Laufzeitanalyse (1) demogr.

```
int i=1;
for (int j = 1; j <= n; j = j+i) {
    i++;
    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.

```
int i=1;
for (int j = 1; j <= n; j = j+i) {
    i++;
    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.

```
int i=1;
for (int j = 1; j <= n; j = j+i) {
    i++;
    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: (2) $\Theta(\sqrt{n})$.

Laufzeitanalyse (2) demogr.

```
for (int i = 1; i < n; i++) {
    for (int j = 0; j < log i; j++) {
        print(j + i);
    }
}
```

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.

```
for (int i = 1; i < n; i++) {
    for (int j = 0; j < log i; j++) {
        print(j + i);
    }
}
```

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.

```
for (int i = 1; i < n; i++) {
    for (int j = 0; j < log i; j++) {
        print(j + i);
    }
}
```

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: (4) $\Theta(n \log n)$.

Laufzeitanalyse (3) demogr.

```
int j;
for (i = 1; i <= n; i++) {
    j = 2;
    while (j < n) {
        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 = 2;
    while (j < n) {
        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 = 2;
    while (j < n) {
        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) $\Theta(n \log \log n)$.

Minimale Spannbäume multiple choice

Der Algorithmus von Kruskal benötigt zur korrekten Ausführung

- (1) Positive Kantengewichte.
- (2) Kreisfreiheit bei negativen Kantengewichten.
- (3) Einen stark zusammenhängenden Graphen.
- (4) Keine dieser Bedingungen.

Minimale Spannbäume multiple choice

Der Algorithmus von Kruskal benötigt zur korrekten Ausführung

- (1) Positive Kantengewichte.
- (2) Kreisfreiheit bei negativen Kantengewichten.
- (3) Einen stark zusammenhängenden Graphen.
- (4) Keine dieser Bedingungen.

Auflösung:

Minimale Spannbäume multiple choice

Der Algorithmus von Kruskal benötigt zur korrekten Ausführung

- (1) Positive Kantengewichte.
- (2) Kreisfreiheit bei negativen Kantengewichten.
- (3) Einen stark zusammenhängenden Graphen.
- (4) Keine dieser Bedingungen.

Auflösung: (4)

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung:

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung: (2)

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung: (2)

Und was passiert, wenn man zu jedem Kantengewicht eine Konstante $c > 0$ dazu addiert?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung: (2)

Und was passiert, wenn man zu jedem Kantengewicht eine Konstante $c > 0$ dazu addiert?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung:

Stabilität minimaler Spannbäume demogr.

Was passiert mit einem minimalen Spannbaum T, wenn bei jeder Kante des Ursprungsgraphen das Kantengewicht verdoppelt wird?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung: (2)

Und was passiert, wenn man zu jedem Kantengewicht eine Konstante $c > 0$ dazu addiert?

- (1) T kann sich ändern.
- (2) T bleibt immer gleich.

Auflösung: (2)

Union-Find demogr.

Bei `union(i,j)` wird der kleinere Baum an die Wurzel des größeren Baumes gehängt. Hier gibt es zwei Strategien:

- (A) Der “kleinere Baum” ist immer der Baum mit weniger Tiefe
- (B) Der “kleinere Baum” ist immer der Baum mit weniger Knoten

Welche Strategie garantiert eine worst-case Laufzeit von $\Theta(\log_2 |V|)$ für die `find(u)`-Operation?

- (1) Nur (A)
- (2) Nur (B)
- (3) Beide

Union-Find demogr.

Bei `union(i,j)` wird der kleinere Baum an die Wurzel des größeren Baumes gehängt. Hier gibt es zwei Strategien:

- (A) Der “kleinere Baum” ist immer der Baum mit weniger Tiefe
- (B) Der “kleinere Baum” ist immer der Baum mit weniger Knoten

Welche Strategie garantiert eine worst-case Laufzeit von $\Theta(\log_2 |V|)$ für die `find(u)`-Operation?

- (1) Nur (A)
- (2) Nur (B)
- (3) Beide

Auflösung:

Union-Find demogr.

Bei `union(i,j)` wird der kleinere Baum an die Wurzel des größeren Baumes gehängt. Hier gibt es zwei Strategien:

- (A) Der “kleinere Baum” ist immer der Baum mit weniger Tiefe
- (B) Der “kleinere Baum” ist immer der Baum mit weniger Knoten

Welche Strategie garantiert eine worst-case Laufzeit von $\Theta(\log_2 |V|)$ für die `find(u)`-Operation?

- (1) Nur (A)
- (2) Nur (B)
- (3) Beide

Auflösung: (3)