

Der Sortieralgorithmus »Quicksort« und seine
Implementation in Java mithilfe der
NRW-Landesklasse für die Datenstruktur Liste

Heiner Stroick

Version: 10. April 2020

Fach:	Informatik
Kurs:	Q1 Grundkurs 1
Betreuer Lehrer:	Herr Stroick
Themenausgabe:	14. März 2021
Abgabe:	01. August 2021

Inhaltsverzeichnis

1	Informationen zu dieser Mini-Facharbeit	3
2	Einleitung	4
2.1	Historie des Quicksort-Algorithmus	5
2.2	Divide and Conquer	5
2.3	Persönlicher Bezug zum Thema	6
2.4	Aufbau der Facharbeit	6
3	Der Quicksort-Algorithmus	8
3.1	Idee des Algorithmus	8
3.2	Laufzeit	9
3.2.1	Best-Case	9
3.2.2	Worst-Case	10
3.2.3	Average-Case	11
3.2.4	Praxisrelevanz	12
3.2.5	Untere Schranke für vergleichsbasierte Sortierverfahren	12
3.3	Speicherplatz	13
4	Implementation	14
4.1	Die generische Landesklasse <code>List</code>	14
4.2	Andere Klassen des Projekts	15
4.3	Klassendiagramm	16
4.4	Implementation des Quicksort-Algorithmus	16
4.4.1	Pseudocode	17
4.4.2	Java-Quelltext	19
4.4.3	Ausgabe	22
5	Fazit	23
5.1	Blick in die Praxis	23
5.2	Ausblick auf andere Algorithmen	23
5.3	Schwierigkeiten beim Bearbeiten der Facharbeit	24

6 Weitere Informationen	25
6.1 Quellenrecherche für die Facharbeit	25
6.2 Weitere Informationen zu Latex	25
6.3 Programme für Informatik	27
6.4 Programme für Mathematik	29
6.5 Hans-Riegel-Fachpreis	30
Literaturverzeichnis	31
Abbildungsverzeichnis	33
Tabellenverzeichnis	34
Algorithmenverzeichnis	35
Quelltextverzeichnis	36
Anhang	37
Selbstständigkeitserklärung	41

1 Informationen zu dieser Mini-Facharbeit

Diese „Mini-Facharbeit“ dient zu Demonstrationszwecken und erhebt dabei keinen Anspruch auf Vollständigkeit oder eine angemessene inhaltliche Tiefe. Sie soll den prinzipiellen Aufbau von Kapiteln anhand eines bekannten Beispiels verdeutlichen (nicht alle Kapitel sind vollständig ausformuliert, vgl. z. B. Kap. 2.3). Dabei wird zwischen »Erklärung des Algorithmus« und »Implementation des Algorithmus« unterschieden (Kap. 3 und Kap. 4).

Die `tex`-Datei kann dabei als Vorlage für die eigene Facharbeit dienen. Es wurde zudem versucht, verschiedene Möglichkeiten von Latex aufzuzeigen (z. B. Aufzählungen, Stichwortlisten, Textauszeichnungen, Verweise, Zitate, Literaturverweise, eingebundene Bilder, Einbinden von Pseudocode und Quelltexten, ...), sodass die `tex`-Datei auch als Nachschlagewerk am praktischen Beispiel verstanden werden kann.

Für die eigene Facharbeit muss dann am Ende selbst entschieden werden, ob und wie die Reihenfolge der (Unter-)Kapitel vielleicht angepasst werden muss (evtl. ist auch nicht die Darstellung in der vorliegenden Kleinschrittigkeit sinnvoll). Gleiches gilt für die im Anhang aufgeführten Verzeichnisse (in dieser Facharbeit sind die Verzeichnisse aufgeführt, um die Möglichkeiten von Latex zu demonstrieren).

Diese Vorlage wurde unter https://github.com/hnrstrck/vorlage_facharbeit veröffentlicht. Das Datum auf der Titelseite entspricht der „Version“ dieser Vorlage.

Viel Erfolg!

10. April 2020 – H. STROICK

2 Einleitung

Sortierten Datenmengen begegnen wir jeden Tag: Sei es die nach Punkten sortierte Bundesligatabelle, die chronologisch sortierten E-Mails (mehrerer Accounts) in der Inbox oder die nach Namen sortierten Dateien in einem Ordner auf dem Computer. Das Zurechtfinden gelingt in sortierten Daten einfacher als in Unsortierten.

Dabei ist das Herstellen von Ordnung – das Sortieren an sich – kein triviales Problem. In der Informatik wird versucht, unter Berücksichtigung verschiedener Zielvorgaben Sortieralgorithmen zu entwickeln und zu studieren. Häufig wird verlangt, dass Algorithmen unsortierte Daten (wie bspw. in Abb. 1) möglichst speicherplatzeffizient (so wenig RAM wie möglich nutzend), mit möglichst wenig Vertauschungen oder mit möglichst wenig Vergleichen zu sortieren. Zum Teil werden diese Anforderungen kombiniert.

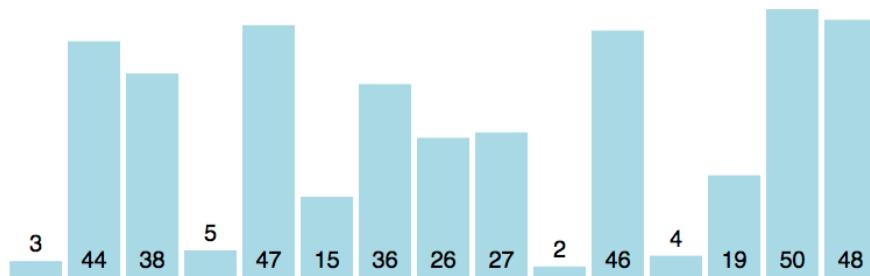


Abbildung 1: Unsortierte Daten. Bild: Screenshot von [VisuAlgo].

Es gibt eine Vielzahl an verschiedenen Sortieralgorithmen, die nach unterschiedlichen Prinzipien funktionieren. Beim Sortieren ist es allerdings unerheblich, ob lexikographisch, chronologisch oder direkt nach Zahlen sortiert wird. Durch Umrechnungen kann jedes Sortierproblem auf ein Sortierproblem von Zahlen zurückgeführt werden (bspw. durch den Einsatz des ASCII-Codes zum Umrechnen von Buchstaben in Zahlen).

Es sind evtl. auch noch andere Kriterien für die Wahl eines passenden Sortieralgorithmus ausschlaggebend: Steht zu Beginn die Anzahl der zu sortierenden Daten noch nicht fest (weil die Daten bspw. nach und nach eintreffen), ist ein Algorithmus zu wählen, der nicht den „Überblick“ über alle Daten verlangt.

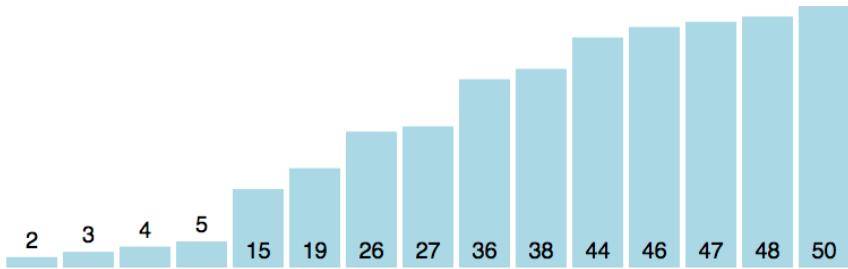


Abbildung 2: Sortierte Daten. Bild: Screenshot von [VisuAlgo].

Ziel ist die korrekte Sortierung der Ausgangsdaten (wie in Abbildung 2 dargestellt). [Wiki SortVerf] gibt einen guten Überblick über verschiedene Sortierverfahren. Die dort veröffentlichte Tabelle zeigt auch sehr schön, wie sich verschiedene Algorithmen in ihren Laufzeiten unterscheiden, d. h. wie „schnell“ oder „langsam“ sie eine Menge von n Zahlen sortieren (dafür wird die sog. \mathcal{O} -Notation¹ verwendet).

2.1 Historie des Quicksort-Algorithmus

Der Quicksort-Algorithmus wurde von Tony HOARE² entwickelt und 1962 in [Hoare 1962] erstmalig vorgestellt und analysiert. Quicksort ist ein Algorithmus, der sich insbesondere durch seine Geschwindigkeit, seine effektive Speicherplatznutzung und seine simple Programmierung auszeichnet („The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming“, [Hoare 1962, S. 1]). Seit seiner Veröffentlichung wurde der Algorithmus von vielen Informatikern analysiert und verbessert (vgl. [Wiki QS]).

2.2 Divide and Conquer

Quicksort setzt die Divide-and-Conquer-Strategie (auch D&C, »Teile und Herrsche«) zum Sortieren ein. Eine Erklärung dieser Strategie wird von HOARE in seinem Paper gegeben (vgl. [Hoare 1962, S. 1]); an dieser Stelle sei der Einfachheit halber auf die in [Lang DuC] aufgeführten Erläuterungen verwiesen. Ein Lösungsansatz, der nach dem Divide-and-Conquer-Prinzip arbeitet, gliedert sich immer in dieselben drei Schritte:

¹Informationen: Linux Related (2014): *Laufzeitkomplexität von Algorithmen – die O-Notation*, <http://www.linux-related.de/index.html?/coding/o-notation.htm>, besucht am 09.04.2020.

²„Sir Charles Antony Richard HOARE (* 11. Januar 1934 in Colombo, Sri Lanka), besser bekannt als Tony HOARE oder C.A.R. HOARE, ist ein britischer Informatiker. HOARE erlangte hohes Ansehen durch die Entwicklung des Quicksort-Algorithmus sowie des HOARE-Kalküls, durch den sich die Korrektheit von Algorithmen beweisen lässt.“ [Wiki Hoare]

Divide	Das Problem wird in Teilprobleme zerlegt.
Conquer	Die Teilprobleme werden gelöst.
Combine	Die Lösungen der Teilprobleme werden zusammengefügt, so dass sie die Lösung des ursprünglichen Problems ergeben.

Tabelle 1: Die Schritte der Divide-and-Conquer-Strategie zur Lösung eines Problems nach [Lang DuC].

Quicksort setzt die Divide-and-Conquer-Strategie sehr geschickt ein, um Daten zu sortieren. Dies wird in Kapitel 3.1 erläutert.

2.3 Persönlicher Bezug zum Thema

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.4 Aufbau der Facharbeit

In Kapitel 3 wird der Quicksort-Algorithmus in seiner Theorie vorgestellt. Nachdem die Idee des Algorithmus in Kapitel 3.1 erläutert wurde, wird die Laufzeit von Quicksort in Kapitel 3.2 in den Blick genommen.

Kapitel 4 widmet sich der Implementation. Da für die Programmierung in Java die NRW-Landesklasse `List` verwendet, wird diese zunächst vorgestellt (Kap. 4.1). Die anderen Klassen des Projekts werden in Kapitel 4.2 erläutert; das Klassendiagramm in Kapitel 4.3 verschafft einen Überblick. Nach der Formulierung von des Quelltextes in Pseudocode (Kap. 4.4.1) wird der eigentliche Java-Quelltext kleinschrittig erklärt (Kap. 4.4.2). Zum Teil wird an bekanntes Wissen aus dem Unterricht angeknüpft.

Das Fazit in Kapitel 5 fasst die Erkenntnisse dieser Facharbeit noch einmal zusammen und gibt einen Ausblick, der auch andere Sortierverfahren umfasst (Mergesort, Heapsort).

Kapitel 6 gibt Tipps zum Umgang mit Latex und zum Erstellen einer Facharbeit (u. a. auch zur Literaturrecherche). Es wird außerdem auf hilfreiche Programme verwiesen, bspw. zum Erstellen von Grafiken / Zeichnungen, UML-Diagrammen oder Plotten von Funktionsgraphen (für eine Facharbeit in Mathematik).

3 Der Quicksort-Algorithmus

3.1 Idee des Algorithmus

Ausgangspunkt ist eine Menge an zu sortierenden Elementen. Zuerst wird ein Element, das *Trennelement* oder *Pivotelement*³, bestimmt. Dann werden alle Elemente mit diesem Pivotelement nacheinander verglichen. Sind Elemente kleiner als das Pivotelement, kommen sie auf die linke Seite, sind sie größer als das Pivotelement, kommen sie auf die rechte Seite. Nun ist offenkundig, dass sich das Pivotelement schon an der richtigen Position befindet und an seiner Position sortiert vorliegt. Dann wird dieses Vorgehen auf die linke bzw. rechte Seite (mehrfach) rekursiv angewendet. Das Verfahren hört auf, wenn nur noch einelementige Mengen vorliegen (vgl. [Hoare 1962][Wiki QS]).

Die Idee wird in [Lang 2018] etwas formaler vorgestellt:

Zunächst wird die zu sortierende Folge a so in zwei Teilstücke b und c zerlegt, dass alle Elemente des ersten Stücks b kleiner oder gleich allen Elementen des zweiten Stücks c sind (*divide*). Danach werden die beiden Teilstücke sortiert, und zwar rekursiv nach demselben Verfahren (*conquer*). Wieder zusammengesetzt ergeben die Teilstücke die sortierte Folge (*combine*).

[Lang 2018]

Quicksort ist ein nicht-stabiles Sortierverfahren (vgl. [Wiki QS] und [Wiki Stab]). Das bedeutet, dass sich die Reihenfolge von Datensätzen ändern kann, wenn nach verschiedenen Sortierschlüsseln sortiert wird. Beispiele finden sich unter [Wiki Stab].

Die Idee von Quicksort kann auch nonverbal dargestellt werden. Folgende Anleitung 3 ist im Stil eines großen schwedischen Möbelhauses gehalten und kommt ohne weitere Erklärungen aus.

³fraz. *pivot* (Drehpunkt)

KVICK SÖRT

idea-instructions.com/quick-sort/
v1.1, CC by-nc-sa 4.0 **IDEA**

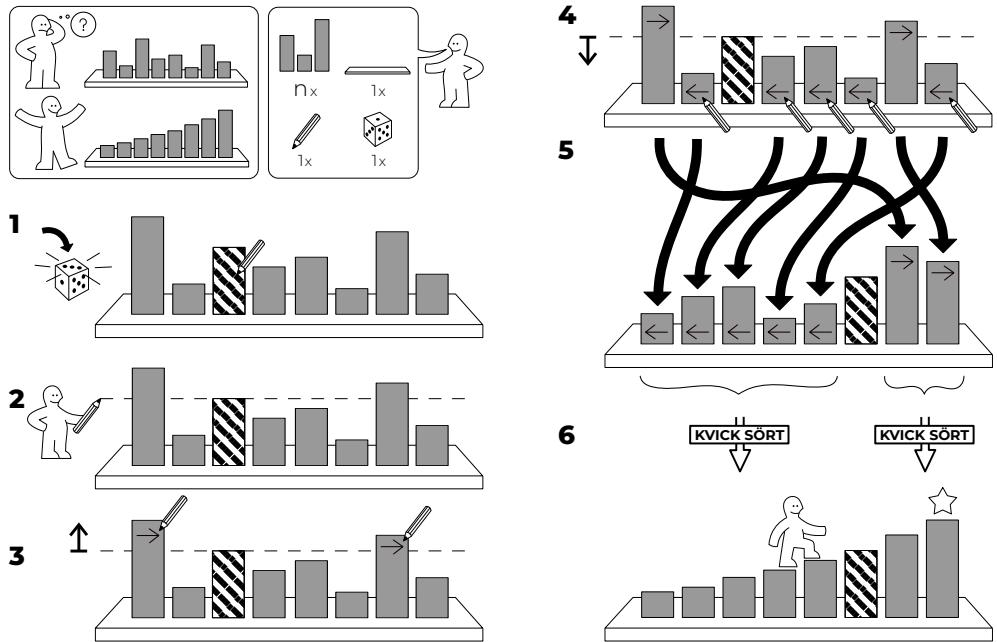


Abbildung 3: Der Ablauf des Quicksort-Algorithmus als Bauanleitung »KVICK SÖRT«. Bild: [IDEA]

3.2 Laufzeit

Bei der Bewertung der Laufzeit von Quicksort ist die Problemgröße die Anzahl der zu sortierenden Zahlen – offensichtlich steigt die Laufzeit, wenn mehr Zahlen zu sortieren sind. Die Problemgröße wird mit n bezeichnet.

Entscheidend für die Geschwindigkeit, mit der eine Datenmenge sortiert wird, ist die Wahl eines passenden Pivotelements, wie die folgenden Kapitel zeigen werden.

3.2.1 Best-Case

Im besten Falle trennt das Pivotelement die Liste genau so, dass linke rechte Liste gleichgroß sind, also gleichviele Elemente enthalten. Damit wird die geringste Rekursionstiefe erreicht (vgl. Abb. 4).

Natürlich ist hier die Schwierigkeit, das Pivotelement entsprechend zu bestimmen. Statistische Verfahren können genutzt werden, um möglichst effizient ein entsprechendes Pivotelement zu ermitteln (z. B. mit der Bestimmung eines Medians).

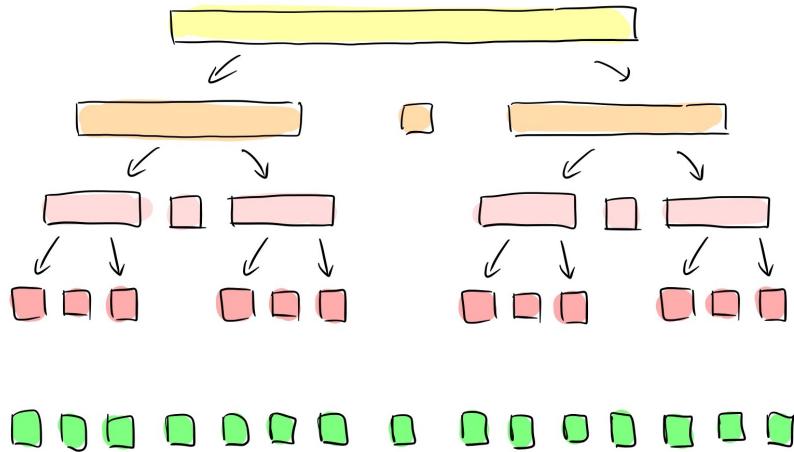


Abbildung 4: Das Pivotelement trennt eine Liste stets in zwei gleichgroße linke und rechte Listen. Dadurch bleibt die Rekursionstiefe minimal – hier wird nur eine Rekursionstiefe von $t = 3$ erreicht.

Es ergibt sich im besten Falle eine Laufzeit von

$$\mathcal{O}(n \cdot \log_2(n)),$$

genaugenommen sogar von $\Theta(n \cdot \log_2(n))$. Auf einen Beweis wird hier verzichtet.⁴

3.2.2 Worst-Case

Im schlechtesten Fall wird in jedem Quicksort-Aufruf immer das größte (oder kleinste) Element als Pivotelement ausgewählt, sodass beim Aufteilen in linke und rechte Liste eine von beiden leer ist (dies kann auch bspw. abwechselnd passieren). Dadurch wird die maximale Rekursionstiefe erreicht. Dies ist insbesondere dann der Fall, wenn die zu sortierenden Daten schon sortiert vorliegen (und als Pivotelement immer das erste oder letzte Element gewählt wird).

⁴Hinweis: In einer echten Facharbeit wäre dieser aber zu führen. Wie dies aussehen könnte, wird in [Khan 2020] deutlich.

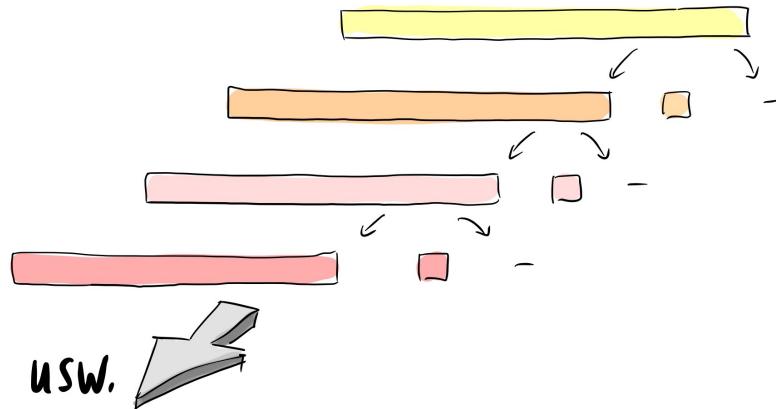


Abbildung 5: Entwicklung der Rekursion im schlechtesten Fall. Das Pivotelement trennt die Listen so, dass eine Seite immer leer bleibt. Dabei ist natürlich unerheblich, ob *immer* die rechte Liste leer bleibt oder ob dies abwechselnd (links und rechts) passiert. Es wird mit $t = 14$ die maximale Rekursionstiefe erreicht.

Es ergibt sich nach [Kempe Löhr 2015, S. 119] dabei der Wert

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

für die Anzahl der Vergleiche. Die Laufzeit ist also quadratisch, also entsprechend $\Theta(n^2)$, da die Anzahl an Vergleichen den Zeitaufwand für das Sortieren im Wesentlichen bestimmt. In [Khan 2020] wird dies weiter ausgeführt.

Diese Summe aus (1) kann durch Anwenden der GAUSS'schen Summenformel

$$1 + 2 + 3 + \dots + n = \sum_{k=1}^n k = \frac{1}{2}n(n + 1) \quad (\text{GAUSS})$$

vereinfacht werden. Auf einen Beweis der Formel (GAUSS) wird hier aber verzichtet.

3.2.3 Average-Case

Ein intuitiver „Beweis“ der mittleren Laufzeit wird bspw. bei [Khan 2020] gegeben. Auf eine weitere Darstellung wird an dieser Stelle aus Gründen des Umfangs verzichtet. Im mittleren Fall erreicht Quicksort ebenfalls eine Laufzeit von $\Theta(n \cdot \log_2(n))$.

3.2.4 Praxisrelevanz

Es wird deutlich, dass die Wahl des Pivotelements entscheidenden Einfluss auf die Laufzeit hat. Die Wahrscheinlichkeit des Eintreffens des Worst-Case ist bei verschiedenen Ansätzen zur Bestimmung günstiger Pivotelemente unterschiedlich groß (Aufzählung nach [Wiki QS]):

- **Naiver Ansatz:** Als Pivotelement wird immer das erste, letzte oder mittlere Element der Liste gewählt. Dieser naive Ansatz ist aber relativ ineffizient.
- **Median-Ansatz:** Eine andere Möglichkeit ist es den Median dieser drei Elemente zu bestimmen und als Pivotelement zu verwenden.
- **Randomisierter Quicksort:** Ein anderer Ansatz ist, als Pivotelement ein zufälliges Element auszuwählen. Bei diesem randomisierten Quicksort ist die Wahrscheinlichkeit, dass das Pivotelement in jedem Teilungsschritt so gewählt wird, dass sich die Worst-Case-Laufzeit ergibt, extrem gering. Man kann davon ausgehen, dass er praktisch nie auftritt.
- **Quicksort mit dem Median-of-medians-Algorithmus:** Verwendet man für die Wahl des Pivotelements den Median-of-medians-Algorithmus, welcher den Median eines Arrays in $\mathcal{O}(n)$ bestimmt, so kann insgesamt eine Laufzeit von $\mathcal{O}(n \cdot \log_2(n))$ für den schlechtesten Fall von Quicksort garantiert werden (vgl. auch [Lang 2018, Kap. „Analyse“]).

3.2.5 Untere Schranke für vergleichsbasierte Sortierverfahren

Quicksort einer der schnellsten Sortieralgorithmen, die es gibt (und geben kann!). Eine Erklärung dieser Tatsache wird in [Wiki SortVerf, Kap. „Beweis der unteren Schranke für vergleichsbasiertes Sortieren“] gegeben, aber hier nicht weiter ausgeführt. Trotzdem soll sie in folgendem Theorem gewürdigt werden:

Es ist für ein vergleichsbasiertes Sortierverfahren nicht möglich, n Zahlen schneller als in einer Zeit von $\mathcal{O}(n \cdot \log_2(n))$ zu sortieren.
 $\Omega(n \cdot \log_2(n))$ ist also die untere Laufzeitgrenze für vergleichsbasiertes Sortieren von n Zahlen.

3.3 Speicherplatz

Das Verwenden der Rekursion kostet Speicherplatz. In der Praxis wird es daher häufig so gehandhabt, dass die Rekursion des Quicksort-Algorithmus nur bis zu einer gewissen Tiefe angewendet und die restlichen Teillisten dann iterativ sortiert werden. Es gibt Varianten von Quicksort, die eine Rekursionstiefe von maximal $\log_2(n)$ garantieren (vgl. bspw. [Wiki QS]).

4 Implementation

Im Folgenden wird der QuickSort-Algorithmus mithilfe der NRW-Landesklasse `List` implementiert. Dafür werden die in Tabelle 2 gegebenen Städte des Kreises Borken betrachtet, die nach ihrer Zahl der Einwohner sortiert werden sollen. Es wird also ein »Städteranking« gemessen an den Einwohnerzahlen erstellt.

Name	Einwohnerzahl
Ahaus	39185
Bocholt	71036
Borken	42509
Gescher	17253
Gronau	47671
Isselburg	10713
Rhede	19165
Stadtlohn	20367
Velen	12989
Vreden	22561

Tabelle 2: Städte des Kreises Borken.

4.1 Die generische Landeskasse `List`

„Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden. Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.“ [LPN IF Doku GK]

Die genauen Methodenbezeichnungen – mitsamt Erläuterungen – können im Anhang nachgeschlagen werden (Anhang S. 38). Die Landeskasse `List` kann auf der Materialseite für Informatik („Lehrplannavigator“) unter [LPN IF, Abs. „Grundlegende Materialien und Dokumentationen für den Unterricht und für das Zentralabitur“] heruntergeladen werden.

den werden (ebenfalls sind dort die Landesklassen für die anderen Datenstrukturen veröffentlicht).

4.2 Andere Klassen des Projekts

Abbildung 6 zeigt das BlueJ-Projekt und die übrigen Klassen, deren Funktionen im Folgenden erläutert werden.

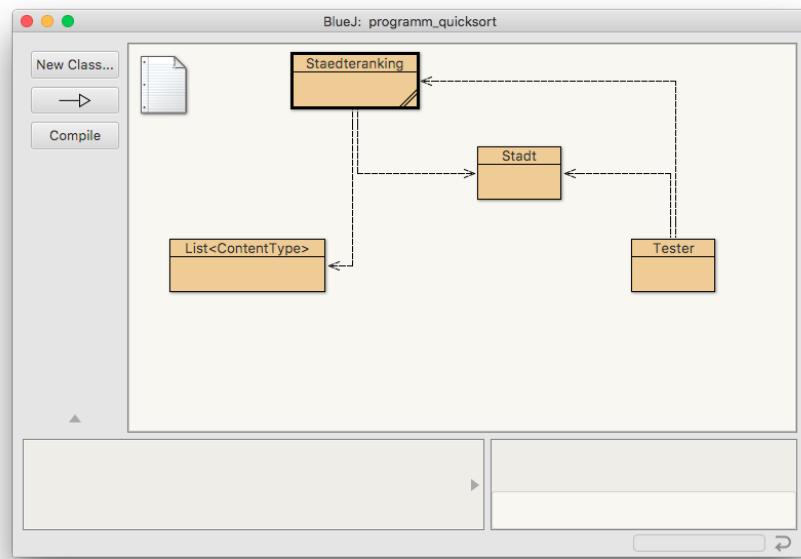


Abbildung 6: Der Screenshot zeigt die verwendeten Klassen und ihre Beziehungen in BlueJ.

Erläuterungen:

- **Die Klasse Stadt:** Objekte dieser Klasse werden von der Klasse **Staedteranking** in einer Liste verwaltet. Objekte dieser Klasse werden verwendet, um die Städte aus Tabelle 2 zu modellieren. Eine Stadt zeichnet sich durch einen Namen und eine Einwohnerzahl aus, entsprechende Getter-Methoden ermöglichen den Zugriff auf die Attribute.
- **Die Klasse Staedteranking:** In dieser Klasse werden die Städte in einer als Attribut angelegten Liste von Namen `rankingliste` verwaltet, welche über den Konstruktor mit Städten (also Objekten der Klasse `Stadt`) gefüllt wird. Ferner gibt es die Möglichkeit, mit der Methode `neueStadtInsRankingAufnehmen(Stadt`

`pStadt`) eine neue Stadt der Liste hinzuzufügen. Die Klasse stellt außerdem auch die Methoden zum Sortieren und Ausgeben der Liste bereit.

- **Die Klasse Tester:** Diese Klasse dient zum Testen der Programmierung (ihre Klassendiagramm ist in Abb. 7 nicht aufgeführt). Beim Instanziieren der Klasse wird eine Test-Methode direkt ausgeführt, welche das Städteranking (also die sortierte Liste der Städte) ausgibt.
- **Die Klasse List<ContentType>:** Dies ist die bekannte NRW-Landesklasse. Eine Dokumentation der Klasse findet sich im Anhang (S. 38).

4.3 Klassendiagramm

Das Klassendiagramm in Abbildung 7 zeigt die verwendeten Klassen. Zu beachten ist insbesondere die über das Attribut `rankingliste` verwaltete Liste an Städten in der Klasse `Staedteranking`. Die Klasse `Tester` ist nicht aufgeführt.

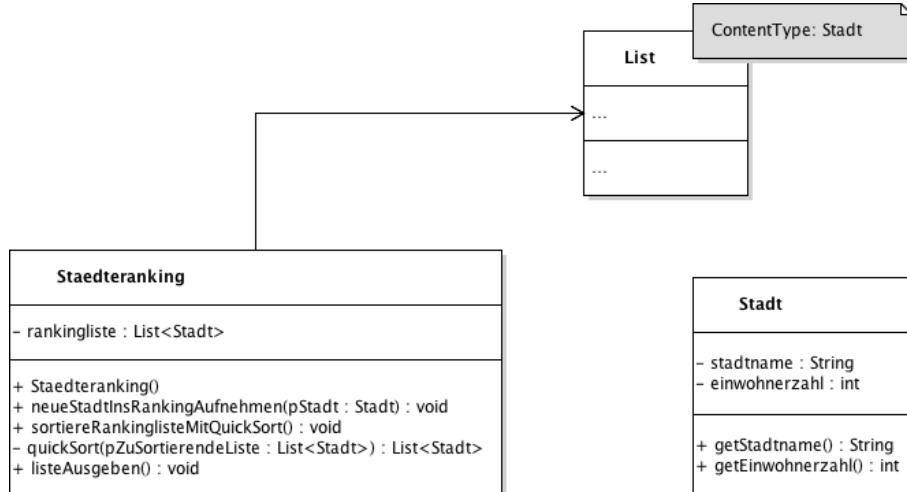


Abbildung 7: Das Klassendiagramm zum BlueJ-Projekts.

4.4 Implementation des Quicksort-Algorithmus

Im Folgenden wird der Quicksort-Algorithmus unter Verwendung der Landesklasse `List` implementiert. Zu beachten ist dabei, dass es dafür zwei Methoden in der Klasse `Staedteranking` gibt. Die private Methode `private List<Stadt> quickSort(List<Stadt> pZuSortierendeListe)` sortiert dabei die übergebene Liste `pZuSortierendeListe`

(diese Methode wird im Folgenden genauer analysiert). Die Methode `public void sortiereRankinglisteMitQuickSort()` ruft diese Methode nur auf und veranlasst damit das Sortieren der Rankingliste.

Als Pivotelement wird immer das letzte Element einer Liste gewählt.

4.4.1 Pseudocode

Folgender Pseudocode liegt der Methode `private List<Stadt> quickSort(List<Stadt> pZuSortierendeListe)` zugrunde (Algorithmus 1). Die sortierte Liste wird am Ende zurückgegeben.

Input: Liste an Daten (Eingabeliste)

Output: Sortierte Liste (Rückgabe der sortierten Liste)

```
1 if Eingabeliste nicht leer then
2   Deklariere und initialisiere eine leere linke Liste
3   Deklariere und initialisiere eine leere rechte Liste
4   Deklariere und initialisiere eine leere sortierte Liste
5   Speichere das letzte Element der Eingabeliste als Pivotelement zwischen
6   Entferne das letzte Element aus der Eingabeliste
7   Springe an den Anfang der Eingabeliste
8   while Eingabeliste nicht leer do
9     if Aktuelles Element der Eingabeliste kleiner als Pivotelement then
10       Füge das aktuelle Element der Eingabeliste in die linke Liste ein
11       Entferne das aktuelle Element aus der Eingabeliste
12     else
13       Füge das aktuelle Element der Eingabeliste in die rechte Liste ein
14       Entferne das aktuelle Element aus der Eingabeliste
15     end
16     Springe ein Element in der Eingabeliste weiter
17   end
18   Hänge an die sortierte Liste die rekursiv sortierte linke Liste an
19   Hänge an die sortierte Liste das Pivotelement an
20   Hänge an die sortierte Liste die rekursiv sortierte rechte Liste an
21   Gib die sortierte Liste zurück
22 else
23   |  Gib nichts zurück
24 end
```

Algorithmus 1: Quicksort auf Listen (allgemeiner Pseudocode).

Es lassen sich also folgende Schritte aus dem Pseudocode ableiten:

1. Es wird überprüft, ob die zu sortierende Liste überhaupt Elemente aufweist (falls ja, wird sortiert, falls nein, wird `null` zurückgegeben).
2. Es werden drei Listen deklariert (`linkeListe`, `rechteListe`, `gebauteListe`).
3. Es werden die drei Listen als leere Listen instanziert.
4. Es wird an das Ende der zu sortierenden Liste gesprungen.
5. Das Pivotelement wird bestimmt, indem die Stadt in der lokalen Variable `Stadt pivot` abgespeichert wird.
6. Das Pivotelement wird aus der zu sortierenden Liste entfernt.
7. Es wird an den Anfang der zu sortierenden Liste gesprungen.
8. Die zu sortierende Liste wird komplett durchlaufen und die Elemente der Liste – ausgehend von den Einwohnerzahlen der „Pivotstadt“ – in die linke bzw. rechte Liste kopiert.
9. An die (noch leere) gebaute Liste wird die rekursiv sortierte linke Liste angehängt.
10. An die gebaute Liste wird die „Pivotstadt“ angehängt.
11. An die gebaute Liste wird die rekursiv sortierte rechte Liste angehängt.
12. Die vollständig sortierte Liste wird zurückgegeben.

4.4.2 Java-Quelltext

Die öffentliche `public void sortiereRankinglisteMitQuickSort()` wird zum Sortieren der Rankingliste aufgerufen und ist selbsterklärend (vgl. Quelltext 1); die Referenz `rankingliste` wird auf die zurückgegebene Liste gesetzt. Damit wird die unsortierte Rankingliste von der Garbage Collection entfernt, sodass nur noch die sortierte Liste unter der Referenz zu erreichen ist.

```

1   public void sortiereRankinglisteMitQuickSort(){
2       rankingliste = quickSort(rankingliste);
3   }

```

Quelltext 1: Öffentliche Methode zum Sortieren der Rankingliste. Der Aufruf, die Rankingliste zu sortieren, wird direkt an die eigentliche Sortiermethode wird direkt aufgerufen (vgl. Quelltext 2). Die Trennung ist notwendig, da die Sortiermethode rekursiv arbeitet.

Die Methode `private List<Stadt> quickSort(List<Stadt> pZuSortierendeListe)` (Quelltext 2) setzt den Pseudocode aus Kapitel 4.4.1 um. Sie ist privat, da sie nur von der Methode aus Quelltext 1 aufgerufen wird. Außerdem ist zu beachten, dass diese Methode rekursiv arbeitet (und am Ende die sortierte Liste zurückgibt); eine Trennung des Sortierproblems in zwei Methoden ist also notwendig, da die Methode aus Quelltext 1 nichts zurückgibt.

Zu beachten ist, wie die drei Listen deklariert und initialisiert werden und die übergebene Liste mit der `hasAccess()`-Methode durchlaufen wird. Dieses Vorgehen ist aus dem Unterricht bekannt.

Für den Vergleich einer Stadt mit dem Pivotelement (der Pivotstadt) ist darauf zu achten, dass unbedingt `pZuSortierendeListe.getContent().getEinwohnerzahl()` aufgerufen wird, um von der Liste auf die aktuelle Stadt und von der aktuellen Stadt auf die Einwohner zugreifen zu können.

Zum Zusammenfügen der linken und rechten Teillisten wird am Ende die Methode `concat(...)` der Liste verwendet, in deren Parametern auch die rekursiven Aufrufe stattfinden. Um das Pivotelement in der „Mitte“ einzufügen, muss die Methode `append(...)` verwendet werden, um die Stadt (also das Objekt der Klasse `Stadt`) an den ersten Teil der sortierten Liste hinten anzuhängen.

```

1  private List<Stadt> quickSort(List<Stadt> pZuSortierendeListe){
2      // Solange die zu sortierende Liste nicht leer ist,
3      // muss sortiert werden...
4      if(!pZuSortierendeListe.isEmpty()){
5          // Lege linke und rechte Listen an
6          List<Stadt> linkeListe;
7          List<Stadt> rechteListe;
8
9          // Lege eine Liste an, die am Ende nach oben weiter gegeben wird
10         List<Stadt> gebauteListe;
11
12         // Initialisiere die drei Listen (zu Beginn leer)
13         linkeListe = new List<Stadt>();

```

```

14     rechteListe = new List<Stadt>();
15     gebauteListe = new List<Stadt>();
16
17     // Waehle als Pivoelement das letzte Element:
18     // Springe zum Ende
19     pZuSortierendeListe.toLast();
20
21     // Speichere das Element ab
22     Stadt pivot = pZuSortierendeListe.getContent();
23
24     // Entferne das Element, damit es beim Aufteilen in die linke
25     // und rechte Liste nicht beruecksichtigt wird
26     pZuSortierendeListe.remove();
27
28     // Springe an den Anfang
29     pZuSortierendeListe.toFirst();
30
31     // Los geht's mit dem Aufteilen
32     while(pZuSortierendeListe.hasAccess()){
33
34         // Entscheide anhand der Einwohnerzahlen, welche Stadt nach
35         // links bzw. nach rechts muss (die zu sortierende Liste
36         // wird dabei von links nach rechts durchlaufen)
37         if(pZuSortierendeListe.getContent().getEinwohnerzahl() <
38             pivot.getEinwohnerzahl()){
39             linkeListe.append(pZuSortierendeListe.getContent());
40         }
41         else {
42             rechteListe.append(pZuSortierendeListe.getContent());
43         }
44
45         // Gehe ein Element in der zu sortierenden Liste weiter
46         pZuSortierendeListe.next();
47     }
48
49     // Setze die fertige (gebaute) Liste zusammen,
50     // allerdings wird vorher quickSort(...) noch auf die
51     // linke und rechte Liste aufgerufen
52
53     // ****
54     // * REKURSIVE AUFRUFE *
55     // ****
56
57     // Zuerst kommt die sortierte (!) linke Liste, dann...
      gebauteListe.concat(quickSort(linkeListe));

```

```

58
59         // ...das aktuelle Pivotelement nicht vergessen...
60         gebauteListe.append(pivot);
61
62         // ...und die sortierte (!) rechte Liste.
63         gebauteListe.concat(quickSort(rechteListe));
64
65         // Die sortierte Liste kann zurueck gegeben werden
66         return gebauteListe;
67     }
68     else {
69         // Ist die Liste leer, wird null zurueck gegeben.
70         // Dies ist der Rekursionsanker
71         return null;
72     }
73
74 }
```

Quelltext 2: Private Methode zum Sortieren einer übergebenen Liste.

4.4.3 Ausgabe

Das Programm gibt die nach Einwohnerzahl sortierte Liste der Städte aus Tabelle 2 in der Konsole aus (Quelltext 3), wenn die Klasse **Tester** instanziert wird. Ein Screenshot ist in Abbildung 8 gegeben (S. 40).

```

1 Rankingliste:
2 ****
3 Isselburg: 10713
4 Velen: 12989
5 Gescher: 17253
6 Rhede: 19165
7 Stadtlohn: 20367
8 Vreden: 22561
9 Ahaus: 39185
10 Borken: 42509
11 Gronau: 47671
12 Bocholt: 71036
```

Quelltext 3: Ausgabe in der Konsole. Die Städte liegen nach Einwohnerzahlen sortiert vor – von Isselburg (Z. 3) bis Bocholt (Z. 12).

5 Fazit

In dieser Facharbeit wurde gezeigt, dass Quicksort ein sehr schneller und leicht zu implementierender Sortieralgorithmus ist. Mit einer Laufzeit von $\mathcal{O}(n \log_2(n))$ ist er sogar eines der schnellstmöglichen vergleichsbasierten Sortieralgorithmen. Im schlechtesten Fall ist die Laufzeit allerdings quadratisch und damit genauso langsam wie z. B. Selectionsort. Da beim randomisierten Quicksort der schlechteste Fall allerdings äußerst unwahrscheinlich ist und in der Praxis eine Datenmenge i. d. R. nicht *komplett* mit Quicksort sortiert wird, kann dieser Nachteil gegenüber anderen Sortieralgorithmen vernachlässigt werden.

5.1 Blick in die Praxis

Quicksort existiert in vielen Varianten, die sich in der Wahl des Pivotelements unterscheiden:

Es ist möglich, mit einer Variante von Quicksort auch im schlechtesten Fall eine Zeitkomplexität von $\mathcal{O}(n \log_2(n))$ zu erreichen (indem als Vergleichselement der Median gewählt wird). Dieses Verfahren ist jedoch im Durchschnitt und im schlechtesten Fall um einen konstanten Faktor langsamer als Heapsort oder Mergesort [vgl. Kap. 5.2]; daher ist es für die Praxis nicht interessant.

[Lang 2018]

5.2 Ausblick auf andere Algorithmen

Mergesort Ein anderer Sortieralgorithmus, der ebenfalls nach dem Teile-und-Herrsche-Prinzip funktioniert, ist der Mergesort-Algorithmus (vgl. [Wiki SortVerf]). Bei diesem Verfahren wird beim Zusammensetzen der Teillösungen („combine“) die Sortierung erzielt (und nicht durch das geschickte Aufteilen in die Teilprobleme wie beim Quicksort-Algorithmus).

Heapsort Heapsort ist mit einer Laufzeit von $\mathcal{O}(n \log_2(n))$ genau wie Quicksort ebenfalls optimal (vgl. [Wiki SortVerf]). Dieses Sortierverfahren verwendet eine besondere Datenstruktur, die als »Heap« bezeichnet wird. Ein Heap basiert auf der Datenstruktur »Baum«. Auf weitere Ausführungen wird an dieser Stelle verzichtet.

5.3 Schwierigkeiten beim Bearbeiten der Facharbeit

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

6 Weitere Informationen

Kapitel 6.1 gibt ein paar Hinweise, wie die Recherche für die Facharbeit anfangen kann. Es ist sinnvoll, zunächst das Inhaltsverzeichnis der Facharbeit sehr genau auszuarbeiten, bevor mit der „Schreibarbeit“ begonnen wird. Vielleicht ist es auch sinnvoll, Texte erst stichpunktartig vorzuschreiben, um den Umfang und genauen Inhalt eines Kapitels besser abschätzen zu können. Für die Struktur des Inhaltsverzeichnisses kann auch ein Blick in die ausgezeichneten Arbeiten des Hans-Riegel-Fachpreises helfen (Kap. 6.5).

In Kapitel 6.2 werden Informationen zu L^AT_EX gegeben. Nützliche Programme für Informatik und Mathematik werden in den Kapiteln 6.3 und 6.4 vorgestellt.

Sehr gute Facharbeiten können auch am Hans-Riegel-Fachpreis teilnehmen. Mehr Informationen werden in Kapitel 6.5 gegeben.

6.1 Quellenrecherche für die Facharbeit

Ansätze für die Literaturrecherche könnten sein:

- Wikipedia-Artikel
- Quellen des Wikipedia-Artikels (unten auf der Seite)
- Suchen Sie nach Präsentationen (als PDFs) aus dem universitären Kontext. Am Ende der Präsentationen finden sich häufig Literaturverweise.
- Einige Webseiten bzw. Verlage bieten eine „Blick ins Buch“-Funktion, wenn ein Buch gefunden ist. So kann man sich einen Überblick über die Inhalte des Buches verschaffen.
- Einige Bücher sind vielleicht auch auf Google Books zugänglich. <https://books.google.com/>
- Bibliotheken (auch die Schulbibliothek)

6.2 Weitere Informationen zu Latex

Folgende Informationen könnten interessant sein:

- **Installation:** Neben dem L^AT_EX- bzw. T_EX-System wird ein Editor benötigt, in welchem die `tex`-Datei bearbeitet und zum PDF-Dokument übersetzt wird. Online finden sich viele Installationsanleitungen für verschiedene Betriebssysteme.

Empfehlenswert sind:

– **Windows:**

- * T_EX-System: »MikTeX«, <https://miktex.org/> (oder »TexLive«, [http s://www.tug.org/texlive/](https://www.tug.org/texlive/))
- * Editor: »TexMaker«, <https://www.xm1math.net/texmaker/>

– **macOS:**

- * T_EX-System: »MacTeX«, <https://www.tug.org/mactex/>
- * Editor: »TeXShop«, <https://pages.uoregon.edu/koch/texshop/>

- Übersicht und Vergleich verschiedener Editoren (für verschiedene Betriebssysteme): https://en.m.wikipedia.org/wiki/Comparison_of_TeX_editors
- Es gibt auch Online-Latex-Editoren (dann ist keine Installation notwendig), bspw. »Overleaf«, <https://www.overleaf.com/>

- **Latex-Einstieg, Erklärungen:** »learnlatex.org«. Lesen Sie unbedingt die ersten Kapitel zum Einstieg, wenn Sie das erste Mal mit Latex arbeiten.
- **Latex-Befehl für ein Zeichen vergessen?:** »Detexify«. Man kann das gesuchte Zeichen mit der Maus malen und die Webseite gibt einen passenden Latex-Befehl aus. <http://detexify.kirelabs.org/classify.html>
- **Mathematischer Textsatz:** »Wikipedia-Hilfe«. Setzen von Integralen, Summen, Reihen, Matrizen etc. <https://de.wikipedia.org/wiki/Hilfe:TeX>
- **Wissenschaftliche Arbeiten als Vorlage:** Im Internet finden sich viele Latex-Einstiegsbeispiele und Vorlagen für Bachelor-, Master- und Diplomarbeiten. Es empfiehlt sich, gezielt nach PDF-Dokumenten zu suchen.
- **Grafiken:** »LatexDraw«. Grafiken selbst gestalten und Latex-kompatibel exportieren (für die PSPicture-Umgebung von Latex). <http://latexdraw.sourceforge.net/>

- **Grafiken, Graphen:** Möchte man Grafiken selbst erstellen, empfiehlt sich das Latex-Paket »TikZ«. TikZ braucht aber etwas Eingewöhnung – andere Programme sind da häufig einfacher (bspw. LatexDraw, s. o.). Eine kleine, erste Übersicht findet man unter https://www.overleaf.com/learn/latex/TikZ_package und <https://en.wikibooks.org/wiki/LaTeX/PGF/TikZ>.
- **Bilder und Grafiken:** Bilder und Grafiken sollten wenn möglich als PDF exportiert werden (Vektorgrafik). Wenn dies nicht möglich ist, sind `eps`-Dateien den üblichen Bild-Dateien (`png`, `jpg`) vorzuziehen, da dies ebenfalls Vektorgrafiken sind.
- **Dokumentation von Paketen:** »CTAN«. Zu jedem Paket, welches in der Präambel des Dokuments geladen wurde, findet sich auf CTAN eine Dokumentation. Diese wird aber häufig sehr schnell sehr umfangreich und detailreich, kann aber für sehr spezielle Fragen einen Blick wert sein. <http://www.ctan.org/tex-archive/macros/latex/contrib/>
- **Hilfeforum:** »Tex.Stackexchange«, <https://tex.stackexchange.com/>
- **Verweise auflösen:** Vergessen Sie nicht, Ihr Dokument mehrfach zu übersetzen, um alle Verweise aufzulösen (damit alle »??« aufgelöst werden). Das Paket `refcheck` (s. Präambel dieses Dokuments) gibt Hinweise auf kaputte Verweise.

6.3 Programme für Informatik

Einige Dienste setzen Accounts voraus, andere sind ohne Account nutzbar. Zum Teil sind Grafiken etc. nicht ohne Account zu speichern oder zu exportieren. In jedem Fall sollte man erst ein bisschen ausprobieren und verschiedene Dienste miteinander vergleichen (immer vor dem Hintergrund, später noch Änderungen vorzunehmen!), bevor man viel Zeit und Mühe in Grafiken investiert.

- **Zeichnungen, Grafiken, Infografiken:** »Draw.io«. Sehr umfangreich und mächtig – es gibt viele Vorlagen (z. T. auch UML-konform). <https://www.draw.io>
- **Zeichnungen, Sequenzdiagramme, Graphen:** »HackMD«. Sehr umfangreich und nach kurzer Einarbeitung einfach zu bedienen. <https://hackmd.io>, Feature-Übersicht: <https://hackmd.io/features?both>

- **ER-Diagramme:** »ERDplus«, <https://erdplus.com/>
- **Graphen, Automaten, Formale Sprachen:** »FLACI«. Kann auch DEAs, NEAs und Kellerautomaten simulieren. <https://flaci.com/home/>
- **Struktogramme:** »Struktogrammeditor«, <https://whiledo.de/index.php?p=struktogrammeditor>
- **Klassendiagramme, Objektdiagramme, UML-Diagramme:** »Violet UML«. Abbildung 7 wurde mit Violet UML erstellt. <http://alexdp.free.fr/violetumleditor/page.php>
- **UML-Diagramme:** »Dia«. Professioneller als Violet UML (s.o.). <http://diainstaller.de/>
- **UML-Diagramme, ER-Diagramme:** »yEd«. Professioneller, aber auch umständlicher als Violet UML. <https://www.yworks.com/products/yed>.
Dieses Programm wird auch genutzt, um die Klassen-, Objekt- und ER-Diagramme für die Abiturprüfungen zu erstellen. Im Lehrplannavigator Informatik gibt es auch die NRW-spezifische Symbole und NRW-Beispiel-Diagramme für yED zum Download, vgl. „Weiterführende Quellen“ unter <https://www.schulentwicklung.nrw.de/lehrplaene/lehrplannavigator-s-ii/gymnasiale-oberstufe/informatik/hinweise-und-beispiele/index.html>
- **Sequenzdiagramme:** »js-sequence-diagrams«, <https://bramp.github.io/js-sequence-diagrams/> und »WebSequenceDiagrams«, <https://www.websequencediagrams.com/>
- **Java-Nachschatzwerk:** »Java ist auch eine Insel« von C. ULLENBOOM, <http://openbook.rheinwerk-verlag.de/javainsel/>
- **Visualisierungen von Algorithmen:** »Visualgo«. Nachschlagewerk und sehr gute Visualisierung. Hat auch die Möglichkeit, eigene Beispiele auszuprobieren. <https://visualgo.net/de>
- **Landesklassen:** Lehrplannavigator Informatik (Kapitel „Grundlegende Materialien und Dokumentationen für den Unterricht und für das Zentralabitur“), <https://>

//www.schulentwicklung.nrw.de/lehrplaene/lehrplannavigator-s-ii/gymnasiale-oberstufe/informatik/hinweise-und-beispiele/index.html.

Dort ist ebenfalls die Dokumentation der Landesklassen verlinkt (GK- und LK-Variante). GK: https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/MaterialZABI/2017-11-28_Dokumentation_GK_ab_Abitur_2018.pdf, LK: https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/MaterialZABI/2017-11-28_Dokumentation_LK_ab_Abitur_2018.pdf

- **Java-Entwicklungsumgebung für die Schule:** »BlueJ«, <https://www.bluej.org/>
- **Umfangreiche Java-Entwicklungsumgebung:** »NetBeans«. Diese IDE halte ich am ehesten für Einsteiger geeignet. <https://netbeans.org/>
- **Profi-Java-Entwicklungsumgebung:** »Eclipse«, <https://www.eclipse.org/>
- **Profi-Java-Entwicklungsumgebung:** »IntelliJ IDEA«, <https://www.jetbrains.com/idea/>

6.4 Programme für Mathematik

Folgende Programme und Dienste können für Facharbeiten im Fach Mathematik interessant sein.

- **Allesköninger, CAS:** »Wolfram Alpha«. Gewissermaßen eine Suchmaschine für Lösungen zu mathematischen Problemen. Ergebnisse lassen sich als Latex-Formelcode kopieren. Kauft man die App, werden auch Lösungswege angezeigt. <https://www.wolframalpha.com/>
- **Mini-CAS:** »Eigenmath«. Rechnet symbolisch (und nicht algebraisch!) und programmierbar. <http://www.eigenmath.org/>
- **Graphen, Visualisierungen, CAS:** »GeoGebra«. Graphen lassen sich Latex-kompatibel exportieren (für PSPicture oder TikZ). Alternativ empfiehlt sich der Export als PDF. <https://www.geogebra.org/>

- **Formeln von Wikipedia:** Möchte man Formeln von Wikipedia kopieren (und nicht abschreiben), kann man sich über die Funktion „Quelltext bearbeiten“ eines Abschnitts bzw. Artikels den Quelltext (und damit auch die Formel in Latex-Notation!) anzeigen lassen und kopieren.

6.5 Hans-Riegel-Fachpreis

Gute Facharbeiten aus dem MINT-Bereich können bei der Hans-Riegel-Stiftung eingereicht werden (die Arbeiten sollten über die Schule eingereicht werden). Informationen (und auch ausgezeichnete Arbeiten) können auf der Webseite des Dr.-Hans-Riegel-Fachpreises eingesehen werden. Es gibt Geldpreise zu gewinnen. Der **Einsendeschluss** ist zu beachten (i. d. R. 01. Mai eines jeden Jahres).

Die Preisverleihung findet für Schulen unseres Kreises traditionell im Schloss Münster statt. Die Anzahl der eingereichten Arbeiten war in den letzten Jahren vergleichsweise gering, sodass die Chancen auf einen Gewinn insgesamt gut sind (eine entsprechende Facharbeit natürlich vorausgesetzt).

Informationen zum Wettbewerb:

<https://www.hans-riegel-fachpreise.com/wettbewerb/>

Ausgezeichnete Facharbeiten:

<https://www.hans-riegel-fachpreise.com/ausgezeichnete-arbeiten/>

Kooperation WWU Münster:

<https://www.hans-riegel-fachpreise.com/universitaeten/westfaelische-wilhelms-universitaet-muenster/>

Literaturverzeichnis

[Hoare 1962] C. A. R. HOARE (1962): *Quicksort*. Computer Journal, Vol. 5, 1, S. 10–15.

[IDEA] S. P. FEKETE, S. MORR (2018): *KVICK SÖRT*. <https://idea-instruction.s.com/quick-sort/>, aufgerufen am 02.04.2020.

[Kempe Löhr 2015] T. KEMPE, A. LÖHR (Hrsg.) (2015): *Informatik – Lehrwerk für die gymnasiale Oberstufe – Neubearbeitung. Schülerband 2*. Schöningh-Verlag. Paderborn.

[Khan 2020] Khan Academy (2020): *Analyse von Quicksort*. <https://de.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort>, aufgerufen am 08.04.2020.

[Lang 2018] H. W. LANG (2018): *Quicksort*. <https://www.inf.hs-flensburg.de/lang/algorithmen/sortieren/quick/quick.htm>, aufgerufen am 07.04.2020.

[Lang DuC] H. W. LANG (unbekanntes Jahr): *Divide-and-Conquer-Prinzip*. <https://www.inf.hs-flensburg.de/lang/glossar/divconq.htm>, aufgerufen am 07.04.2020.

[LPN IF Doku GK] Qualitäts- und UnterstützungsAgentur – Landesinstitut für Schule (2020): *Dokumentation für den Grundkurs*. https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/MaterialZABI/2017-11-28_Dokumentation_GK_ab_Abitur_2018.pdf, aufgerufen am 07.04.2020.

[LPN IF] Qualitäts- und UnterstützungsAgentur – Landesinstitut für Schule (2020): *Hinweise und Beispiele zur standardorientierten Unterrichtsentwicklung im Fach Informatik*. <https://www.schulentwicklung.nrw.de/lehrplaene/lehrplannavigator-s-ii/gymnasiale-oberstufe/informatik/hinweise-und-beispiele/hinweise-und-beispiele.html>, aufgerufen am 07.04.2020.

[VisuAlgo] Dr. S. HALIM, Dr. F. HALIM (2020): *VisuAlgo.net*. <https://visualgo.net/de/sorting>, aufgerufen am 04.04.2020.

[Wiki Hoare] Wikipedia (2019): *Tony Hoare*. https://de.wikipedia.org/wiki/Tony_Hoare, Stand 20.03.2019, aufgerufen am 07.04.2020.

[Wiki QS] Wikipedia (2020): *Quicksort*. <https://de.wikipedia.org/wiki/Quicksort>, Stand 17.03.2020, aufgerufen am 08.04.2020.

[Wiki SortVerf] Wikipedia (2020): *Sortierverfahren*. <https://de.wikipedia.org/wiki/Sortierverfahren>, Stand 07.02.2020, aufgerufen am 07.04.2020.

[Wiki Stab] Wikipedia (2019): *Stabilität (Sortierverfahren)*. [https://de.wikipedia.org/wiki/Stabilit%C3%A4t_\(Sortierverfahren\)](https://de.wikipedia.org/wiki/Stabilit%C3%A4t_(Sortierverfahren)), Stand 12.10.2019, aufgerufen am 06.04.2020.

Abbildungsverzeichnis

1	Unsortierte Daten.	4
2	Sortierte Daten.	5
3	Bildliche Darstellung des Quicksort-Algorithmus.	9
4	Darstellung der Rekursion im Best-Case.	10
5	Darstellung der Rekursion im Worst-Case.	11
6	Screenshot von BlueJ.	15
7	Klassendiagramm.	16
8	Ausgabe der sortierten Rankingliste.	40

Tabellenverzeichnis

1	Schritte der Divide-and-Conquer-Strategie.	6
2	Städte des Städterankings.	14

Algorithmenverzeichnis

1	Quicksort-Pseudocode (für die Datenstruktur Liste).	18
---	---	----

Quelltextverzeichnis

1	Öffentliche Methode zum Sortieren der Rankingliste.	20
2	Private Methode zum Sortieren einer übergebenen Liste.	20
3	Ausgabe des Programms in der Konsole.	22

Anhang

Der Anhang gliedert sich in:

- a) Dokumentation der Landesklasse **List** (S. 38)
- b) Hinweise zum Starten des Programms (S. 40, [LPN IF Doku GK])

Die generische Klasse `List<ContentType>`

Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden. Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

Konstruktor `List()`

Eine leere Liste wird erzeugt.

Anfrage `boolean isEmpty()`

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

Anfrage `boolean hasAccess()`

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

Auftrag `void next()`

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d.h. `hasAccess()` liefert den Wert `false`.

Auftrag `void toFirst()`

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag `void toLast()`

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Anfrage `ContentType getContent()`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.

Auftrag	void setContent(ContentType pContent) Falls es ein aktuelles Objekt gibt (<code>hasAccess() == true</code>) und <code>pContent</code> ungleich <code>null</code> ist, wird das aktuelle Objekt durch <code>pContent</code> ersetzt. Sonst bleibt die Liste unverändert.
Auftrag	void append(ContentType pContent) Ein neues Objekt <code>pContent</code> wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt <code>pContent</code> in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (<code>hasAccess() == false</code>). Falls <code>pContent</code> gleich <code>null</code> ist, bleibt die Liste unverändert.
Auftrag	void insert(ContentType pContent) Falls es ein aktuelles Objekt gibt (<code>hasAccess() == true</code>), wird ein neues Objekt <code>pContent</code> vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (<code>hasAccess() == false</code>), wird <code>pContent</code> in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (<code>hasAccess() == false</code>) und die Liste nicht leer ist oder <code>pContent == null</code> ist, bleibt die Liste unverändert.
Auftrag	void concat(List<ContentType> pList) Die Liste <code>pList</code> wird an die Liste angehängt. Anschließend wird <code>pList</code> eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und <code>pList</code> um dasselbe Objekt handelt, <code>pList == null</code> oder eine leere Liste ist, bleibt die Liste unverändert.
Auftrag	void remove() Falls es ein aktuelles Objekt gibt (<code>hasAccess() == true</code>), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (<code>hasAccess() == false</code>). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (<code>hasAccess() == false</code>), bleibt die Liste unverändert.

Hinweise zum Starten des Programms

Das Projekt wird wie gewohnt über die package.bluej geöffnet. Wenn die Klasse **Tester** instanziiert wird, wird über den Konstruktor die Rankingliste in der Klasse **Staedteranking** mit den Städten aus Tabelle 2 befüllt. Anschließend wird direkt die Testmethode `public void testeDenQuickSortAlgorithmus()` ausgeführt. Es wird die Ausgabe wie in Abbildung 8 erzeugt.



The screenshot shows a terminal window titled "BlueJ: Terminal Window - programm_quicksort". It displays two sorted lists of cities based on population. The first list, labeled "Rankingliste:", shows cities from highest to lowest population: Ahaus (39185), Bocholt (71036), Borken (42509), Gescher (17253), Gronau (47671), Isselburg (10713), Rhede (19165), Stadtlohn (20367), Velen (12989), and Vreden (22561). The second list, also labeled "Rankingliste:", shows the same cities from lowest to highest population: Isselburg (10713), Velen (12989), Gescher (17253), Rhede (19165), Stadtlohn (20367), Vreden (22561), Ahaus (39185), Borken (42509), Gronau (47671), and Bocholt (71036). A status bar at the bottom of the terminal window reads "Can only enter input while your".

```
Rankingliste:  
*****  
Ahaus: 39185  
Bocholt: 71036  
Borken: 42509  
Gescher: 17253  
Gronau: 47671  
Isselburg: 10713  
Rhede: 19165  
Stadtlohn: 20367  
Velen: 12989  
Vreden: 22561  
  
Rankingliste:  
*****  
Isselburg: 10713  
Velen: 12989  
Gescher: 17253  
Rhede: 19165  
Stadtlohn: 20367  
Vreden: 22561  
Ahaus: 39185  
Borken: 42509  
Gronau: 47671  
Bocholt: 71036
```

Abbildung 8: Ausgabe des Projekts beim Instanziieren der Testklasse. Die anfänglich nach Namen sortierten Städte (oben) wurden nach Einwohnerzahlen sortiert (unten).

Selbstständigkeitserklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt habe und nur die im Literaturverzeichnis angefügten Quellen und Hilfsmittel benutzt habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift