Laufzeitanalyse (1) demogr.

```
for (int j = 1; j <= n; j = j * 2) {
    print(j);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Laufzeitanalyse (1) demogr.

```
for (int j = 1; j <= n; j = j * 2) {
    print(j);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Auflösung:

Laufzeitanalyse (1) demogr.

```
for (int j = 1; j <= n; j = j * 2) {
    print(j);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Auflösung: (1) $\Theta(\log n)$

Laufzeitanalyse (2) demogr.

```
int i=1;
for (int j = 1; j <= n; j = j+2*i) {
    i++;
    print(j-i*3);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Laufzeitanalyse (2) demogr.

```
int i=1;
for (int j = 1; j <= n; j = j+2*i) {
    i++;
    print(j-i*3);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Auflösung:

Laufzeitanalyse (2) demogr.

```
int i=1;
for (int j = 1; j <= n; j = j+2*i) {
    i++;
    print(j-i*3);
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(\log n)$
- (2) $\Theta(\sqrt{n})$
- (3) $\Theta(n)$
- (4) $\Theta(n \log n)$
- (5) $\Theta(n^2)$

Auflösung: (2) $\Theta(\sqrt{n})$

Laufzeitanalyse (3) demogr.

```
int j;
for (i = 1; i <= n; i++) {
    j = 3;
    while (j < n) {
        j = j * j * j;
    }
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(n)$
- (2) $\Theta(n \log \log n)$
- (3) $\Theta(n \log n)$
- (4) $\Theta(n \log^2 n)$
- (5) $\Theta(n^2)$

Laufzeitanalyse (3) demogr.

```
int j;
for (i = 1; i <= n; i++) {
    j = 3;
    while (j < n) {
        j = j * j * j;
    }
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(n)$
- (2) $\Theta(n \log \log n)$
- (3) $\Theta(n \log n)$
- (4) $\Theta(n \log^2 n)$
- (5) $\Theta(n^2)$

Auflösung:

Laufzeitanalyse (3) demogr.

```
int j;
for (i = 1; i <= n; i++) {
    j = 3;
    while (j < n) {
        j = j * j * j;
    }
}
```

Die Laufzeit in Abhängigkeit von n ist

- (1) $\Theta(n)$
- (2) $\Theta(n \log \log n)$
- (3) $\Theta(n \log n)$
- (4) $\Theta(n \log^2 n)$
- (5) $\Theta(n^2)$

Auflösung: (2) $\Theta(n \log \log n)$

Huffman-Codes Beispiel multiple choice

Gegeben folgender Text über dem Alphabet $\Sigma = \{a, b, c, d\}$:

d c d c b a b a b a

Welches ist ein Huffman-Code für diesen Text:

- (1) $a \rightarrow 00$, $b \rightarrow 01$, $c \rightarrow 10$, $d \rightarrow 11$
- (2) $a \rightarrow 0$, $b \rightarrow 1$, $c \rightarrow 00$, $d \rightarrow 01$
- (3) $a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, $d \rightarrow 111$
- (4) $a \rightarrow 11$, $b \rightarrow 10$, $c \rightarrow 01$, $d \rightarrow 00$

Huffman-Codes Beispiel multiple choice

Gegeben folgender Text über dem Alphabet $\Sigma = \{a, b, c, d\}$:

d c d c b a b a b a

Welches ist ein Huffman-Code für diesen Text:

- (1) $a \rightarrow$ 00, $b \rightarrow$ 01, $c \rightarrow$ 10, $d \rightarrow$ 11
- (2) $a \rightarrow$ 0, $b \rightarrow$ 1, $c \rightarrow$ 00, $d \rightarrow$ 01
- (3) $a \rightarrow$ 0, $b \rightarrow$ 10, $c \rightarrow$ 110, $d \rightarrow$ 111
- (4) $a \rightarrow$ 11, $b \rightarrow$ 10, $c \rightarrow$ 01, $d \rightarrow$ 00

Auflösung:

Huffman-Codes Beispiel multiple choice

Gegeben folgender Text über dem Alphabet $\Sigma = \{a, b, c, d\}$:

d c d c b a b a b a

Welches ist ein Huffman-Code für diesen Text:

- (1) $a \rightarrow 00$, $b \rightarrow 01$, $c \rightarrow 10$, $d \rightarrow 11$
- (2) $a \rightarrow 0$, $b \rightarrow 1$, $c \rightarrow 00$, $d \rightarrow 01$
- (3) $a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, $d \rightarrow 111$
- (4) $a \rightarrow 11$, $b \rightarrow 10$, $c \rightarrow 01$, $d \rightarrow 00$

Auflösung: (1) & (4)

Einordnung von Quicksort multiple choice

Quicksort ist ein Vertreter

- (1) der Greedy-Algorithmen.
- (2) der Divide-and-Conquer Algorithmen.
- (3) der dynamischen Programmierung.
- (4) keiner dieser Klassen.
- (5) keine Ahnung.

Einordnung von Quicksort multiple choice

Quicksort ist ein Vertreter

- (1) der Greedy-Algorithmen.
- (2) der Divide-and-Conquer Algorithmen.
- (3) der dynamischen Programmierung.
- (4) keiner dieser Klassen.
- (5) keine Ahnung.

Auflösung:

Einordnung von Quicksort multiple choice

Quicksort ist ein Vertreter

- (1) der Greedy-Algorithmen.
- (2) der Divide-and-Conquer Algorithmen.
- (3) der dynamischen Programmierung.
- (4) keiner dieser Klassen.
- (5) keine Ahnung.

Auflösung: (2) (und damit auch (3))

Dynamische Programmierung demogr.

Bei dynamischer Programmierung ist der unterliegende Aufrufgraph kein Baum (sondern ein DAG) und deshalb ist der benötigte Speicherplatz immer superlinear in der Länge der Eingabe.

- (1) stimmt.
- (2) stimmt nicht.
- (3) keine Ahnung.

Dynamische Programmierung demogr.

Bei dynamischer Programmierung ist der unterliegende Aufrufgraph kein Baum (sondern ein DAG) und deshalb ist der benötigte Speicherplatz immer superlinear in der Länge der Eingabe.

- (1) stimmt.
- (2) stimmt nicht.
- (3) keine Ahnung.

Auflösung:

Dynamische Programmierung demogr.

Bei dynamischer Programmierung ist der unterliegende Aufrufgraph kein Baum (sondern ein DAG) und deshalb ist der benötigte Speicherplatz immer superlinear in der Länge der Eingabe.

- (1) stimmt.
- (2) stimmt nicht.
- (3) keine Ahnung.

Auflösung: (2)

Dynamische Programmierung demogr.

Bei dynamischer Programmierung ist der unterliegende Aufrufgraph kein Baum (sondern ein DAG) und deshalb ist der benötigte Speicherplatz immer superlinear in der Länge der Eingabe.

- (1) stimmt.
- (2) stimmt nicht.
- (3) keine Ahnung.

Auflösung: (2)

Bsp: $O(n)$ Platz für Fibonacci-Folge $f(n) = f(n-1) + f(n-2)$.