Quicksort

Finn Jannsen, Philipp Schwarz

22. April 2019

Inhaltsverzeichnis

1 Einführung

Diese Dokumentation beschreibt eine Quicksort-Implementation für Arrays über Integer. Zusätzlich gibt es die Möglichkeit drei verschiedene Pivots zu wählen: Rechtes Element, Median of Three und Zufällig. In Abschnitt 2 wird darauf eingegangen, wie der Algorithmus realisiert wurde. Anschließend wird in Abschnitt 3.1 geprüft, ob die Implementation korrekt funktioniert und in Abschnitt 3.2 die Performance der Pivots verglichen.

2 Implementation

Der Algorithmus wurde als Kasse realisiert, der ein Interface implementiert, welches es ermöglicht, einfach weitere Implementationen, sofern dies gewünscht wird, zu schreiben.

2.1 Pivotauswahl

Es gibt drei Verschiedene Pivots zur Auswahl:
RIGHT: es wird das Rechte Element als Pivot genommen.
MEDIANOFTHREE: der linke, mittlere und rechte Index werden addiert und durch 3 geteilt und das Ergebnis ist der Index des Pivots.
RANDOM: ein zufälliges Element ist das Pivot.

Der Code ist in 3.3 dargestellt. Das gewählte Pivot wird in allen drei Fällen mit dem rechtesten Element der Liste getauscht.

2.2 Sortieren

Der Algorithmus der Sortiert ist im wesentlichen eine Schleife, in der zuerst das erste Element von links, welches gleich oder größer als Pivot ist, gesucht wird. Danach wird das erste Element von rechts, welches kleiner als das Pivot ist, gesucht. Diese werden dann getauscht. Die Schleife wird verlassen, wenn i und j in der Mitte angekommen sind, also entweder gleich sind oder j kleiner als i. Nach der Schleife wird j mit dem Pivot getauscht, damit das Pivot ganz rechts steht und der Algorithmus wird zwei mal rekursiv gestartet, einmal um die Liste von ganz links bis i-1 zu sortieren und nocheinmal um die Liste von i+1 bis rechts zu sortieren.

Der Code ist in 3.4 dargestellt.

3 Testen und Verifikation

3.1 Verifizieren

Der Algorithmus und alle Pivots wurden auf ihre Funktionalität getestet. Alle Tests wurden erfolgreich absolviert.

3.2 Aufwandsanalyse

Für die Aufwandsanalyse wurde der Klasse ein Counter eingeführt, der die Tauschoperationen zählt. Die Beobachtungen sind in Abbildung 3.1 und 3.2 zu sehen.

Es fällt auf, dass die Pivots sich nur durch konstante Faktoren unterscheiden, wobei das Rechte Element als Pivot deutlich weniger Tauschoperationen hat, was sich dadurch erklärt, dass man bei diesem nicht erst das Pivot mit dem rechten Element tauschen muss.

Zwischen Best- bzw. Average Case und Worst Case sind keine Unterschiede sichtbar.

Rechenoperationen bei Best- und Average Case:

$$f(n) = n * ln(n)$$

Rechenoperationen bei Worst Case:

$$g(n) = n^2$$

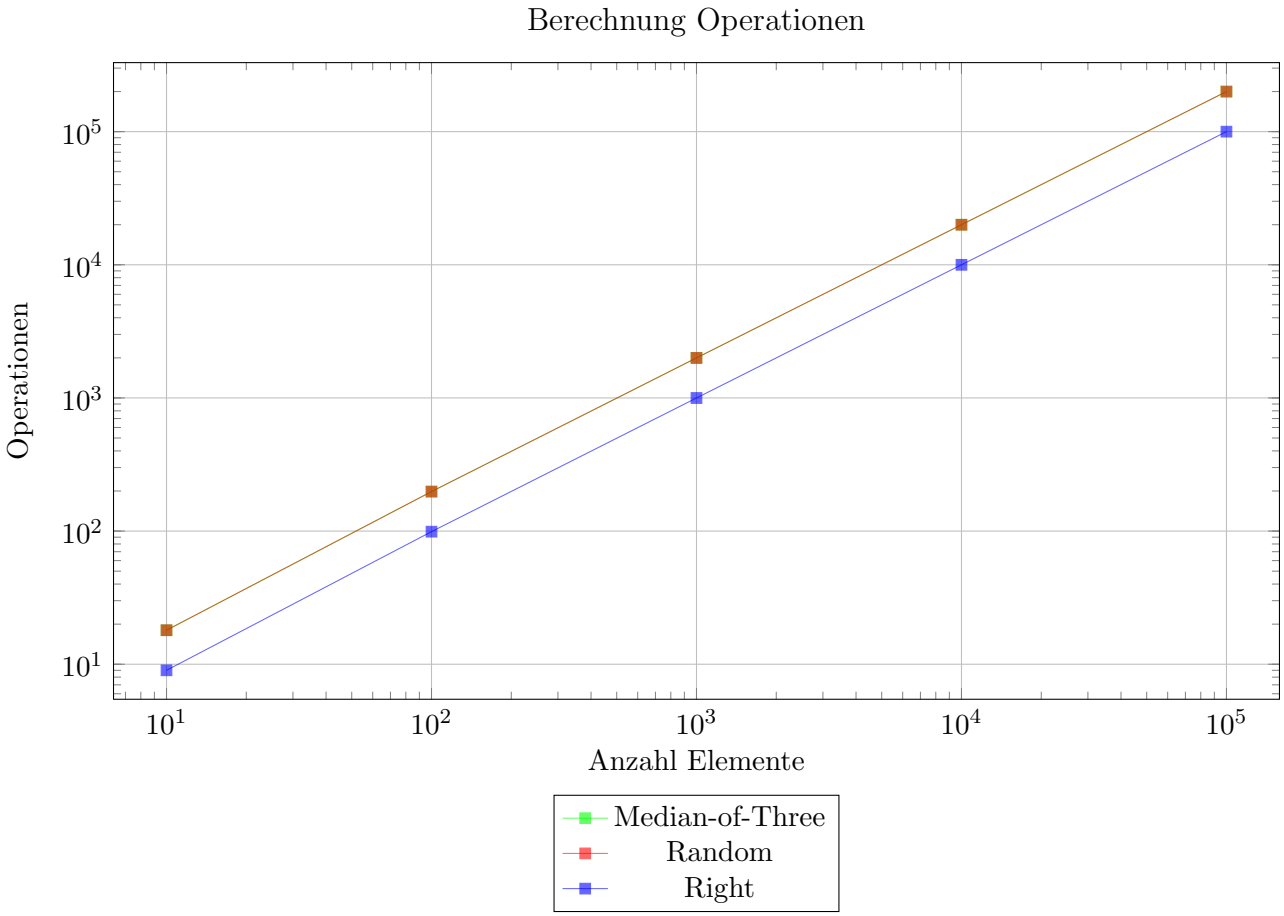

Abbildung 3.1: Quantitativer Vergleich der Best-/Average-Case Quicksort-Verfahren unterschiedlicher Pivots

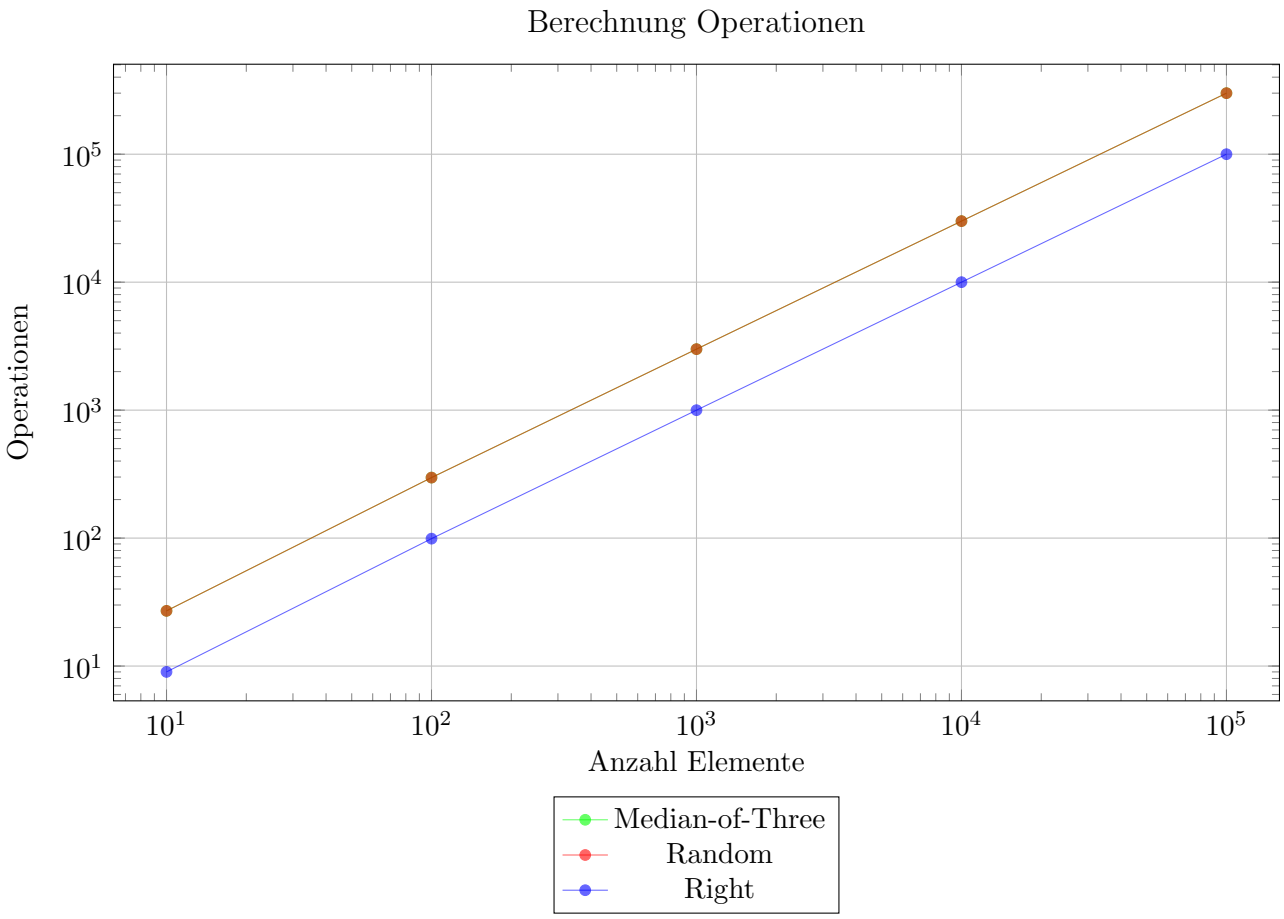

Abbildung 3.2: Quantitativer Vergleich der Worst-Case Quicksort-Verfahren unterschiedlicher Pivots

```
// swap the selected pivot to the last part of the list
switch (pivotType) {
    case RIGHT:
        break;
    case MEDIANOFTHREE:
        swap(list, (left + (right-left)/2 + right) / 3, right);
        break;
    case RANDOM:
        double index = Math.random() * right;
        swap(list, (int) index, right);
        break;
}
```

Abbildung 3.3: Code-Ausschnitt Pivotsuche

```
iterationLoop:
while(true) {
    // increase i, which starts as the smallest index,
    // until the Node it indexes is equal or larger than the pivot
    while(i < right && list[i].getKey() < pivot) {
        i++;
    }

    // decrease j, which starts as the largest index,
    // until the Node it indexes is smaller than the pivot
    while(j > 0 && list[j].getKey() >= pivot) {
        j--;
    }

    // break out of the loop when we iterated over every entry
    if (i >= j) {
        break iterationLoop;
    }
    // swap i, which points at an element larger than the pivot with j,
    // which points at an element smaller than the pivot
    swap(list, i, j);
}

// swap the pivot to the right part
swap(list, i, right);
// sort everything right and left of the pivot
sort(list, left, i-1, pivotType);
sort(list, i+1, right, pivotType);
```

Abbildung 3.4: Code-Ausschnitt Sortieren