Implementierung der `merge`-Methode, Berechnung der Zeitkomplexität[1]

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:

```java
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/ab_7/mergesort/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.

```java
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++];
```

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen: Aufgabenblatt 7: Wiederholung*, Seite 2, Aufgabe 3: Mergesort.

```java
    }

    // 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/ab_7/mergesort/Mergesort.java`

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

$O(n \cdot \log n)$

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 $O(n \cdot \log n)$. 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 $T(n) = O(n \cdot \log n)$.

Aufgabe 1[2]

Bestimmen Sie mit Hilfe des Master-Theorems für die folgenden Rekursionsgleichungen möglichst scharfe asymptotische untere und obere Schranken, falls das Master-Theorem anwendbar ist! Geben Sie andernfalls eine kurze Begründung, warum das Master-Theorem nicht anwendbar ist!

(a) $T(n) = 16 \cdot T\left(\frac{n}{2}\right) + 40n - 6$

(b) $T(n) = 27 \cdot T\left(\frac{n}{3}\right) + 3n^2 \log n$

(c) $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 3n^2 + \log n$

(d) $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 100 \log n + \sqrt{2n} + n^{-2}$

Beispiel-Aufgabe zum Master-Theorem[3][4]

(a) Betrachten Sie die folgende Methode `m` in Java, die initial mit `m(r, 0, r.length)` für das Array `r` aufgerufen wird. Geben Sie dazu eine Rekursionsgleichung $T(n)$ an, welche die Anzahl an Rechenschritten von `m` in Abhängigkeit von der Länge `n = r.length` berechnet.

```java
public static int m(int[] r, int lo, int hi) {
  if (lo < 8 || hi <= 10 || lo >= r.length || hi > r.length) {
    throw new IllegalArgumentException();
  }

  if (hi - lo == 1) {
    return r[lo];
  } else if (hi - lo == 2) {
    return Math.max(r[lo], r[lo + 1]); // O(1)
  } else {
    int s = (hi - lo) / 3;
    int x = m(r, lo, lo + s);
    int y = m(r, lo + s, lo + 2 * s);
    int z = m(r, lo + 2 * s, hi);
    return Math.max(Math.max(x, y), 2); // O(1)
  }
}
```

Allgemeine Rekursionsgleichung: $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

a (Anzahl der rekursiven Aufrufe): 3

b (Um welchen Anteil wird das Problem durch den Aufruf verkleinert): um $\frac{1}{3}$ also $b = 3$

Für den obigen Code: $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + \mathcal{O}(1)$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_3 3 - \varepsilon}\right) = \mathcal{O}\left(n^{1-\varepsilon}\right) = \mathcal{O}(1)$ für $\varepsilon = 1$

2. Fall: $f(n) \notin \Theta\left(n^{\log_3 3}\right) = \Theta\left(n^1\right)$

3. Fall: $f(n) \notin \Omega\left(n^{\log_3 3 + \varepsilon}\right) = \Omega\left(n^{1+\varepsilon}\right)$

[2] **66115:2011:03**.

[4] *Staatsexamen 66115 Theoretische Informatik / Algorithmen (vertieft) 2018 Frühjahr*, Thema 2 Aufgabe 6.

Also: $T(n) \in \Theta\left(n^{\log_b a}\right)$

Aufgabe 6[5]

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

Bestimmen und begründen Sie formal mit Hilfe dieses Satzes welche Komplexität folgende Laufzeitfunktionen haben.

(a) $T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 5n^2$

(b) $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 5n^2$

Aufgabe 4[6]

(a) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```java
int mystery(int n) {
  int a = 0, b = 0;
  int i = 0;
  while (i < n) {
    a = b + i;
    b = a;
    i = i + 1;
  }
  return a;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery1.java`

Bestimmen Sie die asymptotische worst-case Laufzeit des Code-Beispiels in $\mathcal{O}$-Notation bezüglich der Problemgröße n. Begründen Sie Ihre Antwort.

Die asymptotische worst-case Laufzeit des Code-Beispiels in $\mathcal{O}$-Notation ist $\mathcal{O}(n)$.

Die `while`-Schleife wird genau n mal ausgeführt. In der Schleife wird die Variable `i` in der Zeile `i = i + 1;` inkrementiert. `i` wird mit 0 initialisiert. Die `while`-Schleife endet wenn `i` gleich groß ist wir `n`.

(b) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```java
int mystery(int n) {
  int r = 0;
  while (n > 0) {
    int y = n;
    int x = n;
    for (int i = 0; i < y; i++) {
      for (int j = 0; j < i; j++) {
        r = r + 1;
```

[5] **66115:2019:09**.
[6] **66115:2020:09**.

```
            }
            r = r - 1;
          }
          n = n - 1;
        }
        return r;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery2.java`

Bestimmen Sie für das Code-Beispiel die asymptotische worst-case Laufzeit in $\mathcal{O}$-Notation bezüglich der Problemgröße n. Begründen Sie Ihre Antwort.

`while`: n-mal

1. `for`: $n, n-1, \dots, 2, 1$

2. `for`: $1, 2, \dots, n-1, n$

$n \times n \times n = \mathcal{O}(n^3)$

(c) Bestimmen Sie eine asymptotische Lösung (in Θ-Schreibweise) für die folgende Rekursionsgleichung:

$$T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$$

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

a**:** $a = 1$

b**:** $b = 2$

$f(n)$**:** $f(n) = \frac{1}{2}n^2 + n$

$\log_b a = \log_2 1 = 0$

Erster Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$

$\frac{1}{2}n^2 + n \notin \mathcal{O}\left(n^{-1}\right)$

Zweiter Fall: $f(n) \in \Theta\left(n^{\log_b a}\right)$

$\frac{1}{2}n^2 + n \notin \Theta(1)$

Dritter Fall: $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$

$\varepsilon = 2$

$\frac{1}{2}n^2 + n \in \Omega\left(n^2\right)$

Für eine Abschätzung suchen wir eine Konstante, damit gilt:

$1 \cdot f\left(\frac{n}{2}\right) \le c \cdot f(n)$

$\frac{1}{2} \cdot \frac{1}{4}n^2 + \frac{1}{2}n \le c \cdot \left(\frac{1}{2} \cdot n^2 + n\right)$

Damit folgt $c = \frac{1}{4}$

und $0 < c < 1$

$\Rightarrow \Theta(\frac{1}{2}n^2 + n)$
$\Rightarrow \Theta(n^2)$