



Übungslektion 6 – Asymptotik & Sortierung

Informatik II

25. / 26. März 2025

Willkommen!

Polybox



Passwort: jschul

Personal Website



<https://n.ethz.ch/~jschul>

Heutiges Programm

Wiederholung Theorie

Asymptotische Laufzeit

Sortieralgorithmen

Alte Prüfungsaufgaben

Hausaufgaben

1. Wiederholung Theorie

Asymptotisches Verhalten

- Was sind $\Omega(g(n))$, $\Theta(g(n))$, $\mathcal{O}(g(n))$?

Asymptotisches Verhalten

- Was sind $\Omega(g(n))$, $\Theta(g(n))$, $\mathcal{O}(g(n))$?
- ➔ Mengen von Funktionen!

Asymptotisches Verhalten

■ Was sind $\Omega(g(n))$, $\Theta(g(n))$, $\mathcal{O}(g(n))$?

➔ Mengen von Funktionen!

Wiederholung, Mengen A, B :

Teilmenge $A \subseteq B$

echte Teilmenge $A \subsetneq B$

Schnittmenge $A \cap B$

Asymptotisches Verhalten

Gegeben Funktion $f : \mathbb{N} \rightarrow \mathbb{R}$.

Definition:

$$\mathcal{O}(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} \mid \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)\}$$

$$\Omega(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} \mid \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n)\}$$

$$\Theta(g) = \mathcal{O}(g) \cap \Omega(g)$$

Intuition:

$f \in \mathcal{O}(g)$: (Laufzeit) f wächst asymptotisch **nicht mehr** als g . Algorithmus mit Laufzeit f ist **nicht schlechter** als einer mit g .

$f \in \Omega(g)$: (Laufzeit) f wächst asymptotisch **nicht weniger** als g . Algorithmus mit Laufzeit f ist **nicht besser** als einer mit g .

$f \in \Theta(g)$: f wächst asymptotisch **gleich schnell** wie g . Algorithmus mit Laufzeit f ist **gleich gut** wie einer mit g .

Asymptotisches Verhalten

Einige nützliche Formeln:



$$\sum_{i=0}^{n-1} 1 = n$$

Asymptotisches Verhalten

Einige nützliche Formeln:



$$\sum_{i=0}^{n-1} 1 = n$$



$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

Asymptotisches Verhalten

Einige nützliche Formeln:



$$\sum_{i=0}^{n-1} 1 = n$$



$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$



$$\sum_{i=0}^n i^2 = \frac{n \cdot (n + 1)(2n + 1)}{6}$$

(das muss man nicht auswendig wissen)

Formeln auf ZF

Größenordnung von Funktionen

$$c < \log(n) < \sqrt{n} < n < n \cdot \log(n) < n^2 < 2^n < n! < n^n$$

Rechenregeln

Summen:

$$\begin{aligned}\sum_{i=0}^{n-1} c &= \sum_{i=0}^n c = c \cdot n \in \Theta(n) \\ \sum_{i=0}^n (n-i) &= \sum_{i=0}^n i = \frac{n(n+1)}{2} \in \Theta(n^2) \\ \sum_{i=0}^{n^a} i^b &\in \Theta(n^{a \cdot (b+1)}) \\ \sum_{i=0}^n 2^i &= \frac{2^{n+1} - 1}{2 - 1} \in \Theta(2^n)\end{aligned}$$

Logarithmusfunktionen:

$$\begin{aligned}\log n^n &= n \log n \in \Theta(n \log n) \\ \log n! &= n \log n - n \in \Theta(n \log n)\end{aligned}$$

Loris Frey

$$\begin{aligned}\mathcal{O}(1) &< \mathcal{O}(\log(n)) < \mathcal{O}(\sqrt{n}) < \mathcal{O}(n) < \mathcal{O}(n \log(n)) \\ &< \mathcal{O}(n^2) < \mathcal{O}(2^n) < \mathcal{O}(n!) < \mathcal{O}(n^n)\end{aligned}$$

Wobei $\log(\dots)$ in den meisten Fällen der Logarithmus zur Basis 2 ist

Achtung: $n = \mathcal{O}(n^2)$ und $n^2 = \mathcal{O}(n^2)$, dies impliziert aber nicht, dass die Laufzeiten von n und n^2 asymptotisch sind

Für die Laufzeitenanalyse sind folgende Formeln hilfreich:

| | |
|----------------------------|---|
| Grundlagen Logarithmus: | $c^a = n \rightarrow a = \log_c(n), \quad \log(n^a) = a \log(n)$ |
| Vereinfachungen Sigma: | $\sum_{i=0}^{n-1} c = \sum_{i=1}^n c, \quad \sum_{i=0}^n (n-i) = \sum_{i=0}^n i$ |
| Asymptotik Sigma: | $\sum_{i=0}^{n^a} i^b = \Theta(n^{a(b+1)}), \quad \sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}$ |

Max Schaldach

2. Asymptotische Laufzeit

Asymptotische Laufzeiten mit Θ

CodeExpert In-Class Aufgabe: **Nicht-rekursive Snippets Laufzeiten**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Asymptotische Laufzeiten mit Θ

Snippet 0. (not on CodeExpert)

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, n):
            op()
```

- Wie oft wird `op()` aufgerufen?

Asymptotische Laufzeiten mit Θ

Snippet 0. (not on CodeExpert)

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, n):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$n \cdot n \in \Theta(n^2)$$

Asymptotische Laufzeiten mit Θ

Snippet 1.

- Wie oft wird `op()` aufgerufen?

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m):
            op()
```

Asymptotische Laufzeiten mit Θ

Snippet 1.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\sum_{m=0}^{n-1} \sum_{k=0}^{m-1} 1$$

Asymptotische Laufzeiten mit Θ

Snippet 1.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\sum_{m=0}^{n-1} \sum_{k=0}^{m-1} 1 = \sum_{m=0}^{n-1} m$$

Asymptotische Laufzeiten mit Θ

Snippet 1.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\begin{aligned} \sum_{m=0}^{n-1} \sum_{k=0}^{m-1} 1 &= \sum_{m=0}^{n-1} m \\ &= \frac{(n-1) \cdot n}{2} \end{aligned}$$

Asymptotische Laufzeiten mit Θ

Snippet 1.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\begin{aligned}\sum_{m=0}^{n-1} \sum_{k=0}^{m-1} 1 &= \sum_{m=0}^{n-1} m \\ &= \frac{(n-1) \cdot n}{2} \in \Theta(n^2)\end{aligned}$$

Asymptotische Laufzeiten mit Θ

- Wie oft wird `op()` aufgerufen?

Snippet 2.

```
# pre: n is a positive integer
def run(n):
    count = 0
    while n // (2 ** count) >= 1:
        op()
        count += 1
```

Asymptotische Laufzeiten mit Θ

Snippet 2.

```
# pre: n is a positive integer
def run(n):
    count = 0
    while n // (2 ** count) >= 1:
        op()
        count += 1
```

- Wie oft wird `op()` aufgerufen?
- Antwort: $\Theta(\log n)$.

Asymptotische Laufzeiten mit Θ

Snippet 2.

```
# pre: n is a positive integer
def run(n):
    count = 0
    while n // (2 ** count) >= 1:
        op()
        count += 1
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- **op()** wird so lange aufgerufen, wie $n/2^{\text{count}} \geq 1$, d.h. solange $\text{count} \leq \log n$

Asymptotische Laufzeiten mit Θ

Snippet 2.

```
# pre: n is a positive integer
def run(n):
    count = 0
    while n // (2 ** count) >= 1:
        op()
        count += 1
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- **op()** wird so lange aufgerufen, wie $n/2^{\text{count}} \geq 1$, d.h. solange $\text{count} \leq \log n$
- Beachten Sie, dass *count* die Anzahl der Iterationen in der Schleife verfolgt.

Asymptotische Laufzeiten mit Θ

Snippet 2.

```
# pre: n is a positive integer
def run(n):
    count = 0
    while n // (2 ** count) >= 1:
        op()
        count += 1
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- **op()** wird so lange aufgerufen, wie $n/2^{\text{count}} \geq 1$, d.h. solange $\text{count} \leq \log n$
- Beachten Sie, dass *count* die Anzahl der Iterationen in der Schleife verfolgt.
- Daher wird **op()** zwischen $\lfloor \log n \rfloor$ und $\lceil \log n \rceil$ mal aufgerufen.

Asymptotische Laufzeiten mit Θ

Snippet 3.

```
def run(n):  
    l = 0  
    r = n  
    while l < r:  
        op()  
        m = (l + r) // 2  
        if h(m):  
            l = m + 1  
        else:  
            r = m - 1  
  
# n is an integer  
# h returns a bool
```

■ Wie oft wird **op()** aufgerufen?

Asymptotische Laufzeiten mit Θ

Snippet 3.

```
def run(n):  
    l = 0  
    r = n  
    while l < r:  
        op()  
        m = (l + r) // 2  
        if h(m):  
            l = m + 1  
        else:  
            r = m - 1  
  
# n is an integer  
# h returns a bool
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.

Asymptotische Laufzeiten mit Θ

Snippet 3.

```
def run(n):  
    l = 0  
    r = n  
    while l < r:  
        op()  
        m = (l + r) // 2  
        if h(m):  
            l = m + 1  
        else:  
            r = m - 1  
  
# n is an integer  
# h returns a bool
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- Ähnlich wie bei Snippet 2, wenn man den Wert von $r - l$ betrachtet.

Asymptotische Laufzeiten mit Θ

Snippet 3.

```
def run(n):  
    l = 0  
    r = n  
    while l < r:  
        op()  
        m = (l + r) // 2  
        if h(m):  
            l = m + 1  
        else:  
            r = m - 1  
  
# n is an integer  
# h returns a bool
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- Ähnlich wie bei Snippet 2, wenn man den Wert von $r - l$ betrachtet.
- Am Anfang haben wir $r - l = n$.

Asymptotische Laufzeiten mit Θ

Snippet 3.

```
def run(n):  
    l = 0  
    r = n  
    while l < r:  
        op()  
        m = (l + r) // 2  
        if h(m):  
            l = m + 1  
        else:  
            r = m - 1  
  
# n is an integer  
# h returns a bool
```

- Wie oft wird **op()** aufgerufen?
- Antwort: $\Theta(\log n)$.
- Ähnlich wie bei Snippet 2, wenn man den Wert von $r - l$ betrachtet.
- Am Anfang haben wir $r - l = n$.
- Bei jeder Iteration wird l oder r auf den Mittelwert $m = (l + r)/2$ gesetzt.
- Bei der Iteration c gilt also immer, dass $n/2^c - 1 \leq r - l < n/2^c$.

Asymptotische Laufzeiten mit Θ

- Wie oft wird `op()` aufgerufen?

Snippet 4.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m*m):
            op()
```


Asymptotische Laufzeiten mit Θ

Snippet 4.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m*m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\sum_{m=0}^{n-1} \sum_{k=0}^{m^2-1} 1$$

Asymptotische Laufzeiten mit Θ

Snippet 4.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m*m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\sum_{m=0}^{n-1} \sum_{k=0}^{m^2-1} 1 = \sum_{m=0}^{n-1} m^2$$

Asymptotische Laufzeiten mit Θ

Snippet 4.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m*m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\begin{aligned} \sum_{m=0}^{n-1} \sum_{k=0}^{m^2-1} 1 &= \sum_{m=0}^{n-1} m^2 \\ &= \frac{(n-1) \cdot n \cdot (2n-1)}{6} \end{aligned}$$

Asymptotische Laufzeiten mit Θ

Snippet 4.

```
# pre: n is an integer
def run(n):
    for m in range(0, n):
        for k in range(0, m*m):
            op()
```

- Wie oft wird `op()` aufgerufen?
- Antwort:

$$\begin{aligned}\sum_{m=0}^{n-1} \sum_{k=0}^{m^2-1} 1 &= \sum_{m=0}^{n-1} m^2 \\ &= \frac{(n-1) \cdot n \cdot (2n-1)}{6} \\ &\in \Theta(n^3)\end{aligned}$$

3. Sortieralgorithmen

Sortieralgorithmen

Wie können wir ein Array nach möglichst schnell und effizient sortieren?

→ **Mit Sortieralgorithmen**

Die verschiedenen Algorithmen unterscheiden sich in:

- Strategie
- Laufzeit
- Speicherplatz

"15 sorting Algorithms in 6 Minutes"

<https://www.youtube.com/watch?v=kPRA0W1kECg>

Sortieralgorithmen

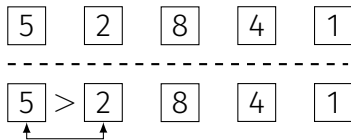
Wir schauen uns heute die folgenden an:

1. Bubblesort
2. Insertion sort (Sortieren durch Einfügen)

Bubblesort

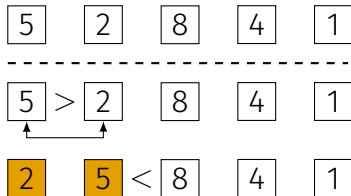
5 2 8 4 1

Bubblesort



- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubblesort



- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubblesort

5 2 8 4 1

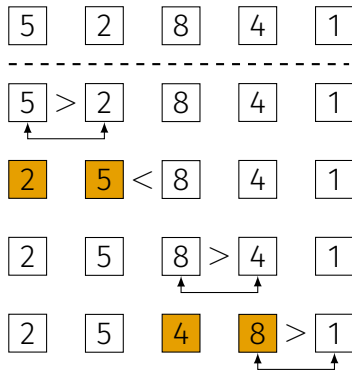
5 > 2 8 4 1

2 5 < 8 4 1

2 5 8 > 4 1

- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubblesort



- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubblesort

5 2 8 4 1

5 > 2 8 4 1
↑ ↑

2 5 < 8 4 1

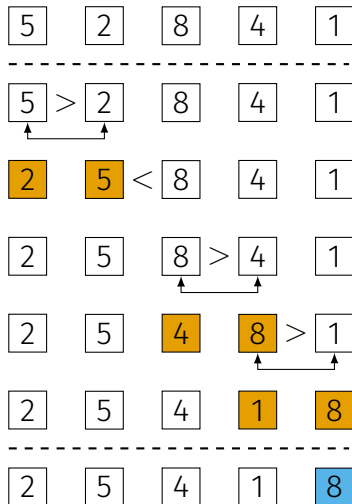
2 5 8 > 4 1
 ↑ ↑

2 5 4 8 > 1
 ↑ ↑

2 5 4 1 8

- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubblesort



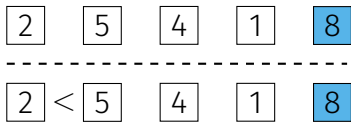
- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.
- Nach einer Iteration (Iteration 0) befindet sich das größte Element am Ende der Liste.

Bubblesort

2 5 4 1 8

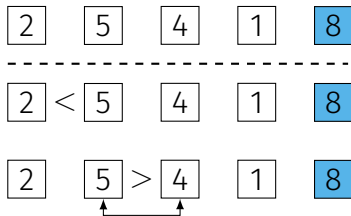
- Schleife, bis die Liste sortiert ist.

Bubblesort



- Schleife, bis die Liste sortiert ist.

Bubblesort



- Schleife, bis die Liste sortiert ist.

Bubblesort

2 5 4 1 8

2 < 5 4 1 8

2 5 > 4 1 8

2 4 5 > 1 8

- Schleife, bis die Liste sortiert ist.

Bubblesort

2 5 4 1 8

2 < 5 4 1 8

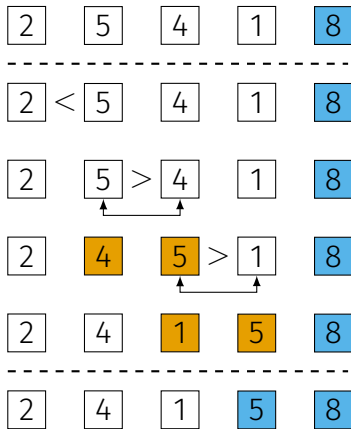
2 5 > 4 1 8

2 4 5 > 1 8

2 4 1 5 8

- Schleife, bis die Liste sortiert ist.

Bubblesort



- Schleife, bis die Liste sortiert ist.
- **Schleifeninvariante:**
Nach Iteration i sind Elemente $1 \dots i$ $[-(i+1) :]$ sortiert und an der richtigen Stelle.

Visualisation

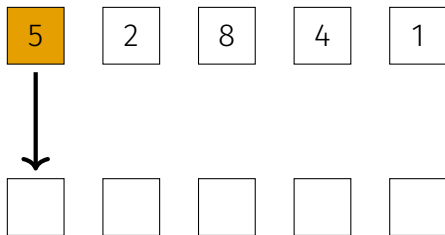
<https://visualgo.net/en/sorting>

Insertion Sort: Konzept



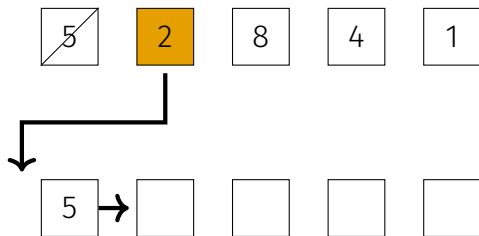
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



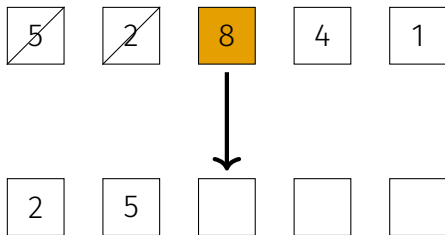
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



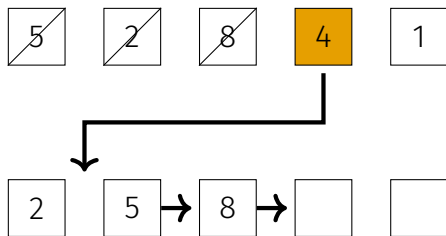
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



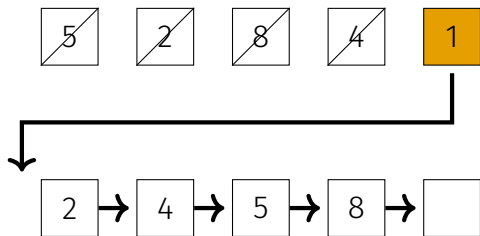
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



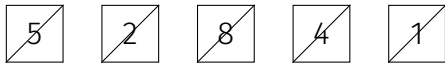
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



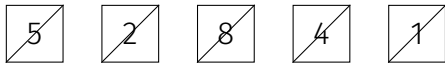
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



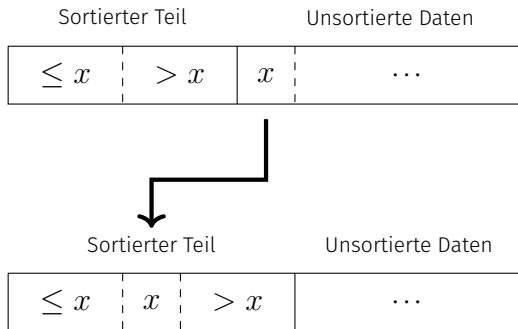
- Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.
- **Problem:** Benötigt n zusätzlichen Speicherplatz.

Insertion Sort

| Sortierter Teil | | Unsortierte Daten | |
|-----------------|-------|-------------------|---------|
| $\leq x$ | $> x$ | x | \dots |

Wir können den Speicherplatz, der in der ursprünglichen Liste frei wird verwenden.

Insertion Sort



Wir können den Speicherplatz, der in der ursprünglichen Liste frei wird verwenden.

Sortieren durch Einfügen

5 2 8 4 1

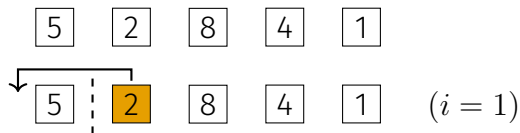
Sortieren durch Einfügen

■ **Schleifeninvariante:** ??

5 2 8 4 1

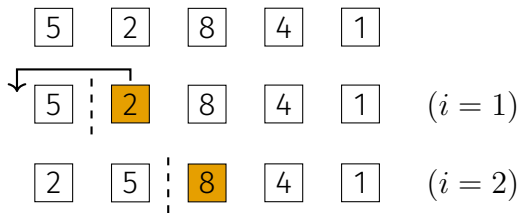
5 | 2 8 4 1 ($i = 1$)

Sortieren durch Einfügen



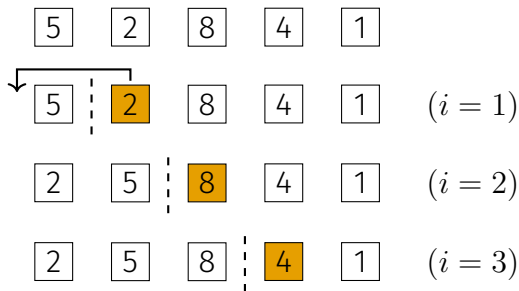
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.

Sortieren durch Einfügen



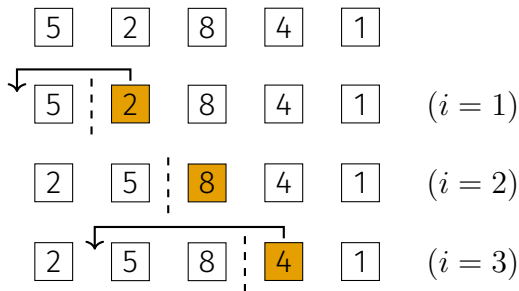
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $li[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $li[:i]$ die i niedrigsten Elemente von li in sortierter Reihenfolge.)
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $li[:i]$ einfügen.
- Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen



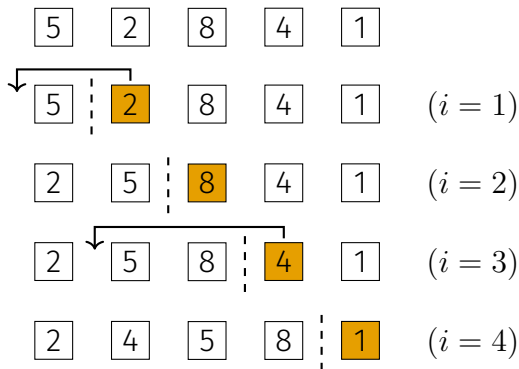
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen



- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)
- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.
- Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen

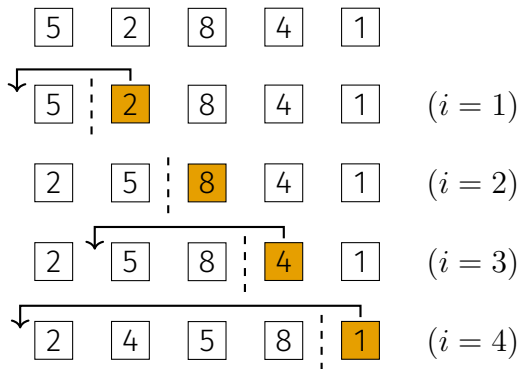


■ **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)

■ Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.

■ Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen

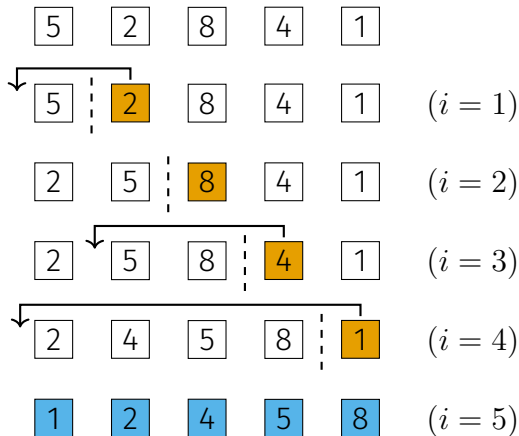


■ **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)

■ Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.

■ Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen

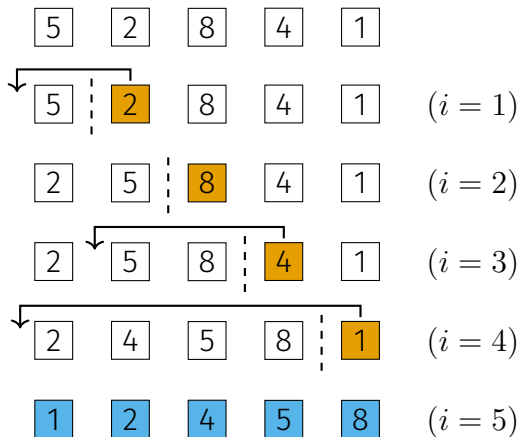


■ **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)

■ Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.

■ Wiederholen bis alles sortiert ist ($i = n$).

Sortieren durch Einfügen



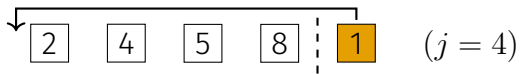
- **Schleifeninvariante:** Vor Iteration i sind Elemente in $1i[:i]$ sortiert. (Für Selection Sort: Vor iteration i enthält $1i[:i]$ die i niedrigsten Elemente von $1i$ in sortierter Reihenfolge.)

- Bei Iteration i , das i -te Element an der korrekten Position in die sortierte Teilliste $1i[:i]$ einfügen.

- Wiederholen bis alles sortiert ist ($i = n$).

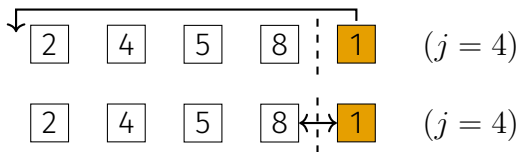
- **Frage:** Wie kann man die Insertion durchführen?

Einfügen durch Austauschen



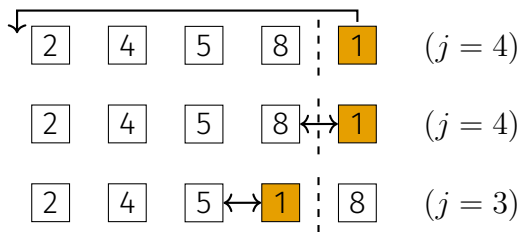
- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.

Einfügen durch Austauschen



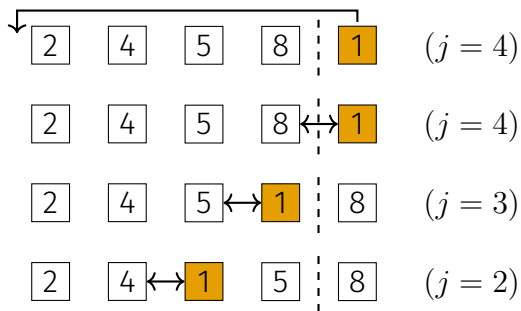
- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.
- Vergleiche $li[j]$ und $li[j-1]$ und tausche, falls $li[j-1] > li[j]$.

Einfügen durch Austauschen



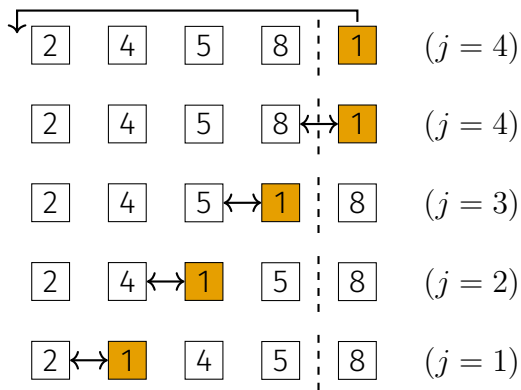
- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.
- Vergleiche $li[j]$ und $li[j-1]$ und tausche, falls $li[j-1] > li[j]$.
- Wiederhole bis $j = 0$ or $li[j-1] \leq li[j]$.

Einfügen durch Austauschen



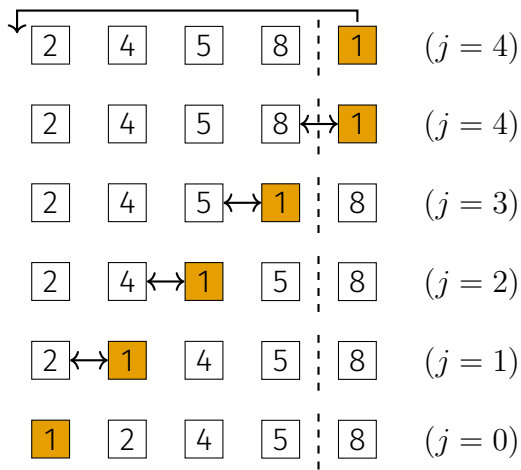
- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.
- Vergleiche $li[j]$ und $li[j-1]$ und tausche, falls $li[j-1] > li[j]$.
- Wiederhole bis $j = 0$ or $li[j-1] \leq li[j]$.

Einfügen durch Austauschen



- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.
- Vergleiche $li[j]$ und $li[j-1]$ und tausche, falls $li[j-1] > li[j]$.
- Wiederhole bis $j = 0$ or $li[j-1] \leq li[j]$.

Einfügen durch Austauschen



- Betrachten wir Iteration $i = 4$.
- Setze Variable $j = i$.
- Vergleiche $li[j]$ und $li[j-1]$ und tausche, falls $li[j-1] > li[j]$.
- Wiederhole bis $j = 0$ or $li[j-1] \leq li[j]$.

Implementierung von Insertion Sort

CodeExpert In-Class Aufgabe: **Insertion Sort Laufzeit**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Danach werden wir folgendes diskutieren:

- Was ist das worst-case Szenario für das Sortieren einer Liste mit Insertion Sort?
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
- Was ist das best-case Szenario für das Sortieren einer Liste mit Insertion Sort?
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?

Implementierung von Insertion Sort

CodeExpert In-Class Aufgabe: **Insertion Sort Laufzeit**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Danach werden wir folgendes diskutieren:

- Was ist das worst-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Falsch herum sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
- Was ist das best-case Szenario für das Sortieren einer Liste mit Insertion Sort?
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?

Implementierung von Insertion Sort

CodeExpert In-Class Aufgabe: **Insertion Sort Laufzeit**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Danach werden wir folgendes diskutieren:

- Was ist das worst-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Falsch herum sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
⇒ $\Theta(n^2)$
- Was ist das best-case Szenario für das Sortieren einer Liste mit Insertion Sort?
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?

Implementierung von Insertion Sort

CodeExpert In-Class Aufgabe: **Insertion Sort Laufzeit**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Danach werden wir folgendes diskutieren:

- Was ist das worst-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Falsch herum sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
⇒ $\Theta(n^2)$
- Was ist das best-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Bereits richtig sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?

Implementierung von Insertion Sort

CodeExpert In-Class Aufgabe: **Insertion Sort Laufzeit**

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Danach werden wir folgendes diskutieren:

- Was ist das worst-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Falsch herum sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
⇒ $\Theta(n^2)$
- Was ist das best-case Szenario für das Sortieren einer Liste mit Insertion Sort?
⇒ Bereits richtig sortiert
- Was ist die Θ Laufzeitkomplexität von Insertion Sort in diesem Fall?
⇒ $\Theta(n)$

4. Alte Prüfungsaufgaben

Runtime Analysis

MAVT 2023

| | | | |
|-------------------------------------|-----------------------------------|--------------------------------|-----------------------------------|
| A: $\Theta(1)$ | B: $\Theta(\log n)$ | C: $\Theta(\log^2 n)$ | D: $\Theta(\log \log n)$ |
| E: $\Theta(n)$ | F: $\Theta(n^2)$ | G: $\Theta(n^3)$ | H: $\Theta(n^4)$ |
| I: $\Theta(n \log n)$ | K: $\Theta(n^2 \log n)$ | L: $\Theta(n \log^2 n)$ | M: $\Theta(n^2 \log^2 n)$ |
| N: $\Theta(n^{\log_2 3})$ | O: $\Theta(\sqrt{n})$ | P: $\Theta(\sqrt{n^3})$ | Q: $\Theta(\sqrt{\log n})$ |
| R: $\Theta(\sqrt{n \log n})$ | S: $\Theta(\sqrt[3]{2^n})$ | T: $\Theta(2^n)$ | U: $\Theta(3^n)$ |
| V: $\Theta(n!)$ | W: $\Theta(n^n)$ | X: N/A | |

| $f(n)$ | $f \in \dots$ | $f(n)$ | $f \in \dots$ |
|-----------------------|---------------|--------------------------------------|---------------|
| 0. $2n$ | E | | |
| 1. $\sqrt{n} + n^4$ | H | 2. 2023^{2023} | A |
| 3. $\sum_{i=0}^n i^2$ | G | 4. $\log\left(\sum_{i=0}^n i\right)$ | B |
| 5. $n^2 + n \log n$ | F | | |

Runtime Analysis

BAUG 2024

| $f(n)$ $f \in \dots$ | | | $f(n)$ $f \in \dots$ | | |
|----------------------|----------------|---|----------------------|-------------------------|---|
| 0. | $2n$ | E | | | |
| 1. | $\binom{n}{2}$ | F | 2. | $\frac{10n}{2n+10}$ | A |
| 3. | 3^{2n} | X | 4. | $\sum_{i=1}^{\log n} i$ | C |

Runtime Analysis

MAVT 2023

(1.b)

```
def g(n):  
    for i in range(n):  
        for j in range(n):  
            for k in range(j):  
                f()
```

Asymptotische Anzahl Aufrufe von f / *Asymptotic number of calls to f :*

☐ 1 ☐ n ☐ $\log n$ ☐ \sqrt{n} ☐ n^2 ☐ 2^n ☐ $n \log n$ ☒ n^3

(1.d)

```
def g(n):  
    i = 1  
    while i <= n**4:  
        f()  
        i = i * 2
```

Asymptotische Anzahl Aufrufe von f / *Asymptotic number of calls to f :*

☐ 1 ☐ n ☒ $\log n$ ☐ \sqrt{n} ☐ n^2 ☐ 2^n ☐ $n \log n$ ☐ n^3

Runtime Analysis: Tipp

```
def g(n):  
    i = 1  
    while i <= n:  
        f()  
        i = i * 4
```

Nun denken wir uns eine counter-variable dazu, das Snippet sieht also so aus:

```
def g(n):  
    i = 1  
    c = 0  
    while i <= n:  
        f()  
        c = c + 1  
        i = i * 4
```

Nun können wir zwar nicht direkt c und n vergleichen, wohl aber i und c :
Wir erkennen, dass $i = 4^c$. Nun setzen wir dies in die Loop-Condition ein:
 $i = n \implies 4^c = n \implies c = \log_4 n = \Theta(\log n)$

Runtime Analysis

MAVT 2024FS

(1.1) $\sqrt{\sum_{i=1}^n i}$

- ☐ $\Theta(1)$ ☐ $\Theta(\log \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(\log^2 n)$ ☒ $\Theta(\sqrt{n})$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^4)$ ☐ $\Theta(n \log n)$
☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2 \log^2 n)$ ☐ $\Theta(n^{\log_2 3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\sqrt{n^3})$ ☐ $\Theta(\sqrt{\log n})$
☐ $\Theta(\sqrt{n \log n})$ ☐ $\Theta(\sqrt[3]{2^n})$ ☐ $\Theta(2^n)$ ☐ $\Theta(3^n)$ ☐ $\Theta(n!)$ ☐ $\Theta(n^n)$ ☐ N/A

Die richtige Antwort ist: $\Theta(\sqrt{n})$

(1.2) $\sum_{i=1}^n 2^{n-i}$

- ☐ $\Theta(1)$ ☐ $\Theta(\log \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(\log^2 n)$ ☒ $\Theta(2^n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^4)$ ☐ $\Theta(n \log n)$
☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2 \log^2 n)$ ☐ $\Theta(n^{\log_2 3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\sqrt{n^3})$ ☐ $\Theta(\sqrt{\log n})$
☐ $\Theta(\sqrt{n \log n})$ ☐ $\Theta(\sqrt[3]{2^n})$ ☐ $\Theta(2^n)$ ☐ $\Theta(3^n)$ ☐ $\Theta(n!)$ ☐ $\Theta(n^n)$ ☐ N/A

Die richtige Antwort ist: $\Theta(2^n)$

(1.3) $\frac{n^5 - n^2}{n^2 + n}$

- ☐ $\Theta(1)$ ☐ $\Theta(\log \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(\log^2 n)$ ☒ $\Theta(n^3)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^4)$ ☐ $\Theta(n \log n)$
☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2 \log^2 n)$ ☐ $\Theta(n^{\log_2 3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\sqrt{n^3})$ ☐ $\Theta(\sqrt{\log n})$
☐ $\Theta(\sqrt{n \log n})$ ☐ $\Theta(\sqrt[3]{2^n})$ ☐ $\Theta(2^n)$ ☐ $\Theta(3^n)$ ☐ $\Theta(n!)$ ☐ $\Theta(n^n)$ ☐ N/A

Die richtige Antwort ist: $\Theta(n^3)$

Runtime Analysis

(1.4) 4613^{976}

- ☐ $\Theta(1)$ ☐ $\Theta(\log \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(\log^2 n)$ ☒ $\Theta(1)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^4)$ ☐ $\Theta(n \log n)$
☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2 \log^2 n)$ ☐ $\Theta(n^{\log_2 3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\sqrt{n^3})$ ☐ $\Theta(\sqrt{\log n})$
☐ $\Theta(\sqrt{n \log n})$ ☐ $\Theta(\sqrt[3]{2^n})$ ☐ $\Theta(2^n)$ ☐ $\Theta(3^n)$ ☐ $\Theta(n!)$ ☐ $\Theta(n^n)$ ☐ N/A

Die richtige Antwort ist: $\Theta(1)$

(1.5) $n \sum_{i=1}^{\log n} i$

- ☐ $\Theta(1)$ ☐ $\Theta(\log \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(\log^2 n)$ ☒ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^4)$ ☐ $\Theta(n \log n)$
☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n \log^2 n)$ ☐ $\Theta(n^2 \log^2 n)$ ☐ $\Theta(n^{\log_2 3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\sqrt{n^3})$ ☐ $\Theta(\sqrt{\log n})$
☐ $\Theta(\sqrt{n \log n})$ ☐ $\Theta(\sqrt[3]{2^n})$ ☐ $\Theta(2^n)$ ☐ $\Theta(3^n)$ ☐ $\Theta(n!)$ ☐ $\Theta(n^n)$ ☐ N/A

Die richtige Antwort ist: $\Theta(n \log^2 n)$

5. Hausaufgaben

Übung 5: Algorithmen und Effizienz

Auf <https://expert.ethz.ch/enrolled/SS25/mavt2/exercises>

Übung 5: Algorithmen und Effizienz

Einfach, Mittel, Schwer

- Asymptotische Laufzeit, Auf ZF schauen!
- Längstes gemeinsames Präfix, Recursion??
- Dutch Flag (Die niederländische Flagge), Probieren (Video schauen?)
- k-kleinstes Element, Recursion??

Abgabedatum: Montag 31.03.2024, 20:00 CET

KEINE HARDCODIERUNG

Fragen?

Feedback



<https://n.ethz.ch/~jschul/Feedback>

6. Herleitungen

Einige Formeln mit Herleitung

Summen

$$\sum_{i=0}^n i = ?$$

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

Summen

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

Warum?

Summen

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

Warum?
Intuition

$$1 + \dots + 100 = (1 + 100) + (2 + 99) + (3 + 98) + \dots + (50 + 51)$$

Summen

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

Warum?
Intuition

$$1 + \dots + 100 = (1 + 100) + (2 + 99) + (3 + 98) + \dots + (50 + 51)$$

Formaler?

$$\sum_{i=0}^n (n - i) = ?$$

$$\sum_{i=0}^n (n - i) = \sum_{i=0}^n i$$

Summen

$$\sum_{i=0}^n (n - i) = \sum_{i=0}^n i$$

$$\begin{aligned}\Rightarrow 2 \cdot \sum_{i=0}^n i &= \sum_{i=0}^n i + \sum_{i=0}^n (n - i) \\ &= \sum_{i=0}^n (i + (n - i)) = \sum_{i=0}^n n = (n + 1) \cdot n\end{aligned}$$

$$\sum_{i=0}^n (n - i) = \sum_{i=0}^n i$$

$$\begin{aligned}\Rightarrow 2 \cdot \sum_{i=0}^n i &= \sum_{i=0}^n i + \sum_{i=0}^n (n - i) \\ &= \sum_{i=0}^n (i + (n - i)) = \sum_{i=0}^n n = (n + 1) \cdot n\end{aligned}$$

$$\sum_{i=0}^n i^2 = ?$$

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Das muss man nicht auswendig wissen. Aber man sollte wissen, dass es ein Polynom dritten Grades in n ist

Summen

Wie kommt man darauf?

Summen

Wie kommt man darauf? Interessanter Trick: Einerseits

$$\sum_{i=0}^n i^3 - \sum_{i=1}^n (i-1)^3 = \sum_{i=0}^n i^3 - \sum_{i=0}^{n-1} i^3 = n^3,$$

Summen

Wie kommt man darauf? Interessanter Trick: Einerseits

$$\sum_{i=0}^n i^3 - \sum_{i=1}^n (i-1)^3 = \sum_{i=0}^n i^3 - \sum_{i=0}^{n-1} i^3 = n^3,$$

andererseits

$$\begin{aligned} \sum_{i=0}^n i^3 - \sum_{i=1}^n (i-1)^3 &= \sum_{i=1}^n i^3 - \sum_{i=1}^n (i-1)^3 \\ &= \sum_{i=1}^n i^3 - (i-1)^3 = \sum_{i=1}^n 3 \cdot i^2 - 3 \cdot i + 1 \end{aligned}$$