|m|$ mittleren Elemente aus den $n-2$ Nicht-Pivot-Elementen gewählt werden, entspricht dies einer Ziehung ohne Zurücklegen. Betrachtet man das Ziehen aus der Menge $\mathcal{K}'$ als Erfolg, erhält man eine hypergeometrische Verteilung mit der Punktwahrscheinlichkeit

$$P[|m| = k] = \frac{\binom{|\mathcal{K}'|}{k}\binom{n-2-|\mathcal{K}'|}{|m|-k}}{\binom{n-2}{|m|}}.$$

Bildet man nun den Erwartungswert $\mathbb{E}[m@\mathcal{K}'|p,q]$, erhält man für den bedingten Erwartungswert

$$\mathbb{E}[m@\mathcal{K}'|p,q] = |m| * \frac{|\mathcal{K}'|}{n-2}$$

und über das Gesetz der totalen Wahrscheinlichkeit schlussendlich

$$\begin{aligned}\mathbb{E}[m@\mathcal{K}'] &= \sum_{1\le p<q\le n} \mathbb{E}[m@\mathcal{K}'|p,q] * \mathbb{P}[pivots(p,q)] \\ &= \frac{2}{n(n-1)} \sum_{1\le p<q\le n} |m| \frac{|\mathcal{K}'|}{n-2}. \end{aligned} \tag{4.21}$$

Nun wissen wir alles Nötige, um c_2 zu berechnen. Für den bedingten Erwartungswert von c_2 bei gegebenen Pivot-Elementen p und q gilt

$$\begin{aligned}\mathbb{E}[c_2|p,q] &= \mathbb{E}[m@\mathcal{K}'|p,q] + \mathbb{E}[l@\mathcal{K}'|p,q] + \mathbb{E}[\delta|p,q] \\ &= ((q-p-1) + (n-q))\,\frac{q-2}{n-2} + \mathbb{E}[\delta|p,q] \\ &= (n-p-1)\,\frac{q-2}{n-2} + \mathbb{E}[\delta|p,q].\end{aligned}$$

Für die unbedingte Erwartung gilt somit

$$\begin{aligned}\mathbb{E}[c_2] &= \mathbb{E}[m@\mathcal{K}'] + \mathbb{E}[l@\mathcal{K}'] + \mathbb{E}[\delta] \\ &= \frac{2}{n(n-1)} \sum_{1\le p<q\le n} (n-p-1)\frac{q-2}{n-2} + \frac{1}{3} \\ &= \left(\frac{2}{n(n-1)} \sum_{1\le p<q\le n} (n-1)q - pq + 2p - 2(n-1)\right) \Big/ (n-2) + \frac{1}{3}. \end{aligned} \tag{4.22}$$

Mit einer Nebenrechnung berechnen wir zwei Teilergebnisse:

$$\begin{aligned}\sum_{1\le p<q\le n} pq &= \sum_{q=2}^{n} q \sum_{p=1}^{q-1} p = \sum_{q=2}^{n} q\left(\frac{(q-1)q}{2}\right) = \frac{1}{2}\sum_{q=2}^{n} q^3 - q^2 \\ &= \frac{1}{2}\left(\frac{n^2(n+1)^2}{4} - \frac{n(n+1)(2n+1)}{6}\right) \\ &= \frac{1}{24}(n-1)n(n+1)(3n+2),\end{aligned}$$

$$2(n-1)\sum_{q=2}^{n}\sum_{p=1}^{q-1} 1 = 2(n-1)\sum_{q=2}^{n} q - 1 = 2(n-1)\frac{n(n-1)}{2},$$

und setzen diese in (4.22) ein:

$$\begin{aligned}\mathbb{E}[c_2] &\overbrace{=}^{\mathbb{E}[p],\mathbb{E}[q]} \frac{(n-1)\frac{2}{3}(n+1) - \frac{1}{12}(n+1)(3n+2) + \frac{2}{3}(n+1) - 2(n-1)}{n-2} + \frac{1}{3} \\ &= \frac{\frac{1}{12}(n-2)(5n-11)}{n-2} + \frac{1}{3} \\ &= \frac{5}{12}n - \frac{7}{12}.\end{aligned}$$

c_3 in Zeile 12:

Der Vergleich in Zeile 12 wird für jeden Wert des Pointers g einmal ausgeführt. Da dieser nur im Inneren der if-Abfrage in Zeile 13 und in Zeile 16 verringert wird, ergibt sich eine sehr einfache Berechnung.

Der Zeiger g wird in Zeile 4 mit $rechts - 1 = n - 1$ initialisiert und aus dem Crossing-Point Lemma wissen wir, dass die äußere while-Schleife mit $g = q - 1$ verlassen wird. Somit kann g nur Werte in

$$G := \{n-1, n-2, \ldots, q\}$$

annehmen. Für gegebenes q ist c_3 somit konstant mit $c_3 = |G| = n - q$. Es gilt also $\mathbb{E}[c_3|p,q] = n - q$. Durch das Gesetz der totalen Wahrscheinlichkeit erhält man den

unbedingten Erwartungswert von c_3:

$$\begin{aligned}\mathbb{E}[c_3] &= \sum_{1\leq p<q\leq n} \mathbb{P}[pivots(p,q)] * \mathbb{E}[c_3|p,q] \\ &= \frac{2}{n(n-1)} \sum_{1\leq p<q\leq n} (n-q) \\ &= n - \frac{2}{3}(n+1) \\ &= \frac{1}{3}n - \frac{2}{3}.\end{aligned}$$

c_4 in Zeile 17:

Der Vergleich in Zeile 17 wird für jeden Vergleich von c_2, welcher als Ergebnis TRUE liefert, einmal durchgeführt. Dies ist genau dann der Fall, wenn $A[k] \geq q$ gilt. Die Anzahl der großen Elemente auf Positionen in $\mathcal{K}$ haben wir mit $l@\mathcal{K}$ bezeichnet. Mit der gleichen Argumentation wie bei c_2 können wir nun auch c_4 darstellen als

$$c_4 = l@\mathcal{K}' + \delta.$$

Somit gilt

$$\begin{aligned}\mathbb{E}[c_4|p,q] &= \mathbb{E}[l@\mathcal{K}'|p,q] + \mathbb{E}[\delta|p,q] \\ &= (n-q)\frac{q-2}{n-2} + \frac{n-q}{n-2}.\end{aligned}$$

Das Gesetz der totalen Wahrscheinlichkeit liefert somit

$$\begin{aligned}
\mathbb{E}[c_4] =& \mathbb{E}[l@\mathcal{K}'] + \mathbb{E}[\delta] \\
=& \frac{2}{n(n-1)} \sum_{1 \le p < q \le n} \left((n-q)\frac{q-2}{n-2} \right) + \frac{1}{3} \\
=& \frac{2}{n(n-1)(n-2)} \sum_{1 \le p < q \le n} (nq - 2n - q^2 + 2q) + \frac{1}{3} \\
=& \frac{2}{n(n-1)(n-2)} \Big(\frac{1}{3}n^2(n^2-1) - 2n\frac{n(n-1)}{2} - \frac{1}{12}(n-1)n(n+1)(3n+2) \\
& + \frac{2}{3}n(n^2-1) \Big) + \frac{1}{3} \\
=& \frac{n-3}{6} + \frac{1}{3} \\
=& \frac{n}{6} - \frac{1}{6}.
\end{aligned}$$

4.3.2 Durchschnittliche Anzahl an Vertauschungen

s_0 in Zeile 3:

Die Vertauschung in Zeile 3 wird so oft ausgeführt, wie der Vergleich c_0 ein wahres Ergebnis liefert. Wir wissen, dass für den Erwartungswert $\mathbb{E}[c_0] = 1$ gilt und da angenommen wurde, dass die Eingabe eine zufällige Permutation von $\{1, \ldots, n\}$ darstellt, gilt für die Wahrscheinlichkeit von $p > q$, $\mathbb{P}[p > q] = \frac{1}{2}$.
Somit erhalten wir für den Erwartungswert für s_0:

$$\begin{aligned}
\mathbb{E}[s_0] &= c_0 * \mathbb{P}[p > q] \\
&= \frac{1}{2}.
\end{aligned}$$

s_1 in Zeile 8:

Die Vertauschung in Zeile 8 wird jedes Mal durchgeführt, wenn der Vergleich $A[k] < p$ in Zeile 7 TRUE als Ergebnis hat. Aus dem Crossing-Point Lemma wissen wir, dass der Index k alle Werte aus der Menge $\mathcal{K} = \{2, \ldots, q + \delta - 1\}$ annimmt. Die if-Abfrage liefert somit für jedes kleine Element mit Index in $\mathcal{K}$ ein wahres Ergebnis, daher gilt $s_1 = s@\mathcal{K}$. Wie bereits bei der Analyse von c_2 finden wir hier eine stochastische Abhängigkeit zwischen $\mathcal{K}$ und $s@\mathcal{K}$, da $\mathcal{K}$ abhängig ist von δ. Aber auch in diesem Fall können wir die Abhängigkeit über eine Fallunterscheidung lösen.
Es sei $\delta = 1$ und $\mathcal{K}' = \{2, \ldots, q-1\}$, wie bei der Analyse der Vergleiche. Aus dem

Crossing-Point Lemma folgt, dass $A[q] > q$ gilt und wir daher keinen Beitrag zu $s@\mathcal{K}$ erhalten. Es gilt somit $s@\mathcal{K} = s@\mathcal{K}'$.
Im Fall $\delta = 0$ gilt laut Definition $\mathcal{K} = \mathcal{K}'$ und wir erhalten somit

$$s_1 = s@\mathcal{K}'.$$

Wir können nun wieder die bedingte Erwartung berechnen und danach über das Gesetz der totalen Wahrscheinlichkeit die unbedingte Erwartung. Dies liefert:

$$\begin{aligned}
\mathbb{E}[s_1|p,q] &= [s@\mathcal{K}'|p,q] \\
&= (p-1)\frac{q-2}{n-q}, \\
\mathbb{E}[s_1] &= \frac{2}{n(n-1)} \sum_{1 \le p < q \le n} (p-1)\frac{q-2}{n-2} \\
&= \frac{1}{4}n - \frac{5}{12}.
\end{aligned}$$

s_2 in Zeile 15:
Nachdem die Vertauschung in Zeile 15 durchgeführt wurde, wird der Vergleich in Zeile 17 genau einmal durchgeführt. Diese beiden Werte sind somit gleich groß und daher gilt

$$\begin{aligned}
\mathbb{E}[s_2] &= \mathbb{E}[c_4] \\
&= \frac{1}{6}n - \frac{1}{6}.
\end{aligned}$$

s_3 in Zeile 18:
Die Vertauschung in Zeile 18 hängt eng mit dem Zeiger l zusammen und kann dadurch leicht berechnet werden. Betrachtet man den Pseudocode von Yaroslavskiy's Partitionierung, stellt man fest, dass der Zeiger l in den Zeilen 9 und 19 jedes Mal unmittelbar nach einer Vertauschung erhöht wird. Wenn nun l alle seine möglichen Positionen in $\mathcal{L}$ durchläuft, erhalten wir $s_1 + s_3 = |\mathcal{L}| - 1$. Da s_1 bereits bekannt ist, können wir s_3 aus $\mathcal{L}$ berechnen.
In Zeile 5 wird l initialisiert mit dem Wert $links + 1 = 2$. Am Ende der while-Schleife wird l um eins verringert, bevor in Zeile 26 das kleinere Pivot-Element p an die Stelle l getauscht wird. Somit muss l am Ende der while-Schleife den Wert $p + 1$ angenommen haben. Also gilt $\mathcal{L} = \{2, \ldots, p+1\}$ und es folgt für s_3:

$$s_3 = |\mathcal{L}| - 1 - s_1 = p - 1 - s@\mathcal{K}'.$$

Für den Erwartungswert von s_3 erhalten wir

$$\begin{aligned}\mathbb{E}[s_3] &= \mathbb{E}[p] - 1 - \mathbb{E}[s_1] \\ &= \frac{1}{3}(n+1) - 1 - \left(\frac{1}{4}n - \frac{5}{12}\right) \\ &= \frac{1}{12}n - \frac{1}{4}.\end{aligned}$$

s_4 in Zeile 26, s_5 in Zeile 27:
Die beiden letzten Vertauschungen in den Zeilen 26 und 27 werden am Ende jedes Partitionierungsschrittes genau einmal ausgeführt. Somit gilt

$$s_4 = s_5 = 1.$$

4.3.3 Ergebnisse

Da wir nun wissen, wie oft die Vergleiche und Vertauschungen an den jeweiligen Stellen ausgeführt werden, können wir die Werte in die gelöste Rekursion von Dual-Pivot-Quicksort einsetzen und erhalten somit die erwartete Anzahl an Vergleichen und Vertauschungen.
Allerdings muss hier das folgende berücksichtigt werden. Bei der Lösung der Rekursion (4.3) haben wir eine lineare Kostenfunktion $f_n = an + b$ für $n \geq 2$ angenommen. Im Fall $n = 2$ folgen bestimmte Stellen im Algorithmus nicht dieser Annahme und haben keine Kosten.
Als erstes werden wir erläutern, wie in diesem speziellen Fall $f_2 = 0 \neq 2a + b$ vorzugehen ist, danach werden wir eine Tabelle mit den genauen Werten, die in die Rekursion eingesetzt werden, angeben.

Betrachten wir den elementaren Lösungsweg der Rekursion wie in Kapital 4.2.2. Der Fall $f_2 = 0$ kommt in unserer Herleitung der Rekursion im allgemeinen Fall mit $M \geq 1$ glücklicherweise nur einmal vor. Nämlich bei der Berechnung von F_{M+3} im Fall $M = 1$. Bei der Berechnung von ρ in (4.14) wird F_{M+3} mit dem Faktor $\left(\frac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}}\right) = \frac{n+1}{5} + O(\frac{1}{n^4})$ multipliziert.
Das bedeutet, dass im Fall $M = 1$ und $f_2 = 0 \neq 2a + b$ noch der Term $\left(\frac{n+1}{5}\frac{3}{\binom{M+3}{2}}(2a + b) + O(\frac{1}{n^4})\right) = \frac{2a+b}{10}(n + 1) + O(\frac{1}{n^4})$ abgezogen werden muss.

Für die Kosten für die einzelnen Vergleiche ergibt sich die folgende Tabelle

Stelle	c_0	c_1	c_2	c_3	c_4
Erwartungswert ($n \geq 3$)	1	$\frac{2}{3}n - 1$	$\frac{5}{12}n - \frac{7}{12}$	$\frac{1}{3}n - \frac{2}{3}$	$\frac{1}{6}n - \frac{1}{6}$
best. Wert für $n = 2$	nein	0	0	0	0

Für die Kosten für die einzelnen Vertauschungen ergibt sich die folgende Tabelle

Stelle	s_0	s_1	s_2	s_3	s_4	s_5
Erwartungswert ($n \geq 3$)	$\frac{1}{2}$	$\frac{1}{4}n - \frac{5}{12}$	$\frac{1}{6}n - \frac{1}{6}$	$\frac{1}{12}n - \frac{1}{4}$	1	1
best. Wert für $n = 2$	nein	0	0	0	nein	nein

Als Ergebnis für die Anzahl der zu erwartenden Vergleiche und Vertauschungen während der Partitionierung erhalten wir nach Einsetzen der jeweils betrachteten Stellen c_i und s_i in die gelöste Rekursion (4.15) somit die folgenden Tabellen. Im Fall $M = 1$ wird mittels $\mathcal{H}_n \approx \ln(n) + \gamma$ eine Näherung angegeben.
Um etwaige Unklarheiten bei der Berechnung zu beseitigen, führen wir die Berechnung für die Stelle c_0 hier näher aus.
Für c_0 gilt: $c_0 = 1$. Koeffizientenvergleich mit der angenommenen linearen Kostenfunktion $f_n = an + b$ liefert für a und b die Werte: $a = 0, b = 1$. Setzt man diese Werte nun, gemeinsam mit $M = 1$ und $F^{IS} = 0$, in die Gleichung (4.15) ein, erhält man: $F_n = \frac{n+1}{5}\left(\frac{6}{3}\right) - \frac{1}{2} = \frac{2}{5}n - \frac{1}{10}$.

Stelle	$M = 1$, exakt für $n \geq 4$	Asymptotische Näherung, Fehler $\mathcal{O}(\ln n)$
c_0	$\frac{2}{5}n - \frac{1}{10}$	$0,4n$
c_1	$\frac{4}{5}(n+1)\mathcal{H}_n - \frac{83}{50}n - \frac{2}{75}$	$0,8n\ln(n) - 1,198n$
c_2	$\frac{1}{2}(n+1)\mathcal{H}_n - \frac{41}{40}n - \frac{1}{40}$	$0,5n\ln(n) - 0,736n$
c_3	$\frac{2}{5}(n+1)\mathcal{H}_n - \frac{22}{25}n + \frac{1}{50}$	$0,4n\ln(n) - 0,649n$
c_4	$\frac{1}{5}(n+1)\mathcal{H}_n - \frac{39}{100} - \frac{7}{300}$	$0,2n\ln(n) - 0,275n$
s_0	$\frac{1}{5}n - \frac{1}{20}$	$0,2n$
s_1	$\frac{3}{10}(n+1)\mathcal{H}_n - \frac{127}{200}n - \frac{1}{600}$	$0,3n\ln(n) - 0,462n$
s_2	$\frac{1}{5}(n+1)\mathcal{H}_n - \frac{39}{100}n - \frac{7}{300}$	$0,2n\ln(n) - 0,275n$
s_3	$\frac{1}{10}(n+1)\mathcal{H}_n - \frac{49}{200}n + \frac{13}{600}$	$0,1n\ln(n) - 0,187n$
s_4	$\frac{2}{5}n - \frac{1}{10}$	$0,4n$
s_5	$\frac{2}{5}n - \frac{1}{10}$	$0,4n$
$C = \sum c_i$	$\frac{19}{10}(n+1)\mathcal{H}_n - \frac{711}{200}n - \frac{31}{200}$	$1,9n\ln(n) - 2,458n$
$S = \sum s_i$	$\frac{3}{5}(n+1)\mathcal{H}_n - \frac{27}{100}n - \frac{19}{75}$	$0,6n\ln(n) + 0,076n$

Stelle	$M \geq 2$, Asymptotisch korrekt mit Fehler $\mathcal{O}(1/n^4)$
c_0	$\frac{6}{5(M+2)}(n+1) - \frac{1}{2}$
c_1	$\frac{4}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{38}{75} - \frac{2}{M+2}\right)(n+1) + \frac{5}{6}$
c_2	$\frac{1}{2}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{60} - \frac{6}{5(M+2)}\right)(n+1) + \frac{1}{2}$
c_3	$\frac{2}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{75} - \frac{6}{5(M+2)}\right)(n+1) + \frac{1}{2}$
c_4	$\frac{1}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{150} - \frac{2}{5(M+2)}\right)(n+1) + \frac{1}{6}$
s_0	$\frac{3}{5(M+2)}(n+1) - \frac{1}{4}$
s_1	$\frac{3}{10}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{100} - \frac{4}{5(M+2)}\right)(n+1) + \frac{1}{3}$
s_2	$\frac{1}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{150} - \frac{2}{5(M+2)}\right)(n+1) + \frac{1}{6}$
s_3	$\frac{1}{10}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{300} - \frac{2}{5(M+2)}\right)(n+1) + \frac{1}{6}$
s_4	$\frac{6}{5(M+2)}(n+1) - \frac{1}{2}$
s_5	$\frac{6}{5(M+2)}(n+1) - \frac{1}{2}$
$C = \sum c_i$	$\frac{19}{10}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{361}{300} - \frac{18}{5(M+2)}\right)(n+1) + \frac{3}{2} + O(\frac{1}{n^4})$
$S = \sum s_i$	$\frac{3}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{19}{50} + \frac{7}{5(M+2)}\right)(n+1) - \frac{7}{12} + O(\frac{1}{n^4})$

4.4 Average-Case Analyse von Sedgewicks Partitionierung

Auch der zentrale Teil von Sedgewicks Partitionierung sei an dieser Stelle für den Leser erneut angeführt.

```
1: if right − left ≥ M then
2:     i := left; i_1 := left;
3:     j := right; j_1 := right;
4:     if A[left] > A[right] then exchange(A[left], A[right]);
5:     p := A[left]; q := A[right];
6:     while true do
7:         i := i + 1
8:         while A[i] ≤ q do
9:             if i ≥ j then break outer while //pointers crossed
10:            if A[i] < p then
11:                A[i_1] := A[i];
12:                i_1 := i_1 + 1 ;
13:                A[i] := A[i_1];
14:            end if
15:            i := i + 1;
16:        end while
17:        j := j − 1;
18:        while A[j] ≥ p do
19:            if A[j] > q then
20:                A[j_1] := A[j];
21:                j = j + 1;
22:                A[j] := A[j + 1];
23:            end if
24:            if i ≥ j then break outer while //pointers crossed
25:            j := j − 1;
26:        end while
27:        A[i_1] := A[j]; A[j_1] := A[i];
28:        i_1 := i_1 + 1; j_1 := j_1 − 1;
29:        A[i] := A[i_1]; A[j] := A[j_1];
30:    end while
31:    A[i_1] := p
32:    A[j_1] := q
33:    DualPivotQuicksortSedgewick(A, left, i_1 − 1);
34:    DualPivotQuicksortSedgewick(A, i_1 + 1, j_1 − 1);
35:    DualPivotQuicksortSedgewick(A, j_1 + 1, right);
36: end if
```

Die für unsere Analyse interessanten Stellen sind in der folgenden Tabelle zusammengefasst:

	Vergleich	Tausch
Zeile 4	c_0	s_0
Zeile 8	c_1	
Zeile 10	c_2	
Zeile 11-13		s_1
Zeile 18	c_3	
Zeile 19	c_4	
Zeile 20-22		s_2
Zeile 27-29		s_3
Zeile 31		s_4
Zeile 32		s_5

Für eine exakte Analyse ist es erneut wesentlich, zu wissen, an welcher Stelle sich die beiden Zeiger i und j kreuzen. Dafür benötigen wir zuerst ein Hilfslemma

Lemma 4.6:
Bis auf die Pivot-Elemente wurden die Elemente, die in den Vergleichen c_0 bis c_4 verwendet wurden, bis zu diesem Zeitpunkt nicht verändert.

Beweis: Für den Beweis genügt ein Blick auf den Pseudocode. Nach jedem Tausch mit einem Element $A[i]$ oder $A[j]$ werden die entsprechenden Zeiger i oder j um eins erhöht oder um eins verringert. Somit zeigen die Pointer immer auf ein noch unberührtes Element.

□

Wir nennen den Punkt, an dem sich i und j kreuzen, erneut den Crossing-Point. Um diesen Crossing-Point mit jenem aus dem letzten Kapitel zu unterscheiden, werden wir ihn nun mit χ bezeichnen.
Eine erste Beobachtung sei an dieser Stelle gleich festgehalten. Die Vergleiche in den Zeilen 9 und 24 garantieren, dass die äußere while-Schleife verlassen wird, sobald $i = j$ gilt. Für χ gilt daher sofort $\chi = i = j$.
Es wird nun der Erwartungswert von χ bestimmt.

Lemma 4.7: Crossing-Point Lemma:
Es sei das Array A eine Permutation von $\{1, \ldots, n\}$, für $n \geq 2$. Weiters seien $p = \min\{A[1], A[n]\}$ und $q = \max\{A[1], A[n]\}$ die beiden Pivot-Elemente.
Dann ist der Crossing-Point χ gleich dem Index des $(p+1)$-ten nicht-mittleren Elements von A.

Um uns zukünftig komplizierte Formulierungen zu ersparen führen wir eine neue Notation ein. Im folgenden sei $\#m(x)$ die Anzahl der nicht-mittleren-Elemente zur Linken von x im Ausgangsarray. Formal gilt

$$\#m(x) := |\{A[y] : 1 \leq y \leq x, A[y] \notin M\}|.$$

Mit dieser Notation kann das Crossing-Point Lemma auch folgendermaßen formuliert werden.

Es gelten dieselben Voraussetzungen wie oben. Für den Crossing-Point χ von i und j gilt $\chi := \min\{x : \#m(x) = p + 1\}$.

Beweis Lemma 4.7: Als erstes wird festgestellt werden, dass $A[\chi]$ kein mittleres Element sein kann.
Angenommen wir verlassen die äußere while-Schleife aufgrund der if-Bedingung in Zeile 9. Dann können wir zwei Fälle unterscheiden. Im ersten Fall hat sich der Wert von j nicht verändert. Es gilt daher $j = n$ und daher auch $A[j] = q \notin M$. Im zweiten Fall hat sich der Wert von j geändert. Daher muss $A[j] < p$ gelten, da wir uns nur in dem Fall, dass die Bedingung in Zeile 18 verletzt wurde, wieder in der i-Schleife befinden. In beiden Fällen ist somit $A[j]$ kein mittleres Element.
Verlassen wir die äußere while-Schleife über die Bedingung in Zeile 24, dann wurde zuvor die i-Schleife mit $A[i] > q$ verlassen. Auch hier ist $A[i]$ offensichtlich kein mittleres Element.

Als nächstes wollen wir zeigen, dass wir die äußere Schleife mit $\#m(i) = i_1 + 1$ verlassen.
Wir betrachten die i-while Schleife. Der Index i durchläuft alle Elemente der Menge $\{2, \ldots, \chi\}$. Für jedes $i < \chi$ ist $A[i]$ trivialerweise ein kleines, mittleres oder großes Element. Ist $A[i]$ ein kleines Element, wird es in Zeile 10 hinter i_1 getauscht. Ist $A[i]$ ein

großes Element, wird es, nachdem in der j-while Schleife ein entsprechendes, kleines Element gefunden wurde, in Zeile 27 mit diesem getauscht. Somit befinden sich zwischen den Zeigern i und i_1 nur mittlere Elemente.
Für $i = \chi$ wissen wir von oben bereits, dass $A[\chi]$ kein mittleres Element sein kann. Wenn $A[\chi] \in S$ gilt, verlassen wir schlussendlich die äußere Schleife durch den Vergleich in Zeile 9. Wenn $A[\chi] \in G$ gilt, überspringen wir die erste innere while-Schleife und beenden schlussendlich durch den Vergleich in Zeile 24 die äußere Schleife. In beiden Fällen werden die relevanten Vertauschungen nicht durchgeführt.
Somit erhalten wir, dass i_1 genau die Anzahl an nicht-mittleren Elementen links von χ darstellt, allerdings ohne das Element an der Stelle χ. Es gilt somit

$$\#m(\chi) = \#m(i) = i_1 + 1 = p + 1.$$

Da die Anzahl der mittleren Elemente links von χ an der Stelle χ nochmals erhöht wird, ist χ garantiert der kleinste Index mit $\#m(\chi) = p + 1$.

□

Wir können nun den bedingten und unbedingten Erwartungswert von χ berechnen.

Lemma 4.8:
Für den bedingten Erwartungswert von χ gilt:

$$\mathbb{E}[\chi|p, q] = p + 1 + (q - p - 1)\frac{p}{p + n - q}.$$

Für den unbedingten Erwartungswert von χ gilt:

$$\mathbb{E}[\chi] = \frac{1}{2}n - \frac{3}{2} - \frac{1}{n}.$$

Beweis Lemma 4.8: Es sei μ die Anzahl der mittleren Elemente zur Linken des $(p+1)$-ten nicht-mittleren Elements. Offensichtlich gilt $0 \leq \mu \leq |M| = q - p + 1$.
Aus dem letzten Lemma wissen wir, dass $\chi = p+1+\mu$ gelten muss und aus der Linearität der Erwartung folgt somit $\mathbb{E}[\chi|p, q] = p+1+\mathbb{E}[\mu|p, q]$. Wir müssen also die erwartete Anzahl an mittleren Elementen in Abhängigkeit von p und q bestimmen.
Nehmen wir an, wir entnehmen aus der eingegebenen Folge alle mittleren Elemente sowie die Pivot-Elemente am Beginn und am Ende. Übrig bleiben $p-1+n-q$ kleine oder große Elemente. Zwischen jedem Paar von solchen Elementen, aber auch am Anfang und

am Ende der Folge, können wir nun die $|M|$ mittleren Elemente in zufälliger Reihenfolge und beliebiger Anzahl wieder einfügen. Es existieren somit $p-1+n-q+1 = p+n-q$ Stellen, an denen jedes mittlere Element eingefügt werden kann.

Da nur jene Elemente zu μ zählen, die links des $(p+1)$-ten nicht-mittleren Elements liegen, gibt es für uns nur p „gute“ Slots an denen wir Elemente einfügen können.
Zusammengefasst ergibt dies eine Binomialverteilung: Wir wählen $|M|$ Stellen mit Zurücklegen aus $p+n-q$ Möglichkeiten, wobei p dieser Möglichkeiten einen Erfolg darstellen.
Somit ist $\mathbb{E}[\mu|p,q]$ binomialverteilt und es gilt für den Erwartungswert:

$$\mathbb{E}[\mu|p,q] = |M|\frac{p}{p+n-1} = (q-p-1)\frac{p}{p+n-q}.$$

Aus dem Gesetz der totalen Wahrscheinlichkeit und der Linearität des Erwartungswertes folgt:

$$\begin{aligned}\mathbb{E}[\chi] &= \mathbb{E}[p] + \mathbb{E}[1] + \mathbb{E}[\mu] \\ &= \frac{1}{3}(n+1) + 1 + \mathbb{E}[\mu].\end{aligned}$$

Wir können nun den unbedingten Erwartungswert von μ berechnen. Es gilt

$$
\begin{aligned}
\mathbb{E}[\mu] &= \sum_{1\le p<q\le n} \mathbb{P}[\text{pivots}(p,q)] * \mathbb{E}[\mu|p,q] \\
&= \frac{2}{n(n-1)} \sum_{1\le p<q\le n} (q-p-1)\frac{p}{p+n-q} \\
&= \frac{2}{n(n-1)} \sum_{p=1}^{n-1} p \sum_{m=0}^{n-p-1} \frac{m}{(n-1)-m} \\
&= \frac{2}{n(n-1)} \sum_{p=1}^{n-1} p \sum_{m=0}^{n-p-1} \frac{m-(n-1)+(n-1)}{(n-1)-m} \\
&= \frac{2}{n(n-1)} \sum_{p=1}^{n-1} p \sum_{m=0}^{n-p-1} \left(-1+\frac{n-1}{(n-1)-m}\right) \\
&= -\frac{2}{n(n-1)} \sum_{p=1}^{n-1} p(n-p) + \frac{2}{n} \sum_{p=1}^{n-1} p \sum_{m=0}^{n-p-1} \frac{1}{(n-1)-m} \\
&= -\frac{1}{3}(n+1) + \frac{2}{n} \sum_{p=1}^{n-1} p \sum_{m=p}^{n-1} \frac{1}{m} \\
&= -\frac{1}{3}(n+1) + \frac{2}{n} \sum_{p=1}^{n-1} p(\mathcal{H}_{n-1} - \mathcal{H}_{p-1}) \\
&= \frac{1}{3}(n+1) + \mathcal{H}_{n-1}(n-1) - \frac{2}{n} \sum_{p=1}^{n-1} (p-1)\mathcal{H}_{p-1} - \frac{2}{n} \sum_{p=1}^{n-1} \mathcal{H}_{p-1} \\
&= \frac{1}{3}(n+1) + \mathcal{H}_{n-1}(n-1) - \frac{(n-2)(n-1)(2\mathcal{H}_{n-1}-1)}{2n} - \frac{2}{n}(n-1)(\mathcal{H}_{n-1}-1) \\
&= \frac{1}{3}(n+1) + \mathcal{H}_{n-1} \underbrace{\left(n-1-\frac{(n-1)(n-2)}{n} - \frac{2(n-1)}{n}\right)}_{=0} + \frac{1}{2n} \underbrace{((n-1)(n-2)+4(n-1))}_{n(n+1)} \\
&= \frac{1}{6}(n+1) - \frac{1}{n}.
\end{aligned}
$$

Für den Erwartungswert $\mathbb{E}[\chi]$ folgt somit:

$$\begin{aligned}\mathbb{E}[\chi] &= \mathbb{E}[p] + \mathbb{E}[1] + \mathbb{E}[\mu] \\ &= \frac{1}{3}(n+1) + 1 + \frac{1}{6}(n+1) - \frac{1}{n} \\ &= \frac{1}{2}n + \frac{3}{2} - \frac{1}{n}.\end{aligned}$$

Da die theoretische Vorarbeit nun geleistet wurde, können wir in den folgenden Abschnitten bestimmen, wie oft die jeweiligen Vergleiche und Vertauschungen durchgeführt werden.

4.4.1 Durchschnittliche Anzahl an Vergleichen

c_0 in Zeile 4

Der Vergleich in Zeile 4 wird in jedem Partitionierungsschritt durchgeführt. Es gilt daher

$$c_0 = 1.$$

c_1 in Zeile 8

Der Vergleich in Zeile 8 wird für jeden Wert von i genau einmal durchgeführt. Nach der Initialisierung mit $i = 1$ wird i in Zeile 7 um eins erhöht und durchläuft somit sämtliche Werte der Menge

$$I := \{2, \ldots, \chi\}.$$

Es gilt daher $c_1 = |I| = \chi - 1$ und somit gilt für den bedingten und unbedingten Erwartungswert von c_1:

$$\begin{aligned}
\mathbb{E}[c_1|p,q] &= \mathbb{E}[\chi|p,q] - 1 \\
&= p + 1 + (q - p - 1)\frac{p}{p + n - q} - 1 \\
&= p + \frac{p(q - p - q)}{p + n - q}, \\
\mathbb{E}[c_1] &= \mathbb{E}[\chi] - 1 \\
&= \frac{1}{2}n + \frac{3}{2} - \frac{1}{n} - 1 \\
&= \frac{1}{2}n + \frac{1}{2} - \frac{1}{n}.
\end{aligned}$$

c_2 in Zeile 10

Der Vergleich in Zeile 10 befindet sich innerhalb der ersten inneren while-Schleife und wird deswegen nur im Fall $A[i] \leq q$ erreicht. Auf Grund der if-Abfrage in Zeile 9 wird der Vergleich im Fall $i = \chi$ nicht mehr ausgeführt. Der Pointer i läuft somit in der Menge

$$I' := \{2, \ldots, \chi - 1\}.$$

Für c_2 folgt daher, dass $c_2 = s@I' + m@I'$ gilt. Bei der Berechnung ist zu beachten, dass I', $s@I'$ und $m@I'$ Zufallsvariablen sind und auch voneinander abhängen.
Wir beginnen mit $s@I'$. In der zu sortierenden Folge befinden sich ohne den Pivot-Elementen $p - 1 + n - q$ kleine und große Elemente. Aus dem Crossing-Point Lemma für Sedgewicks Partitionierung wissen wir, dass das Element an der Stelle χ das $(p+1)$-te nicht-mittlere Element ist. Aus dem Beweis von diesem Lemma wissen wir weiter, dass auch die Elemente an den Stellen 1 und χ keine mittleren Elemente sind. Somit befinden sich genau $p-1$ Elemente an Positionen in I' und es gilt aufgrund der gleichen Berechnung wie bei (4.21):

$$\mathbb{E}[s@I'|p,q] = (p-1)\frac{p-1}{p-1+n-q}.$$

Betrachten wir nun $m@I'$. Auch hier liefert das Crossing-Point Lemma die Lösung. Wenn das Element an der Stelle χ das $(p+1)$-te nicht-mittlere Element darstellt, dann müssen sich links von χ genau $\chi - (p + 1)$ mittlere Elemente befinden. Diese Elemente können wieder nicht an den Stellen 1 und χ liegen und müssen deswegen alle an Positionen in

I' sein. Es gilt daher $m@I' = \chi - p - 1$ und wegen der Linearität des Erwartungswertes folgt

$$\begin{aligned}\mathbb{E}[m@I'|p,q] &= \mathbb{E}[\chi|p,q] - p - 1 \\ &= p + 1 + (q - p - 1)\frac{p}{p+n-q} - p - 1 \\ &= (q - p - 1)\frac{p}{p+n-q}.\end{aligned}$$

Mit diesen beiden Zwischenergebnissen können wir nun den Erwartungswert von c_2 berechnen. Für den bedingten Erwartungswert ergibt sich

$$\begin{aligned}E[c_2|p,q] &= \mathbb{E}[s@I'|p,q] + \mathbb{E}[m@I'|p,q] \\ &= \frac{(p-1)^2}{n+p-q-q} + \frac{p(q-p-1)}{n+p-q}.\end{aligned}$$

Mit Hilfe des Gesetzes der totalen Wahrscheinlichkeit folgt weiter

$$\begin{aligned}\mathbb{E}[c_2] &= \mathbb{E}[s@I'] + \mathbb{E}[m@I'] \\ &= \sum_{1\le p<q\le n} \mathbb{P}[\text{pivots}(p,q)]\left(\mathbb{E}[s@I'|p,q] + \mathbb{E}[s@I'|p,q]\right) \\ &= \frac{2}{n(n-1)} \sum_{1\le p<q\le n} \left(\frac{(p-1)^2}{n+p-q-1} + \frac{p(q-p-1)}{n+p-q}\right).\end{aligned}$$

Der zweite Summand wurde von uns bereits berechnet. Dies ist genau $\mathbb{E}[\mu] = \frac{1}{6}(n+1)+\frac{1}{6}$. Betrachten wir nun den ersten Summanden. Wir erhalten:

$$\begin{aligned}\frac{2}{n(n-1)} \sum_{1\le p<q\le n} \frac{(p-1)^2}{n+p-q-1} &= \frac{2}{n(n-1)} \sum_{p=1}^{n-1} (p-1)^2 \sum_{q=p+1}^{n} \frac{1}{n+p-q-1} \\ &= \frac{2}{n(n-1)} \sum_{p=1}^{n-2} p^2(\mathcal{H}_{n-2} - \mathcal{H}_{p-1}) \\ &= \frac{2}{n(n-1)} \mathcal{H}_{n-2} \sum_{p=1}^{n-2} p^2 - \frac{2}{n(n-1)} \sum_{p=1}^{n-2} p^2 \mathcal{H}_{p-1} \\ &= \frac{2(n-2)(n-1)(2n-3)}{n(n-1)6} \mathcal{H}_{n-2} - \frac{2}{n(n-1)} \sum_{p=0}^{n-3} (p+1)^2 \mathcal{H}_p.\end{aligned}$$

Wir betrachten die zweite Summe. Wegen $(p+1)^2 = 2\binom{p}{2} + 3p + 1$ können wir weiter vereinfachen:

$$\begin{aligned}\frac{2}{n(n-1)}\sum_{p=0}^{n-3}(p+1)^2\mathcal{H}_p &= \frac{2}{n(n-1)}\sum_{p=0}^{n-3}\left(2\tbinom{p}{2} + 3p + 1\right)\mathcal{H}_p \\ &= \frac{2}{n(n-1)}\left[2\tbinom{n-2}{3}(\mathcal{H}_{n-2} - \frac{1}{3}) + 2\tbinom{n-2}{2}(\mathcal{H}_{n-2} - \frac{1}{2}) + (n-2)(\mathcal{H}_{n-2} - 1)\right] \\ &= \frac{2}{n(n-1)}\left[\left(\frac{(n-1)(2n-3)}{6}\right)\mathcal{H}_{n-2} - \frac{4n^2 - n + 3}{36}\right].\end{aligned}$$

Der zweite Umformungsschritt folgt aus (2.5).
Einsetzen des Ergebnisses liefert

$$\begin{aligned}\frac{2}{n(n-1)}\sum_{1\le p<q\le n}\frac{(p-1)^2}{n+p-q-1} &= \frac{4n^3 - 9n^2 + 5n - 6}{18(n-1)n} \\ &= \frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}.\end{aligned}$$

Setzt man schlussendlich beide Teilergebnisse zusammen, erhält man für den Erwartungswert von c_2:

$$\begin{aligned}\mathbb{E}[c_2] &= \left(\frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}\right) + \left(\frac{1}{6}(n+1) - \frac{1}{n}\right) \\ &= \frac{7}{18}n - \frac{1}{9} - \frac{1}{n} - \frac{1}{3n(n-1)}.\end{aligned}$$

c_3 in Zeile 18

Der Vergleich in Zeile 18 wird für jeden Wert von j genau einmal durchgeführt. Nach der Initialisierung mit $j = n$ wird j in Zeile 17 um eins verringert und durchläuft somit alle Werte der Menge

$$J := \{n-1, \ldots, \chi\}.$$

Daher gilt $c_3 = |J] = n - \chi$ und somit gilt für den bedingten und unbedingten Erwartungswert von c_3 :

$$\begin{aligned}
\mathbb{E}[c_3|p,q] &= n - \mathbb{E}[\chi|p,q] \\
&= n - 1 - p - \frac{p(q-p-1)}{p+n-q}, \\
\mathbb{E}[c_3] &= n - \mathbb{E}[\chi] \\
&= n - \frac{1}{2}n - \frac{3}{2} + \frac{1}{n} \\
&= \frac{1}{2}n - \frac{3}{2} + \frac{1}{n}.
\end{aligned}$$

c_4 in Zeile 19

Der Vergleich in Zeile 19 befindet sich innerhalb der zweiten, inneren while-Schleife. Aus der Bedingung der while-Schleife ergibt sich, dass der Vergleich für jedes j aus J mit $A[j] \geq p$ durchgeführt wird. Somit erhalten wir $c_4 = m@J + l@J$.
Wie bei der Berechnung von c_2 sind sowohl J als auch die Mengen $m@J$ und $g@J$ Zufallsvariablen und voneinander abhängig.
Wir wissen aus der Berechnung von $\mathbb{E}[\chi|p,q]$ und $\mathbb{E}[\chi]$, dass die kleinen und großen Elemente zusammen $(p-1)+(n-q)$ Elemente ausmachen, von denen $p-1$ in $I' = \{2,\ldots,\chi-1\}$ liegen. Da die Menge $J = \{2,\ldots,n-1\} \setminus I'$, abgesehen von den Pivot-Elementen, genau das Komplement von I' darstellt ergibt sich, dass I' genau $n-q$ kleine und große Elemente enthält. Es folgt daher für den zweiten Summanden $l@J$:

$$\mathbb{E}[l@J|p,q] = (n-q)\frac{n-q}{(p-1)+(n-q)}.$$

Wir betrachten den ersten Summanden. Aus der Berechnung von c_2 wissen wir, dass $\mathbb{E}[m@I'|p,q] = \frac{(q-p-1)p}{p+n-q}$ gilt. Da in Summe $q-p-1$ mittlere Elemente existieren, folgt aus $J = \{2,\ldots,n-1\} \setminus I'$:

$$\mathbb{E}[m@J|p,q] = (q-p-1) - \frac{(q-p-1)p}{p+n-q}.$$

Zusammen erhalten wir den bedingten Erwartungswert für c_4:

$$\mathbb{E}[c_4|p,q] = \frac{(n-q)^2}{n+p-q-1} + (q-p-1) - \frac{(q-p-1)p}{n+p-q}.$$

Indem wir die Symmetrie von $\sum_{1\leq p<q\leq n} \frac{(p-1)^2}{(p-1)+(n-q)} = \sum_{1\leq p<q\leq n} \frac{(n-q)^2}{(p-1)+(n-q)}$ ausnützen, müssen wir nur noch bereits bestimmte Terme einsetzen und erhalten für den unbedingten Erwartungswert von c_4:

$$\begin{aligned}\mathbb{E}[c_4] &= \frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)} + \frac{1}{3}(n-2) - \frac{(q-p-1)p}{n+p-q} \\ &= \frac{7}{18}n - \frac{10}{9} + \frac{1}{n} - \frac{1}{3n(n-1)}.\end{aligned}$$

4.4.2 Durchschnittliche Anzahl an Vertauschungen

s_0 in Zeile 4

Die Vertauschung in Zeile 4 wird immer dann durchgeführt, wenn sich die Pivot-Elemente in der falschen Reihenfolge befinden. Da als Eingabe eine zufällige Permutation angenommen wurde, gilt

$$\mathbb{E}[s_0] = \frac{1}{2}.$$

s_1 in den Zeilen 11 bis 13:

Die Vertauschung in Zeile 11-13 befindet sich in der ersten der inneren while-Schleife und wird ausgeführt, wenn $A[i] < p$ gilt. Wenn wir mit I' wieder die gleiche Menge wie im letzten Abschnitt bezeichnen, gilt somit $s_1 = s@I'$. Die Erwartungswerte wurden bereits bei der Berechnung von c_2 bestimmt und es gilt

$$\begin{aligned}\mathbb{E}[s_1|p,q] &= \frac{(p-1)^2}{(p-1)+(n-q)}, \\ \mathbb{E}[s_1] &= \frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}.\end{aligned}$$

s_2 in den Zeilen 20-22:

Die Vertauschung in Zeile 20-22 wird analog zu s_1 für jedes j mit $A[j] > q$ ausgeführt. Somit gilt $s_2 = l@J$. Der bedingte und der unbedingte Erwartungswert wurden bereits bei der Berechnung von c_4 bestimmt und es folgt

$$\begin{aligned}\mathbb{E}[s_2|p,q] &= \frac{(n-q)^2}{(p-1)+(n-q)}, \\ \mathbb{E}[s_2] &= \frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}.\end{aligned}$$

s_3 in den Zeilen 27-29:

Die Vertauschung in Zeile 27-29 wird für jeden äußeren Schleifendurchlauf einmal durch-

geführt. Um diese Anzahl zu bestimmen, betrachten wir den Zeiger i_1. Dieser Zeiger wurde mit $links = 1$ initialisiert, wird während der Vertauschungen s_1 und s_3 um eins erhöht und stoppt an der Stelle des kleineren Pivot-Elements p. Somit wird i_1 genau $(p-1)$-mal erhöht und es gilt $s_1 + s_3 = p - 1$. Der Erwartungswert von s_1 ist bereits bekannt und es gilt somit:

$$\begin{aligned}
\mathbb{E}[s_3|p,q] &= p - 1 - \mathbb{E}[s_1|p,q] \\
&= p - 1 - \frac{(p-1)^2}{(p-1)+(n-q)}, \\
\mathbb{E}[s_3] &= \mathbb{E}[p] - 1 - \mathbb{E}[s_1] \\
&= \frac{1}{3}(n-2) - \left(\frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}\right) \\
&= \frac{1}{9}n - \frac{7}{18} + \frac{1}{3n(n-1)}.
\end{aligned}$$

s_4 und s_5 in Zeile 31 und 32:
Die Vertauschungen in den Zeilen 31 und 32 erscheinen auf den ersten Blick zwar nicht wie ein normaler Tausch, sondern sehen eher wie eine einfache Zuweisung aus. Doch betrachtet man zusätzlich noch die Zeile 4, dann erkennt man, dass ein Tausch vollzogen wird.
Die Vertauschungen s_4 und s_5 werden in jedem Partitionierungsschritt genau einmal durchgeführt und daher ergibt sich

$$\begin{aligned}
\mathbb{E}[s_4] &= 1, \\
\mathbb{E}[s_5] &= 1.
\end{aligned}$$

4.4.3 Ergebnisse

Wie bereits bei der Analyse von Yaroslavskiy's Partitionierung erhalten wir nun die Anzahl an Vergleichen und Vertauschungen indem wir die berechneten Werte in die gelöste Rekursion (4.15) einsetzen.
Erneut sind spezielle Werte im Fall $M = 1$ und $f_2 = 0 \neq 2a + b$ zu berücksichtigen. Weiters muss der Umstand berücksichtigt werden, dass bei der Lösung der Rekursion Partitionierungskosten der Form $f_n = an + b$ angenommen wurden, Sedgewicks Partitionierung aber an einigen Stellen zusätzliche, nicht-lineare Kosten der Form $f_n = an + b + c/n$ oder $f_n = an + b + c/((n(n-1))$ aufweist.
Auf Grund der Linearität der Gleichung (4.11) können wir den linearen Beitrag $f_n =$

$an + b$ wie gewohnt berechnen und müssen nur noch den einen Beitrag für c/n beziehungsweise $c/((n(n-1))$ addieren. Dieser nichtlineare Beitrag lässt sich auf die gleiche Weise bestimmen und man erhält als Ergebnis für $M = 1$, dass im Fall c/n noch der Term $\frac{c}{20}(n+1)$ addiert werden muss, im Fall $c/((n(n-1))$ ergibt sich der zusätzliche Term $\frac{c}{60}(n+1)$. Wenn $M \neq 1$ gilt, muss im Fall c/n noch der Term $c\left(\frac{n+1}{5}\right)\left(\frac{3(M^2+4M+7)}{(M+3)(M+2)(M+1)^2}\right)$ addiert werden, im Fall $c/(n(n-1)$ ergibt sich der zusätzliche Term $c\left(\frac{n+1}{5}\right)\left(\frac{2(M^2+4M+9)}{(M+3)(M+2)(M+1)^2M}\right)$.

Für die Kosten der einzelnen Vergleiche ergibt sich die folgende Tabelle.

Stelle	c_0	c_1	c_2
Erwartungswert ($n \geq 3$)	1	$\frac{1}{2}n + \frac{1}{2} - \frac{1}{n}$	$\frac{7}{18}n - \frac{1}{9} - \frac{1}{n} - \frac{1}{3n(n-1)}$
best. Wert für $n = 2$	nein	0	0

Stelle	c_3	c_4
Erwartungswert ($n \geq 3$)	$\frac{1}{2}n - \frac{3}{2} + \frac{1}{n}$	$\frac{7}{18} - \frac{10}{9} + \frac{1}{n} - \frac{1}{3n(n-1)}$
best. Wert für n=2	0	0

Für die Kosten der einzelnen Vertauschungen ergibt sich die folgende Tabelle.

Stelle	s_0	s_1	s_2
Erwartungswert ($n \geq 3$)	$\frac{1}{2}$	$\frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}$	$\frac{2}{9}n - \frac{5}{18} - \frac{1}{3n(n-1)}$
best. Wert für $n = 2$	nein	0	0

Stelle	s_3	s_4	s_5
Erwartungswert ($n \geq 3$)	$\frac{1}{9}n - \frac{7}{18} + \frac{1}{3n(n-1)}$	1	1
best. Wert für $n = 2$	0	nein	nein

Als Ergebnis für die Anzahl der zu erwartenden Vergleiche und Vertauschungen während der Partitionierung erhalten wir nach Einsetzen in die gelöste Rekursion somit die folgenden exakten und asymptotischen Formeln.

Stelle	M=1, exakt für $n \geq 4$	Asymptotische Näherung, Fehler $\mathcal{O}(\ln n)$
c_0	$\frac{2}{5}n - \frac{1}{10}$	$0,4n$
c_1	$\frac{3}{5}(n+1)\mathcal{H}_n - \frac{23}{25}n - \frac{8}{25}$	$0,6n\ln(n) - 0,574n$
c_2	$\frac{7}{15}(n+1)\mathcal{H}_n - \frac{397}{450}n - \frac{149}{900}$	$0,467n\ln(n) - 0,613n$
c_3	$\frac{3}{5}(n+1)\mathcal{H}_n - \frac{71}{50}n + \frac{9}{50}$	$0,6n\ln(n) - 1,074n$
c_4	$\frac{7}{15}(n+1)\mathcal{H}_n - \frac{487}{450}n + \frac{121}{900}$	$0,467n\ln(n) - 0,813n$
s_0	$\frac{1}{5}n - \frac{1}{20}$	$0,2n$
s_1	$\frac{4}{15}(n+1)\mathcal{H}_n - \frac{122}{225} - \frac{23}{900}$	$0,267n\ln(n) - 0,388n$
s_2	$\frac{4}{15}(n+1)\mathcal{H}_n - \frac{122}{225} - \frac{23}{900}$	$0,267n\ln(n) - 0,388n$
s_3	$\frac{2}{15}(n+1)\mathcal{H}_n - \frac{76}{225}n + \frac{41}{900}$	$0,133n\ln(n) - 0,261n$
$s_4 = s_5$	$\frac{2}{5}n - \frac{1}{10}$	$0,4n$
$C = \sum_i c_i$	$\frac{32}{15}\mathcal{H}_n(n+1) - \frac{1757}{450}n - \frac{61}{225}$	$2,133n\ln(n) - 2,674n$
$S = \sum_i s_i$	$\frac{4}{5}\mathcal{H}_n(n+1) - \frac{19}{25}n - \frac{21}{100}$	$0,8n\ln(n) - 0,298n$

Stelle	$M \geq 2$, Asymptotisch korrekt mit Fehler $\mathcal{O}(1/n^4)$
c_0	$\frac{6}{5(M+2)}(n+1) + \frac{1}{2}$
c_1	$\frac{3}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{19}{10} - \frac{3(M^2+4M+7)}{(M+3)(M+2)(M+1)^2}\right)$
c_2	$\frac{7}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{133}{90} - \frac{9M^4+54M^3+101M^2+98M+18}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{1}{4}$
c_3	$\frac{3}{5}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{19}{10} - \frac{3(4M^3+19M^2+24M+5)}{(M+3)(M+2)(M+1)^2}\right) - 1$
c_4	$\frac{7}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{133}{90} + \frac{27M^4+144M^3+223M^2+136M-18}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{3}{4}$
s_0	$\frac{3}{5(M+2)}(n+1) + \frac{1}{4}$
s_1	$\frac{4}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{38}{45} - \frac{2(6M^4+30M^3+43M^2+22M+9)}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{1}{3}$
s_2	$\frac{4}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{38}{45} - \frac{2(6M^4+30M^3+43M^2+22M+9)}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{1}{3}$
s_3	$\frac{2}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2})\left(\frac{n+1}{5}\right) + \left(\frac{19}{45} - \frac{9M^4+45M^3+61M^2+19M-18}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{1}{4}$
$s_4 = s_5$	$\frac{6}{5(M+2)}(n+1) + \frac{1}{2}$
$C = \sum_i c_i$	$\frac{32}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{304}{45} - \frac{4(M^2+4m+9)}{3(M+3)(M+2)(M+1)^2M}\right) - \frac{3}{2}$
$S = \sum_i s_i$	$\frac{12}{15}(n+1)(\mathcal{H}_{n+1} - \mathcal{H}_{M+2}) + \left(\frac{n+1}{5}\right)\left(\frac{38}{15} - \frac{1}{M+2}\right) - \frac{5}{12}$

In beiden Fällen muss s_3 doppelt gezählt werden. Der Grund ergibt sich direkt aus dem Pseudocode.

4.5 Optimale Strategien

In diesem Abschnitt werden wir zwei weitere Partitionierungs-Strategien mit den Namen "Clairvoyant" und "Count" kennen lernen [AD13][ADH+16]. Clairvoyant greift dabei während des Partitionierungsprozesses auf ein Orakel zurück, um zu wissen, mit welchem Pivot-Element verglichen werden soll. Die Strategie Count stellt eine algorithmische Version von Clairvoyant dar.

Beide Strategien werden zur Vereinfachung nur anhand der benötigten Vergleiche verglichen werden. Wir gehen wieder von der bereits bekannten Rekursionsgleichung für Dual-Pivot Quicksort aus:

$$F_n = f_n + \frac{3}{\binom{n}{2}} \sum_{k=0}^{n-2} (n-k-1) F_k.$$

Wiederum besteht der einzige Unterschied nur in den Kosten f_n für den ersten Partitionierungsschritt. Nach der Wahl der Pivot-Elemente p und q bleiben genau $s = p - 1$ kleine Elemente, $m = q - p - 1$ mittlere Elemente und $l = n - q$ große Elemente übrig. Im Idealfall wird während der Partitionierung für kleine und große Elemente jeweils nur ein Vergleich benötigt. Um ein mittleres Element zu klassifizieren, werden dagegen zwei Vergleiche gebraucht.
Die minimale Anzahl an Vergleichen für das Sortieren von n Elementen mittels Dual-Pivot Quicksort lässt sich somit folgendermaßen berechnen: 1 Vergleich um die Pivot-Elemente richtig zu ordnen, zumindest 1 Vergleich für die übrigen $n - 2$ Elemente plus eines zusätzlichen Vergleichs für die durchschnittlich $\frac{n-2}{3}$ mittleren Elemente:

$$1 + (n-2) + \frac{n-2}{3} = \frac{4}{3}(n-2) + 1.$$

Wird nun während des Partitionierungsprozesses ein kleines Element zuerst mit q oder ein großes Element zuerst mit p verglichen, ergeben sich zusätzliche Vergleiche. Die dadurch entstehenden zusätzlichen Vergleiche werden im folgenden mit S_2, für einen zweiten Vergleich zur Klassifikation eines kleinen Elements, und L_2, für einen zweiten Vergleich zur Klassifikation eines großen Elements, bezeichnet.
Somit können nach der Durchschnittsbildung über alle p, q die Kosten für den ersten

Partitionierungsschritt wie folgt berechnet werden:

$$f_n = \frac{4}{3}(n-2) + 1 + \frac{1}{\binom{n}{2}} \sum_{s+l \leq n-2} \mathbb{E}[S_2 + L_2 | s, l]. \tag{4.23}$$

Der letzte Summand bezeichnet die zusätzlichen Kosten und ist der einzige Summand, der vom konkreten Klassifikationsalgorithmus abhängt. Für Clairvoyant werden wir diese zusätzlichen Kosten mit A_n^{cv} bezeichnen, für Count mit A_n^{ct}
Es werden nun die beiden bereits angesprochenen Klassifikationsalgorithmen Clairvoyant und Count kurz beschrieben und danach ihre jeweiligen zusätzlichen Kosten angegeben. Danach sind wir im Stande, die Partitionierungskosten anzugeben und schlussendlich die erwartete Anzahl an Vergleichen zu ermitteln. Details liefern die Arbeiten [ADH+16] und [AD13].

4.5.1 Clairvoyant

Wie bereits erwähnt, greift Clairvoyant während der Partitionierung auf ein Orakel zurück. Dadurch ist bereits am Beginn der Partitionierung bekannt, dass die zu sortierende Folge aus s kleinen und l großen Elementen besteht.
Weiters bezeichnen s_i und l_i die Anzahl der bereits klassifizierten kleinen und großen Elemente, nachdem i Elemente klassifiziert wurden. Für die Ausgangswerte gilt: $s_i = l_i = 0$.
Dann ist der Klassifikationsalgorithmus Clairvoyant folgendermaßen definiert:

> **Clairvoyant:**
> *Angenommen, die Eingabe besteht aus $s = p - 1$ kleinen und $l = n - q$ großen Elementen. Gehe bei der Klassifikation des i-ten Elements, für $1 \leq i \leq n-2$, folgendermaßen vor: wenn $s - s_i \geq l - l_i$ gilt, vergleiche zuerst mit p, ansonsten vergleiche zuerst mit q.*

An dieser Stelle sei nochmals darauf hingewiesen, dass Clairvoyant nicht implementiert werden kann, da er Zugang zu einem Orakel benötigt.
In [AD13, Sec. 6] wurde gezeigt, dass diese Strategie die geringsten Klassifikationskosten für alle Strategien mit Zugang zu einem Orakel aufweist. Aus diesem Grund ist es eine optimale Strategie.

Für die zu erwartenden zusätzlichen Kosten A_n^{cv} ergibt sich:

$$\mathbb{E}[A_n^{cv}] = \frac{n}{6} - \frac{7}{12} + \frac{1}{4(n - [n\ gerade])} - \mathbb{E}[X_n^{\searrow}],$$

wobei $[n]$ die Indikatorfunktion bezeichnet und für $\mathbb{E}[X_n^{\searrow}]$ folgendes gilt: $\mathbb{E}[X_n^{\searrow}] = \mathbb{E}[X_n^{\nearrow}] - \frac{1}{2} + \frac{1}{2(n-[n\ gerade])}$.
Für $\mathbb{E}[X_n^{\nearrow}]$ gilt: $\mathbb{E}[X_n^{\nearrow}] = \frac{1}{2}\mathcal{H}_{n-2}^{ungerade} - \frac{1}{8} + \frac{(-1)^n}{8(1-[n\ gerade])}$ mit $\mathcal{H}_n^{ungerade} = \sum_{m=1}^{n} \frac{[m\ ungerade]}{m}$.
Mit $\mathbb{E}[X_n^{\searrow}]$ und $\mathbb{E}[X_n^{\nearrow}]$ werden die erwarteten Anzahlen an sogenannten „down-to-zero“ und „up-to-zero“ Pfaden in Gitter bezeichnet. Mehr Details findet der interessierte Leser in [ADH+16, Seite 7].

Gemeinsam mit den zusätzlichen Kosten ergeben sich für die Partitionierungskosten von Clairvoyant:

$$\begin{aligned} f_n &= \frac{4}{3}(n-2) + 1 + \mathbb{E}[A_n^{cv}] \\ &= \frac{3}{2}n - \frac{9}{4} + \frac{1}{4(n - [n\ gerade])} - \mathbb{E}[X_n^{\searrow}]. \end{aligned}$$

Für die zu erwartende Anzahl an Vergleichen ergibt sich schlussendlich die folgende asymptotische Näherung mit Fehler $\mathcal{O}(\ln(n))$:

$$C_n^{cv} = 1,8n\ln(n) - 2,65n.$$

4.5.2 Count

Count:

Gehe bei der Klassifikation des i-ten Elements, für $1 \leq i \leq n-2$, folgendermaßen vor: wenn $s_i \geq l_i$ gilt, vergleiche zuerst mit p, ansonsten vergleiche zuerst mit q.

Für die zu erwartenden zusätzlichen Kosten A_n^{cn} von Count ergibt sich:

$$\mathbb{E}[A_n^{ct}] = \frac{n}{6} - \frac{7}{12} + \frac{1}{4(n - [n\ gerade])} + \mathbb{E}[X_n^{\nearrow}].$$

Gemeinsam mit den zusätzlichen Kosten ergeben sich für die Partitionierungskosten von

Count:

$$\begin{aligned} f_n &= \frac{4}{3}(n-2) + 1 + \mathbb{E}[A_n^{ct}] \\ &= \frac{3}{2}n - \frac{9}{4} + \frac{1}{4(n - [n\ gerade])} + \mathbb{E}[X_n^{\nearrow}]. \end{aligned}$$

Für die zu erwartende Anzahl an Vergleichen ergibt sich schlussendlich die folgende asymptotische Näherung mit Fehler $\mathcal{O}(\ln(n))$:

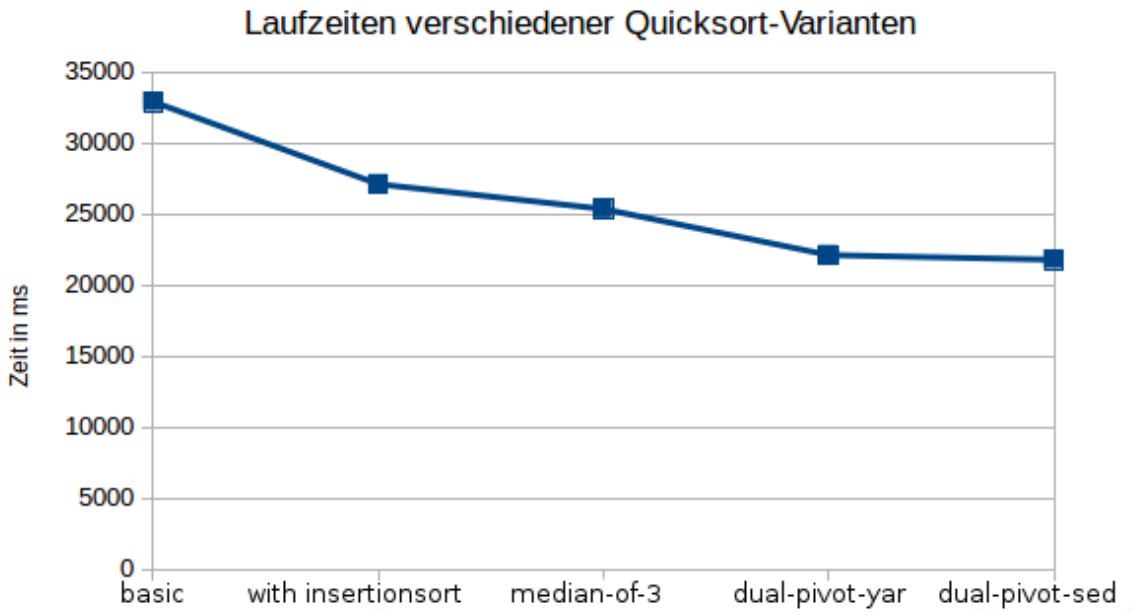
$$C_n = 1.8n\ln(n) - 2,382n.$$

5 Praktische Umsetzung

Der letzte Teil dieser Diplomarbeit widmet sich der praktischen Umsetzung der besprochenen Varianten von Quicksort.
Die theoretisch erhaltenen Resultate konnten dabei gut bestätigt werden.

Das folgende Szenario stellt die Grundlage der Analyse dar: Es waren 100000 verschiedene Integer-Werte zu sortieren. Nach 12 Durchläufen wurden der schnellste und der langsamste Durchlauf gestrichen und der Mittelwert aus den verbleibenden 10 ermittelt, sortiert wurde für jede Variante die gleiche Permutation an Werten.
Ausgeführt wurden die Berechnungen auf einem Lenovo ThinkPad-X220 mit einem Intel Core i5-2520M Prozessor mit 2,5GHz.
Die Zeit wurde in Millisekunden gemessen, bis auf die Basis-Variante verwenden alle Varianten $M = 9$ als Grenze für die Verwendung von Insertionsort.
Es ergeben sich die folgende Tabelle und Grafik auf der nächsten Seite:

basic	with insertionsort	median-of-3	dual-pivot-yar	dual-pivot-sed
(34769) max	(19057) min	25274	19163	21975
34540	30883	25474	22328	22162
31597	26454	(29835) max	(29081) max	18836
31786	23745	22264	(17948) min	22111
33932	25498	19451	24454	24175
32718	23086	28969	18727	22671
(30401) min	31167	26094	23832	16810
33402	(32029) max	28926	19679	(14083) min
32436	24512	29454	23217	19848
31742	28999	29440	24582	25728
33190	30503	17323	22641	23664
33804	26324	(14143) min	22667	(27362) max
∅ 32914 ms	∅ 27117 ms	∅ 25266 ms	∅ 22129 ms	∅ 21798 ms



Listing 1: Das Main-Programm

```
int main(){
        int tmp,i=0,j;
        int a[MAX];
        double time=0.0, tstart;
        FILE *in,*out;

        in=fopen("qs.in","r");
        while(fscanf(in,"%d",&tmp)!=EOF){
                a[i]=tmp;
                i++;
        }
        fclose(in);

        tstart=clock();   //Start der Zeitmessung
        quicksort(a,0,i-1);
        insertionsort(a,0,i-1); //Bei der Basis Variante fehlt diese Zeile
        time+=(clock()-tstart);

        printf("%f␣Millisekunden\n",time);

        out=fopen("qs.out","w+");
        for(j=0;j<i;j++){
                fprintf(out,"%d%s",a[j],(j+1)%10==0 ? "\n" : "␣");
        }
        fclose(out);

        return 0;
}
```

Listing 2: Basic Quicksort

```
void quicksort(int *A, int l, int r){
  int i,j,pivot;

  if (r-l <= 1) return;

  pivot = A[l];

  i, j;
  for (i = l+1, j = r; ; i++, j--)
  {
    while (A[i] < pivot) i++;
    while (A[j] > pivot) j--;

    if (i >= j) break;

    swap(A,i,j);
  }
  swap(A,l,j);

  quicksort(A, l , j-1);
  quicksort(A, j + 1, r);
```

```
}
```

Listing 3: Quicksort mit Insertionsort

```
void quicksort(int *A, int l, int r){
        int i,j,pivot;

        if (r-l <=9) return;
        else{
                pivot = A[l];

                i, j;
                for (i = l+1, j = r; ; i++, j--){
                        while (A[i] < pivot) i++;
                while (A[j] > pivot) j--;

                        if (i >= j) break;

                swap(A,i,j);
                }
                swap(A,l,j);

                quicksort(A, l, j - 1);
                quicksort(A, j + 1, r);
  }
}
```

Listing 4: Insertionsort

```
void insertionsort(int *A, int l, int r){
        int i,j, temp;
        for( i = l+1; i <= r; i++) {
                temp = A[i];
                j = i;
                while(j > 0 && temp < A[j - 1]) {
                        A[j] = A[j - 1];
                        j--;
                }
                A[j] = temp;
        }
}
```

Listing 5: Quicksort mit Median-of-Three

```
void quicksort(int *A, int l, int r){
        int i,j,pivot;

        if (r-l <=9) return;
        else{
                swap(A,l+1,(l+r)/2);
                if(A[l+1]>A[r]) swap(A,l+1,r);
                if(A[l]>A[r]) swap(A,l,r);
                if(A[l+1]>A[r]) swap (A,l+1,r);
```

```
                pivot = A[l];

                i, j;
                for (i = l+1, j = r; ; i++, j--){
                        while (A[i] < pivot) i++;
                while (A[j] > pivot) j--;

                        if (i >= j) break;

                swap(A,i,j);
                }
                swap(A,l,j);

                quicksort(A, l, j - 1);
                quicksort(A, j + 1, r);
  }
}
```

Listing 6: Dual-Pivot Quicksort mit Partitionierung nach Yaroslavskiy

```
void dualpivotquicksortyaroslavskiy(int *A, int left, int right){
        int l,g,k,p,q,tmp;

        if(right-left <= 9) return;
        else{

                if(A[left]>A[right]){
                        swap(A,left,right);
                }
                p=A[left];
                q=A[right];

                l=left+1;
                g=right-1;
                k=l;
                while(k<=g){
                        if(A[k]<p){
                                swap(A,k,l);
                                l++;
                        }
                        else{
                                if(A[k] >= q){
                                        while(A[g]>q && k<g){
                                                g--;
                                        }
                                        swap(A,g,k);

                                        g--;
                                        if(A[k]<p){
                                                swap(A,k,l);
                                                l++;
                                        }
                                }
```

```
                        }
                        k++;
                }

                l--;
                g++;

                swap(A,left,l);
                swap(A,right,g);

                dualpivotquicksortyaroslavskiy(A, left , l-1);
                dualpivotquicksortyaroslavskiy(A, l+1, g-1);
                dualpivotquicksortyaroslavskiy(A, g + 1, right);
        }
}
```

Listing 7: Dual-Pivot Quicksort mit Partitionierung nach Yaroslavskiy

```
void dualpivotquicksortsedgewick(int *A, int left, int right){
        int i,i1,j,j1,p,q,tmp,check;

        if(right-left <= 9) return;
        else{
                i=left;
                i1=left;
                j=right;
                j1=right;

                if(A[left]>A[right]){
                        swap(A,left,right);
                }
                p=A[left];
                q=A[right];

                check=1;
                LOOP:while(check){
                        i++;
                        while(A[i] <= q && check !=0){
                                if(i>=j){
                                        check=0;
                                        goto LOOP;
                                }
                                if(A[i]<p){
                                        A[i1]=A[i];
                                        i1++;
                                        A[i]=A[i1];
                                }
                                i++;
                        }
                        j--;
                        while(A[j]>=p && check!=0){
                                if(A[j]>p){
                                        A[j1]=A[j];
```

```
                                        j1--;
                                        A[j]=A[j1];
                                }
                                if(i>=j){
                                        check=0;
                                        goto LOOP;
                                }
                                j--;
                        }
                        A[i1]=A[j];
                        A[j1]=A[i];
                        i1++;
                        j1--;
                        A[i]=A[i1];
                        A[j]=A[j1];
                }
                A[i1]=p;
                A[j1]=q;

                dualpivotquicksortsedgewick(A, left , i1-1);
                dualpivotquicksortsedgewick(A, i1+1, j1-1);
                dualpivotquicksortsedgewick(A, j1 + 1, right);
        }
}
```

Literatur

[AD13] M. Aumüller and M. Dietzfelbinger. Optimal partitioning for dual pivot quicksort. In *Automata, Langueges, and Programming*, pages 33–44. Springer Nature, 2013.

[ADH+16] M. Aumüller, M. Dietzfelbinger, C. Hauberger, D. Krenn, and H. Prodinger. Couting zeros in random walks on the integers and analysis of optimal dual-pivot quicksort. 2016. Preprint.

[CSRL09] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. The MIT Press, 3. edition, 2009.

[GKP94] R. Graham, D. Knuth, and O. Patashnik. *Concrete mathematics: a foundation for computer science*. Addision-Wesley, 1994.

[Hen89] P. Hennequin. Combinatorial analysis of quicksort algorithm. *Informatique théorique et applications*, 23(3), 1989.

[Hen91] P. Hennequin. *Analyse en moyenne d'algorithmes : tri rapide et arbres de recherche.* PhD thesis, Ecole Politechnique, 1991.

[Hoa61] C. A. R. Hoare. Algorithm 63, partition; algorithm 64, quicksort. *Communications of the ACM*, 4(7):321, 1961.

[Hoa62] C. A. R. Hoare. Quicksort. *The Computer Journal*, (5):10–16, 1962.

[KLOMQ14] S. Kushagra, A. López-Ortiz, J. I. Munro, and A. Qiao. Multi-pivot quicksort: Theory and experiments. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*, pages 47–60. Society for Industrial and Applied Mathematics, May 2014.

[Knu76] D. Knuth. Big omicron and big omega and big theta. *SIGACT News*, April-June 1976.

[Knu98] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1998.

[Knu05] D. Knuth. *The Art of Computer Programming: Volume 1, Fascicle 1. MMix, A RISC Computer for the new Millennium.* Addison-Wesley, 2005.

[Kus11] N. Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie*. Springer-Verlag/Wien, 2011.

[Sed75] R. Sedgewick. *Quicksort*. PhD thesis, Standford University, 1975.

[Sed77] R. Sedgewick. The analysis of quicksort programs. *Acta Informatica*, 7(4):327–355, 1977.

[vE70] M. van Emden. Increasing the efficiency of quicksort. *Communications of the ACM*, 13(9):563–567, 1970.

[Wil94] H. Wilf. *generatingfunctionology*. Academic Press, 2. edition, 1994.

[Wil12] S. Wild. Java 7's dual pivot quicksort. Master's thesis, University of Kaiserslautern, 2012.

[WNN15] S. Wild, M. Nebel, and R. Neininger. Average case and distributional analysis of dual-pivot quicksort. *ACM Transactions an Algorithms*, 11(3), 2015.

[Yar09] V. Yaroslavskiy. Dual-pivot quicksort algorithm. 2009. `http://codeblab.com/wp-content/uploads/2009/09/DualPivotQuicksort.pdf`, Zuletzt aufgerufen: 2.2.2018.