

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 zusammen gemischt werden (to merge = zusammenmischen, verschmelzen).

(a) Gegeben seien folgende Methoden:

```
13 public int[] mergesort(int[] s) {
14     int[] left = new int[s.length / 2];
15     int[] right = new int[s.length - (s.length / 2)];
16     int[] result;
17
18     if (s.length <= 1) {
19         result = s;
20     } else {
21         for (int i = 0; i < s.length / 2; i++) {
22             left[i] = s[i];
23         }
24         int a = 0;
25         for (int j = (s.length / 2); j < s.length; j++) {
26             right[a++] = s[j];
27         }
28         result = merge(mergesort(left), mergesort(right));
29     }
30     return result;
31 }
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java](https://github.com/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.

```
33 public int[] merge(int[] s, int[] r) {
34     int[] ergebnis = new int[s.length + r.length];
35     int indexLinks = 0;
36     int indexRechts = 0;
37     int indexErgebnis = 0;
38
39     // Im Reisverschlussverfahren s und r sortiert zusammenfügen.
40     while (indexLinks < s.length && indexRechts < r.length) {
41         if (s[indexLinks] < r[indexRechts]) {
42             ergebnis[indexErgebnis] = s[indexLinks++];
43         } else {
44             ergebnis[indexErgebnis] = r[indexRechts++];
45         }
46         indexErgebnis++;
47     }
48
49     // Übrig gebliebene Elemente von s einfügen.
50     while (indexLinks < s.length) {
51         ergebnis[indexErgebnis++] = s[indexLinks++];
52     }
53 }
```

```

54 // Übrig gebliebene Elemente von r einfügen.
55 while (indexRechts < r.length) {
56     ergebnis[indexErgebnis++] = r[indexRechts++];
57 }
58
59 return ergebnis;
60 }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/aufgaben/aud/sortier/Mergesort.java](https://github.com/bschlangaul/aufgaben/aud/sortier/Mergesort.java)

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

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

Erklärung

Mergesort ist ein stabiles Sortierverfahren, vorausgesetzt der Merges-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

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

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 $T(n) = \mathcal{O}(n \cdot \log n)$.