Aufgabe zum Mergesort

(*Implementierung der merge-Methode, Komplexität*)

Das Sortierverfahren *Mergesort*, das nach der Strategie *Divide-and-Conquer* arbeitet, sortiert eine Sequenz, indem die Sequenz in zwei Teile zerlegt wird, die dann einzeln sortiert und wieder zu einer sortierten Sequenz zusammengemischt werden (to merge = zusammenmischen, verschmelzen).

(a) Gegeben seien folgende Methoden:

```
public int[] mergesort(int[] s) {
  int[] left = new int[s.length / 2];
  int[] right = new int[s.length - (s.length / 2)];
  int[] result;

  if (s.length <= 1) {
    result = s;
  } else {
    for (int i = 0; i < s.length / 2; i++) {
      left[i] = s[i];
    }
    int a = 0;
    for (int j = (s.length / 2); j < s.length; j++) {
      right[a++] = s[j];
    }
    result = merge(mergesort(left), mergesort(right));
  }
  return result;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java`

Schreiben Sie die Methode `public int[] merge (int[] s, int[] r)`, die die beiden aufsteigend sortierten Sequenzen `s` und `r` zu einer aufsteigend sortierten Sequenz zusammenmischt.

Lösungsvorschlag

```
public int[] merge(int[] s, int[] r) {
  int[] ergebnis = new int[s.length + r.length];
  int indexLinks = 0;
  int indexRechts = 0;
  int indexErgebnis = 0;

  // Im Reisverschlussverfahren s und r sortiert zusammenfügen.
  while (indexLinks < s.length && indexRechts < r.length) {
    if (s[indexLinks] < r[indexRechts]) {
      ergebnis[indexErgebnis] = s[indexLinks++];
    } else {
      ergebnis[indexErgebnis] = r[indexRechts++];
    }
    indexErgebnis++;
  }

  // Übrig gebliebene Elemente von s einfügen.
```

```
    while (indexLinks < s.length) {
      ergebnis[indexErgebnis++] = s[indexLinks++];
    }

    // Übrig gebliebene Elemente von r einfügen.
    while (indexRechts < r.length) {
      ergebnis[indexErgebnis++] = r[indexRechts++];
    }

    return ergebnis;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java`

(b) Analysieren Sie die Zeitkomplexität von `mergesort`.

Lösungsvorschlag

$\mathcal{O}(n \cdot \log n)$

Erklärung

Mergesort ist ein stabiles Sortierverfahren, vorausgesetzt der Merge-Schritt ist korrekt implementiert. Seine Komplexität beträgt im Worst-, Best- und Average-Case in Landau-Notation ausgedrückt stets $\mathcal{O}(n \cdot \log n)$. Für die Laufzeit $T(n)$ von Mergesort bei n zu sortierenden Elementen gilt die Rekursionsformel

$$
\begin{aligned}
T(n) = & \\
& T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + && \text{Aufwand, 1. Teil zu sortieren} \\
& T\left(\left\lceil \frac{n}{2} \right\rceil\right) + && \text{Aufwand, 2. Teil zu sortieren} \\
& \mathcal{O}(n) && \text{Aufwand, beide Teile zu verschmelzen}
\end{aligned}
$$

mit dem Rekursionsanfang $T(1) = 1$.

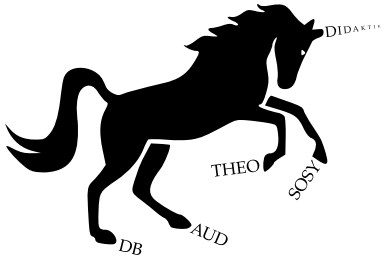
Nach dem Master-Theorem kann die Rekursionsformel durch

$$2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$$

bzw.

$$2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n$$

approximiert werden mit jeweils der Lösung $T(n) = \mathcal{O}(n \cdot \log n)$.



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieses Dokuments kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben/blob/main/Module/30_AUD/40_Sortieralgorithmen/50_Mergesort/Aufgabe_Implementierung-merge-Methode.tex`