Informatik II Woche 6

Bubble Sort, Insertion Sort, Asymptotik

Website: n.ethz.ch/~kvaratharaja/

Die Slides basieren auf den offiziellen Übungsslides der Kurswebsite: https://lec.inf.ethz.ch/mavt/informatik2/2025/

Heute

1. **Bubble Sort**
2. **Insertion Sort**
3. **Asymptotik**
4. **Inclass-Exercise**
5. **Hausaufgaben**

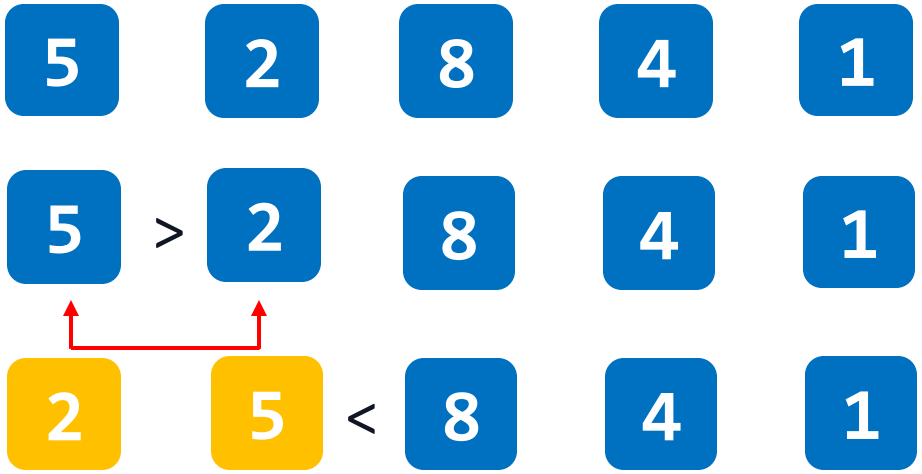
1. Bubble Sort

Bubble Sort

Bubble Sort

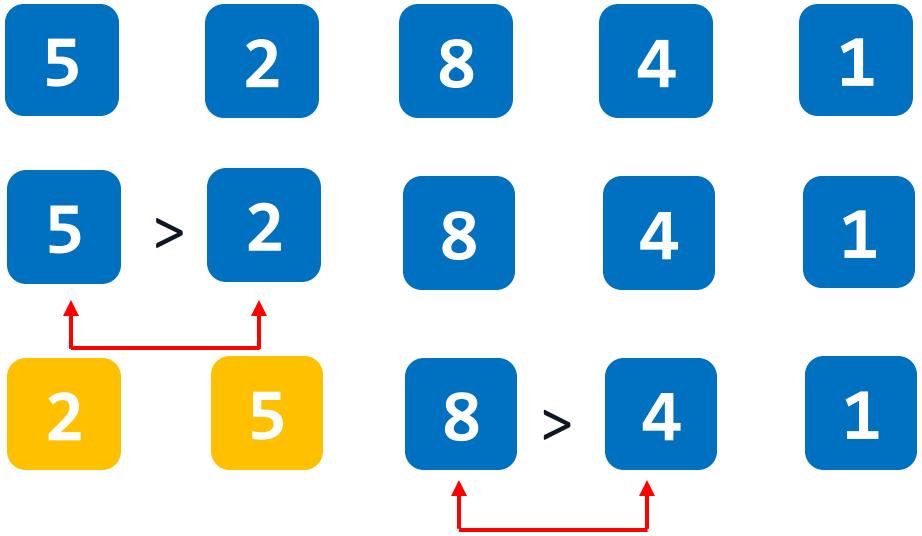
- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubble Sort



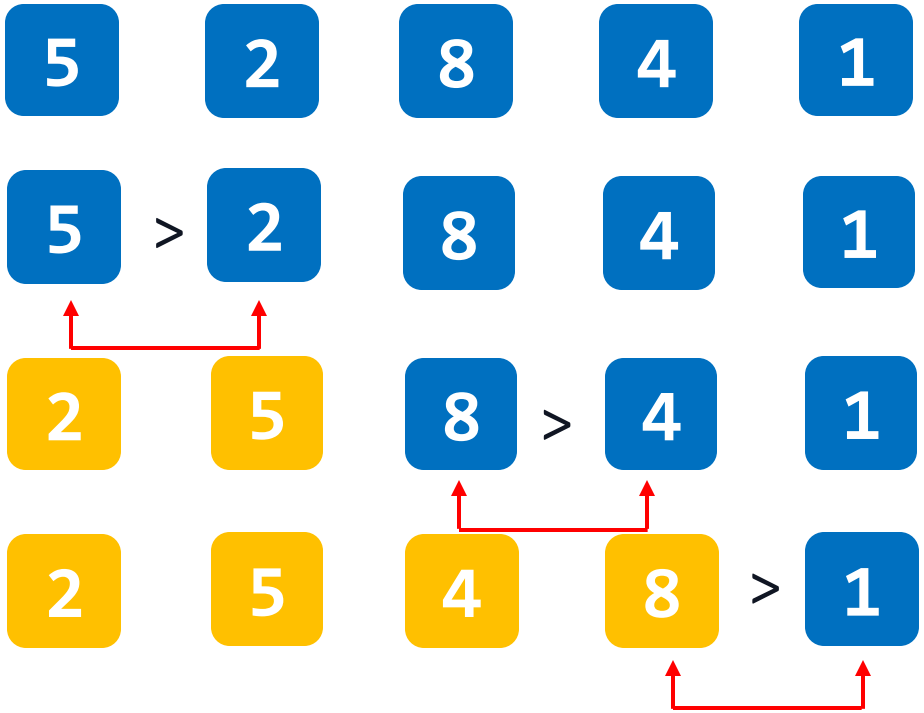
- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubble Sort



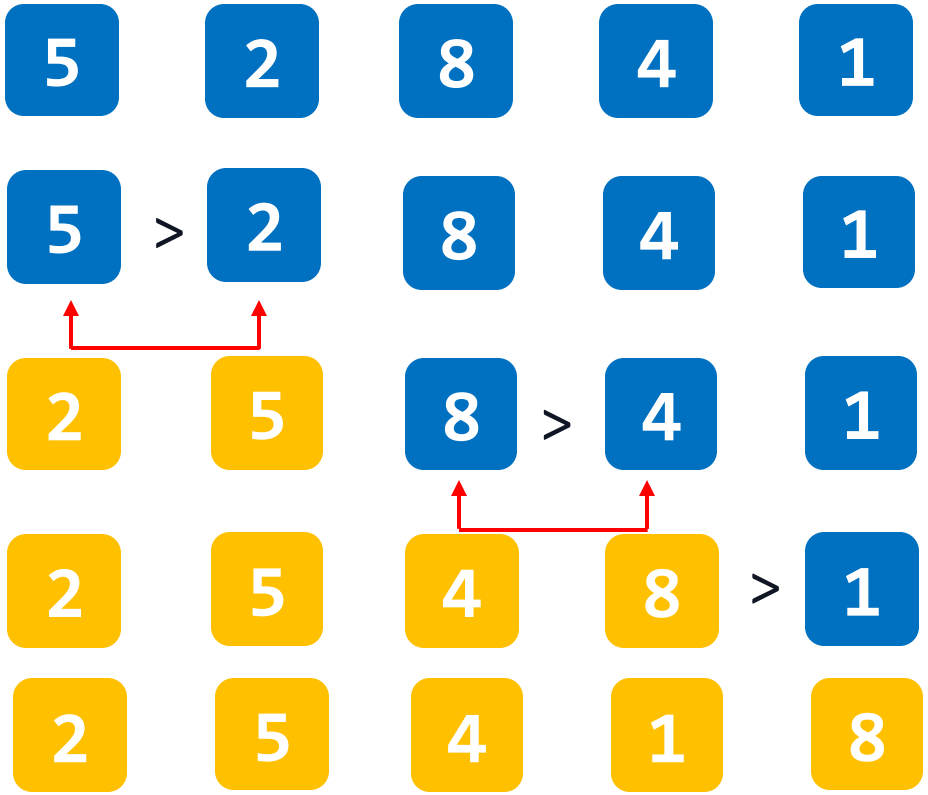
- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubble Sort



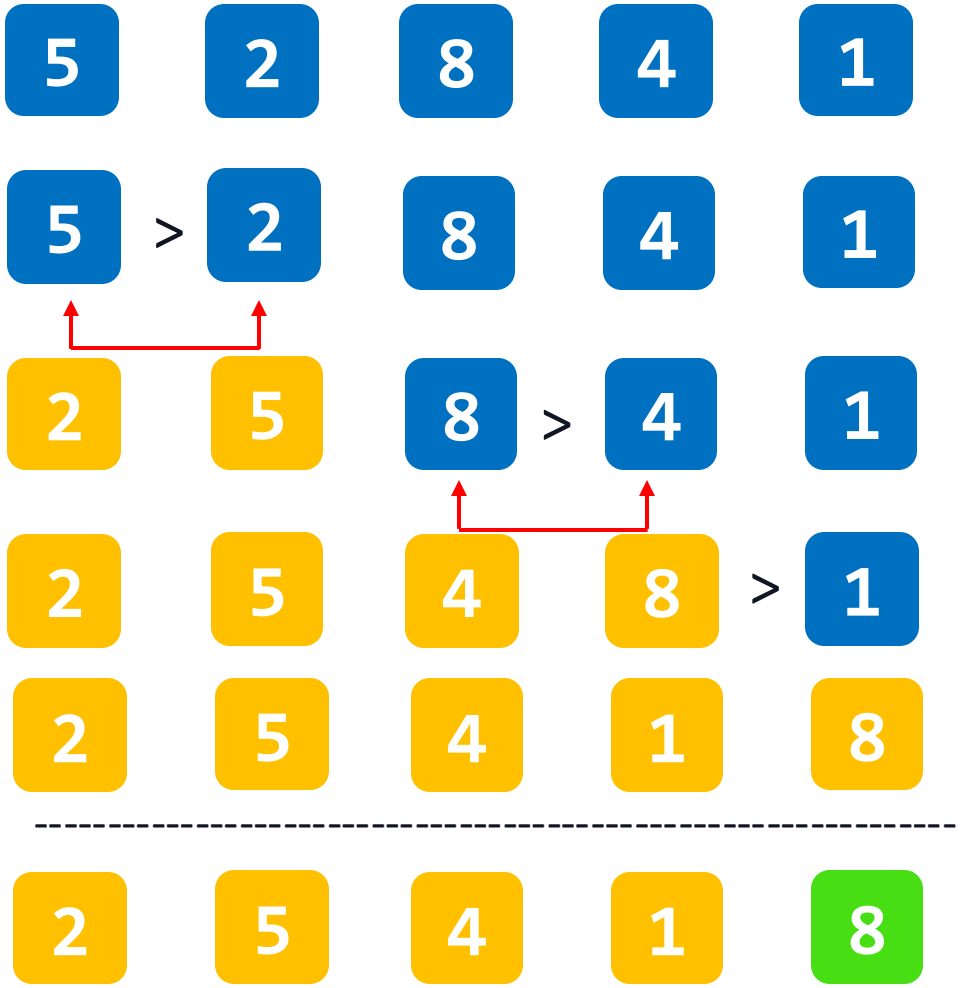
- Vergleiche jedes Paar benachbarter Elemente der Reihe nach. Wenn das erste größer ist, tausche sie aus.

Bubble Sort



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

Bubble Sort



- 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.

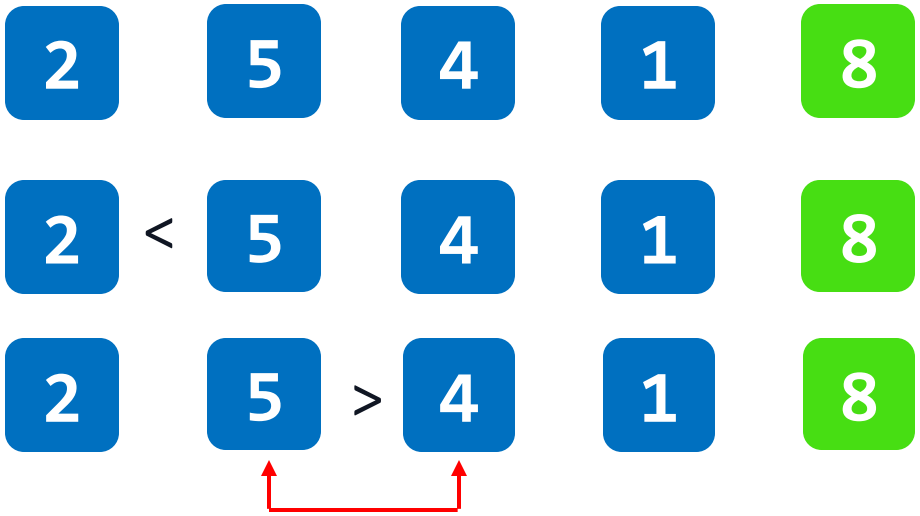
Bubble Sort

- Schleife, bis die Liste sortiert ist.

Bubble Sort

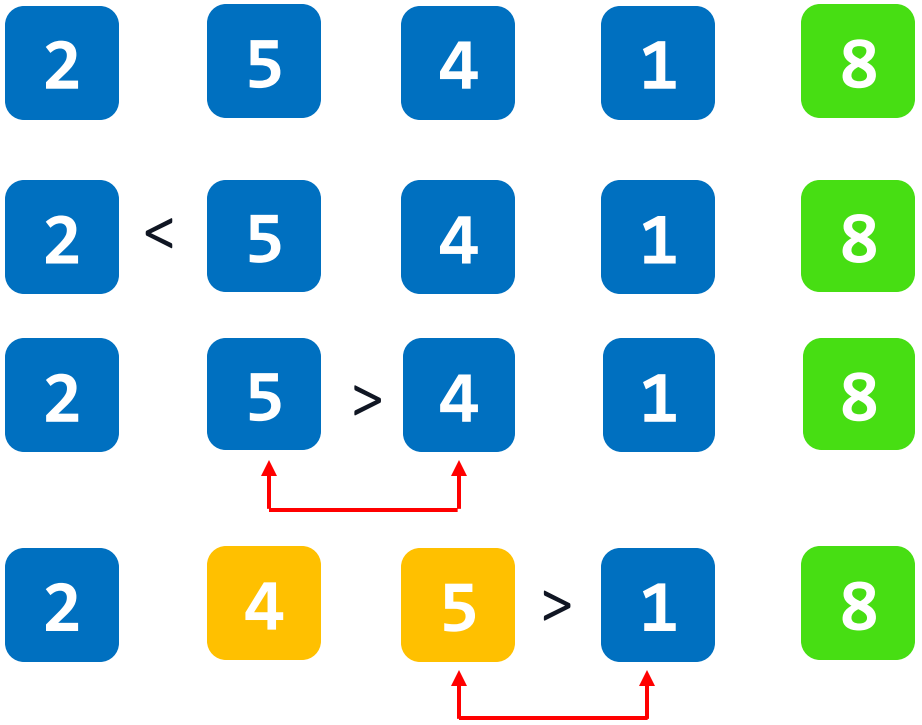
- Schleife, bis die Liste sortiert ist.

Bubble Sort



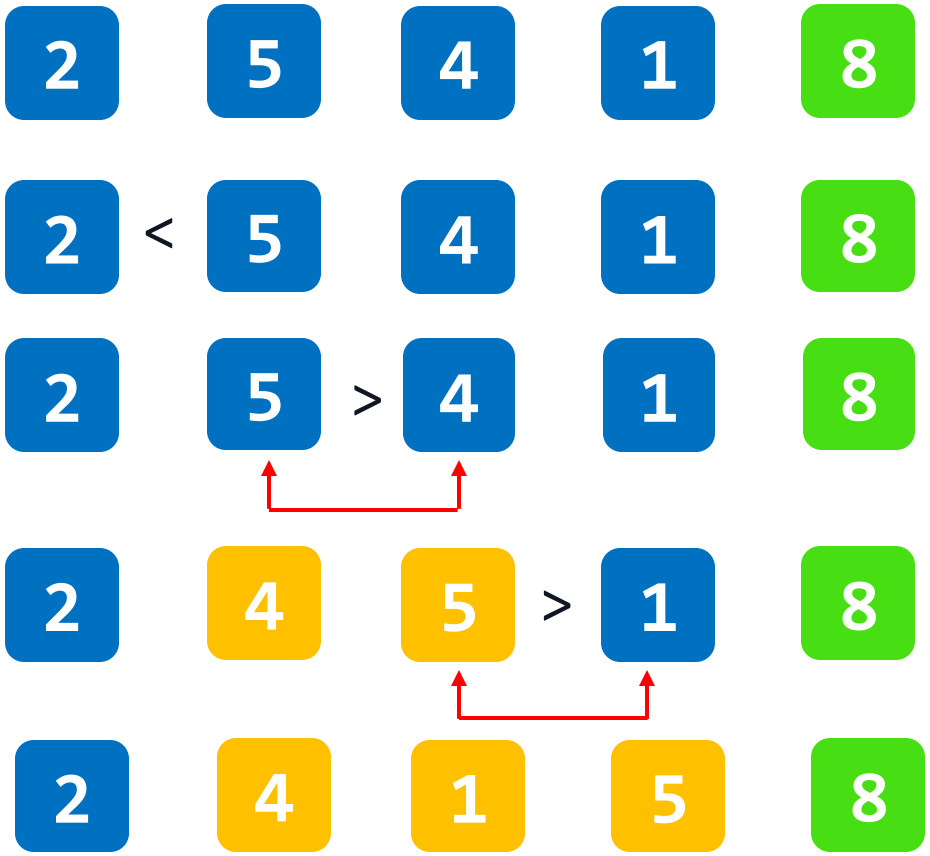
- Schleife, bis die Liste sortiert ist.

Bubble Sort



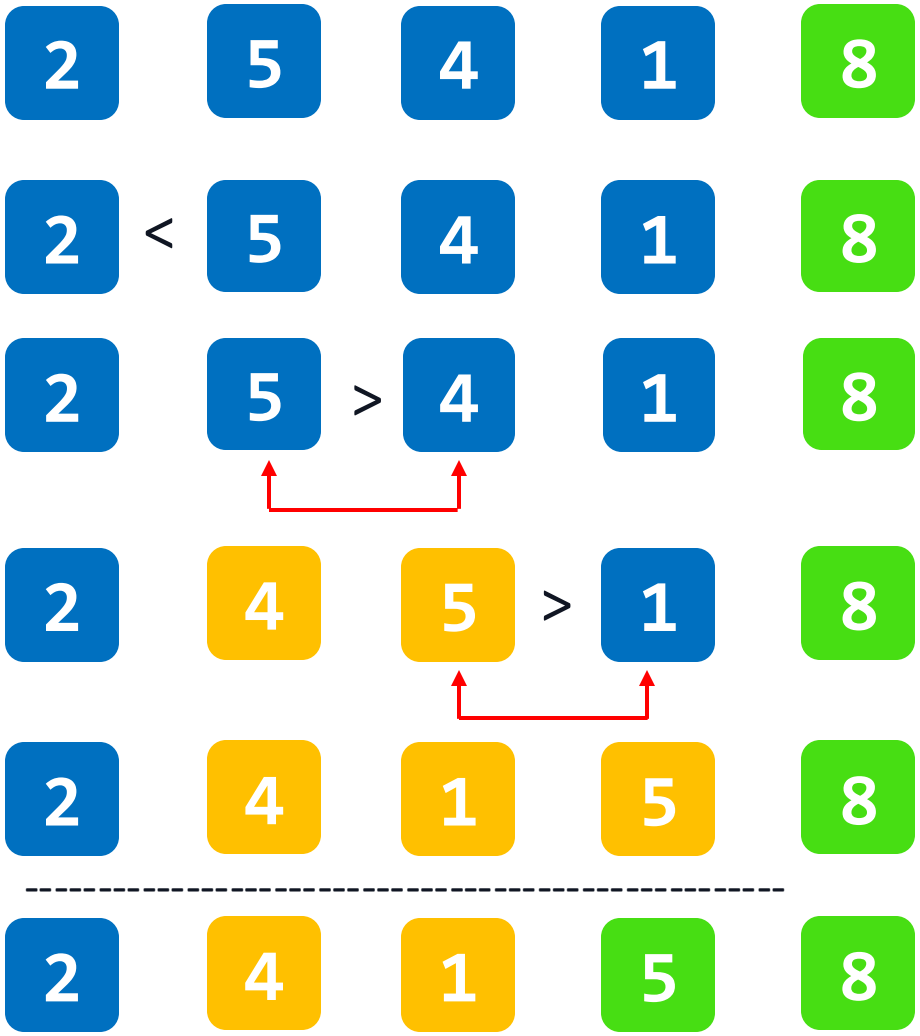
- Schleife, bis die Liste sortiert ist.

Bubble Sort



- Schleife, bis die Liste sortiert ist.

Bubble Sort



- Schleife, bis die Liste sortiert ist.
- **Schleifeninvariante**:

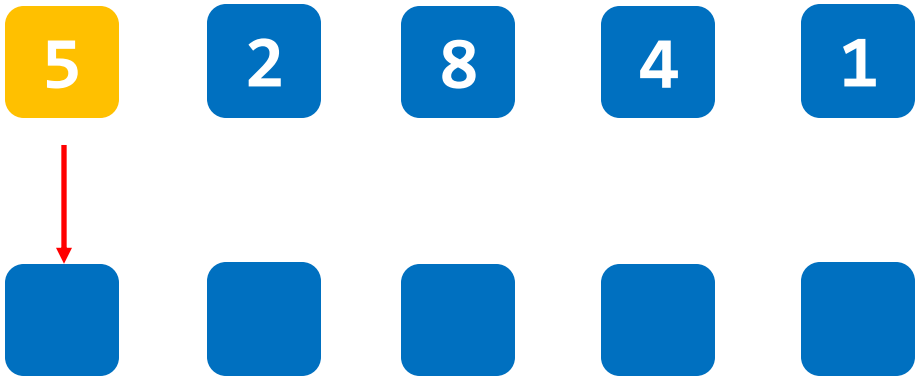
Nach Iteration i sind Elemente li[-(i+1):] sortiert und an der richtigen Stelle.

2. Insertion Sort

Insertion Sort: Konzept

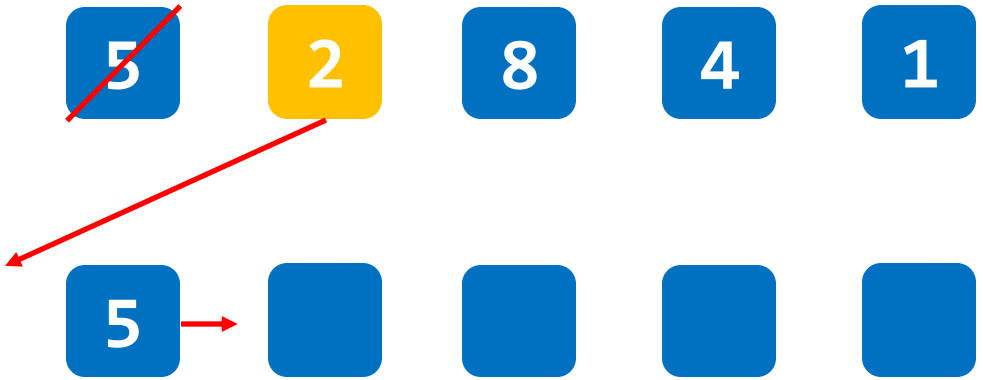
Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



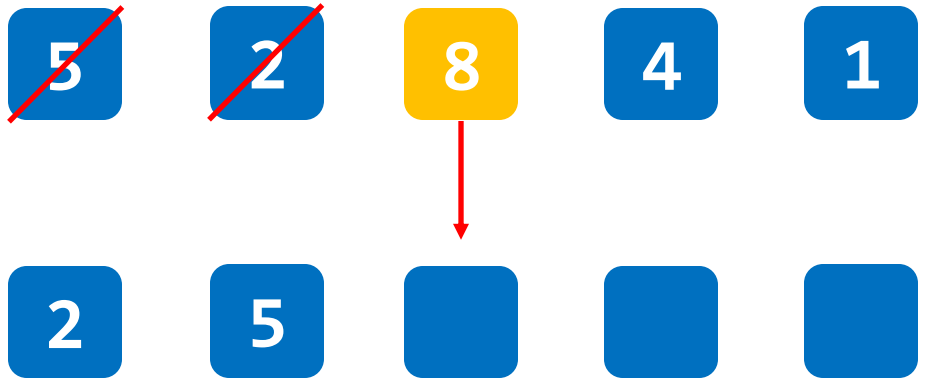
Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



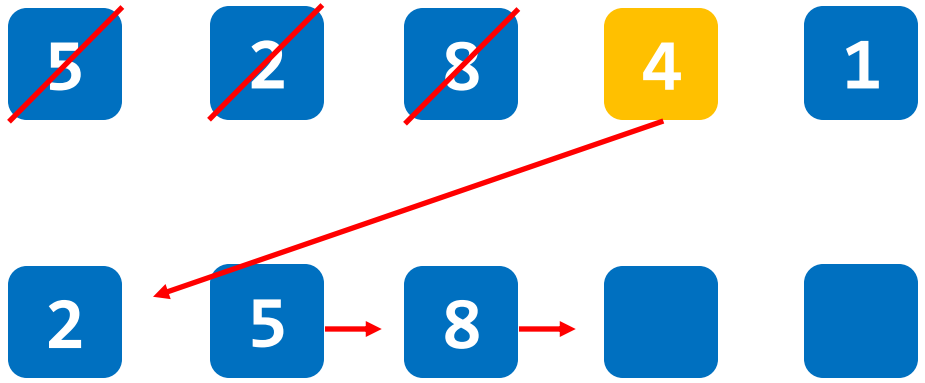
Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



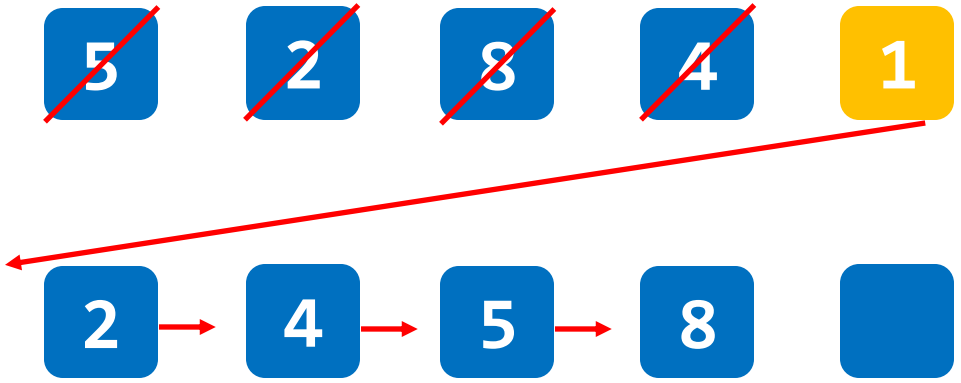
Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept



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

Insertion Sort: Konzept



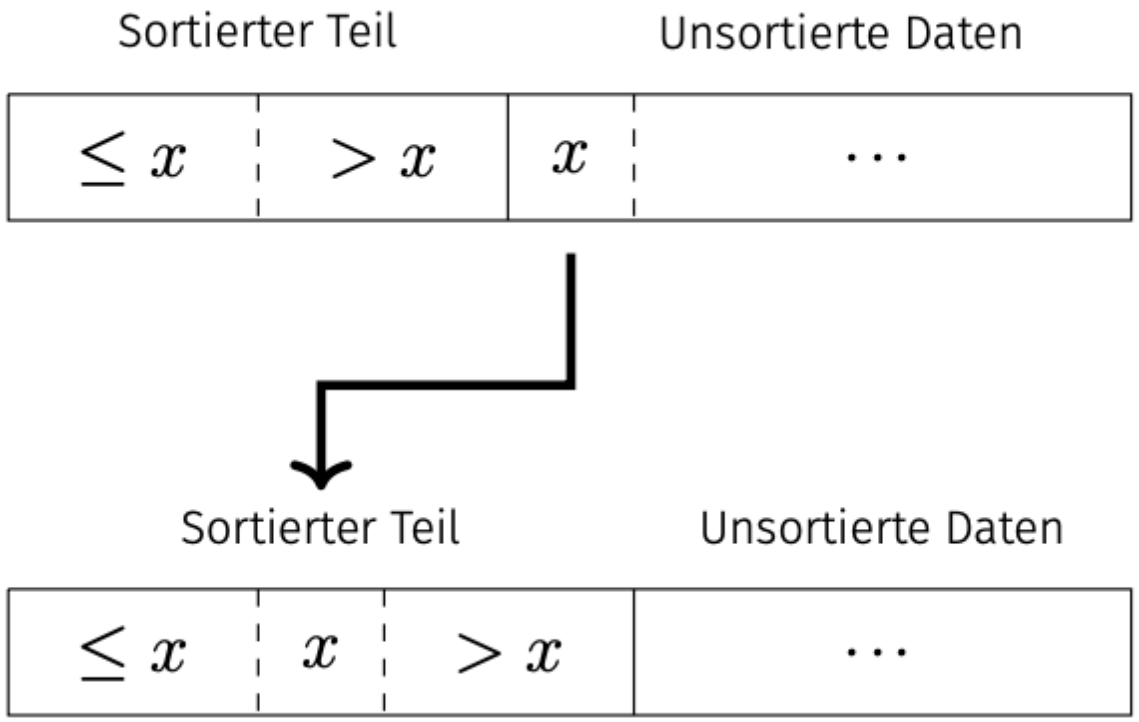
Mit einer zweiten Liste können wir einfach jedes Element an der korrekten Stelle einsortieren.

Insertion Sort: Konzept

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

PROBLEM: Benötigt n zusätzlich Speicherplatz

Insertion Sort



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

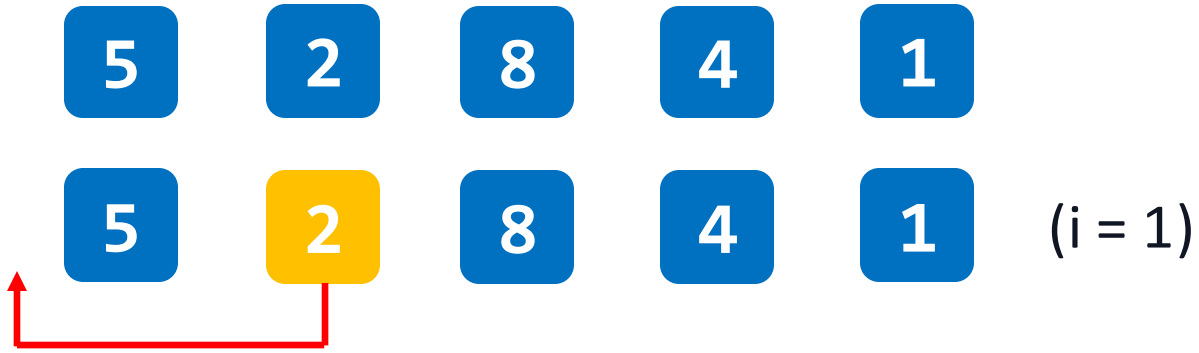
Insertion Sort

Insertion Sort

(i = 1)

Schleifeninvariante: ??

Insertion Sort



- **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.

Insertion Sort

- **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.
- Wiederhole bis alles sortiert ist

  (i = n)

Insertion Sort

- **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.
- Wiederhole bis alles sortiert ist

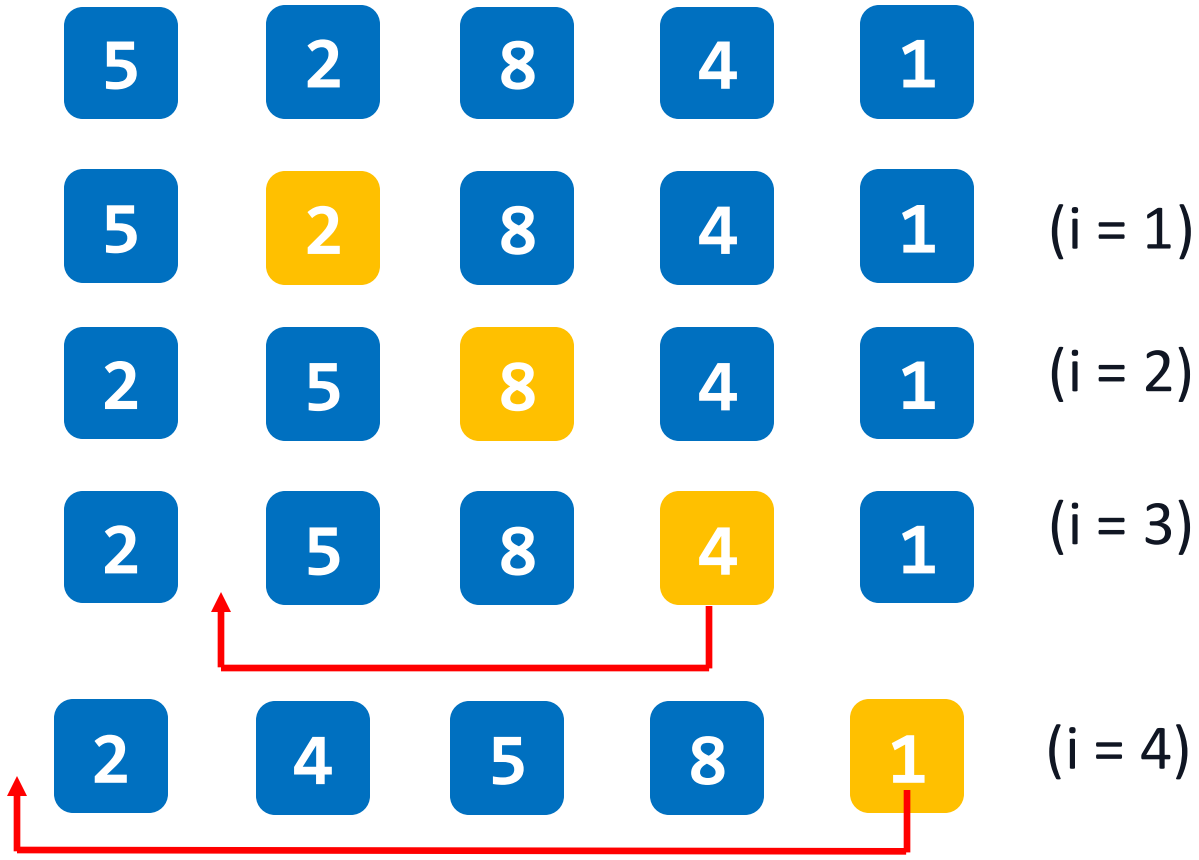
  (i = n)

Insertion Sort

- **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.
- Wiederhole bis alles sortiert ist

  (i = n)

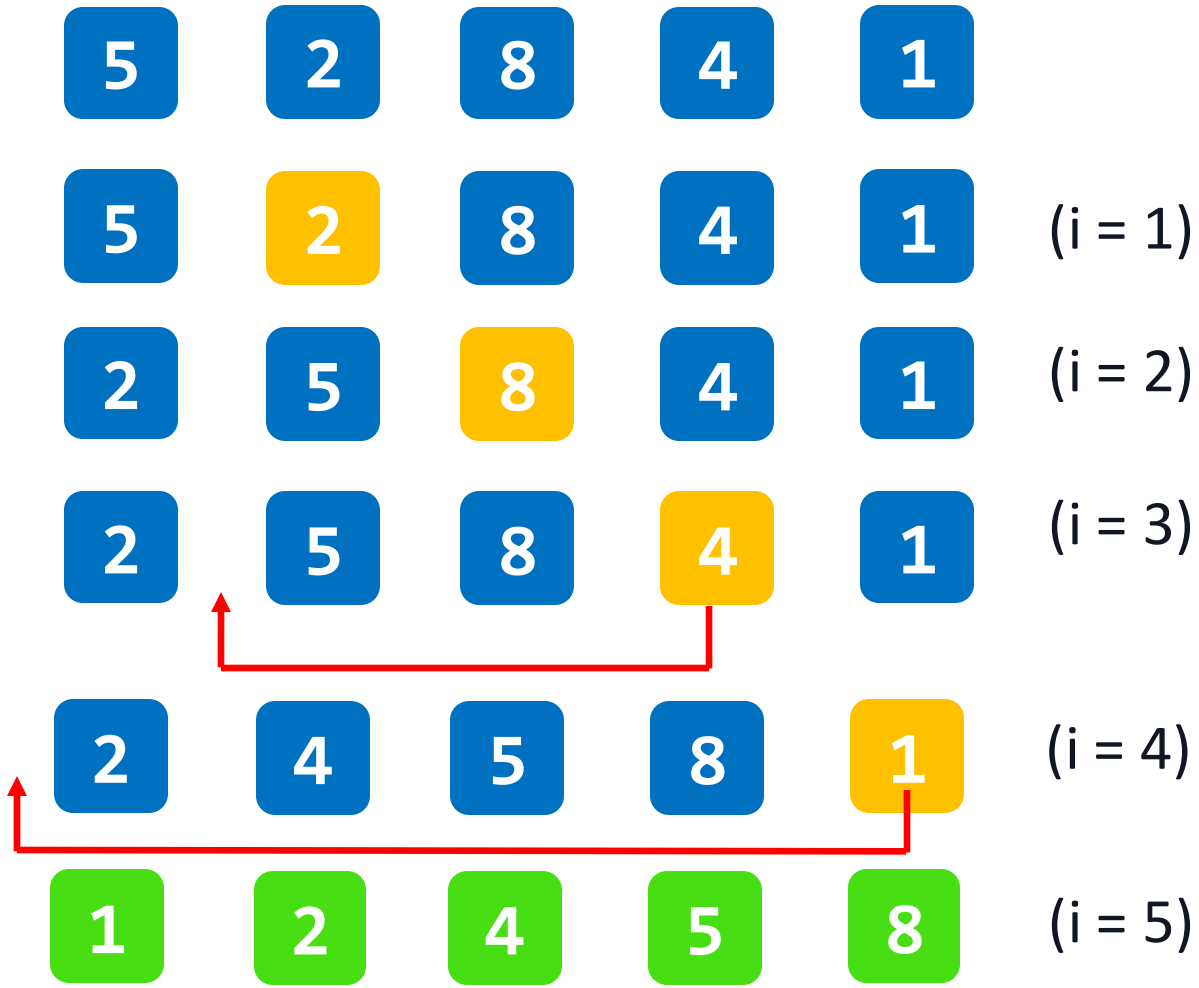
Insertion Sort



- **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.
- Wiederhole bis alles sortiert ist

  (i = n)

Insertion Sort



- **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.
- Wiederhole bis alles sortiert ist

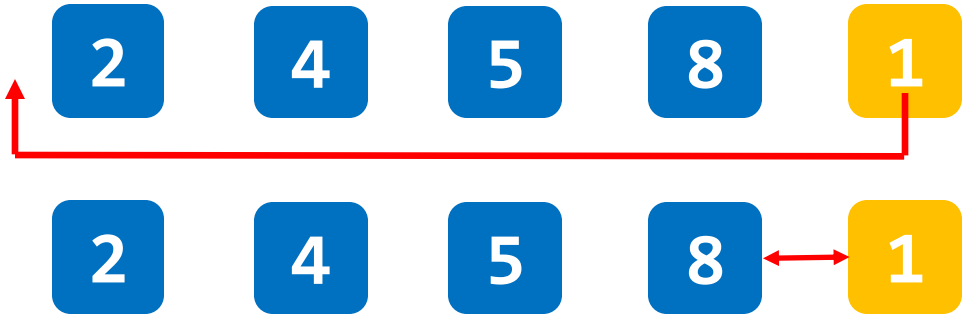
  (i = n)
- **Frage**: Wie kann man die Insertion durchführen?

Einfügen durch Austauschen

(j = 4)

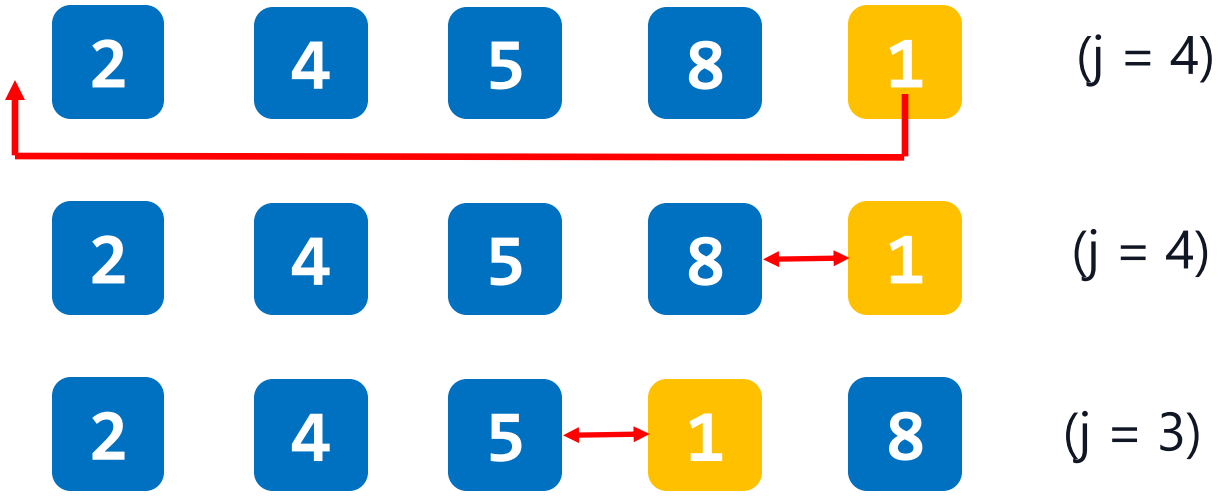
- Betrachten wir Iteration i = 4
- Setze Variable j = i

Einfügen durch Austauschen



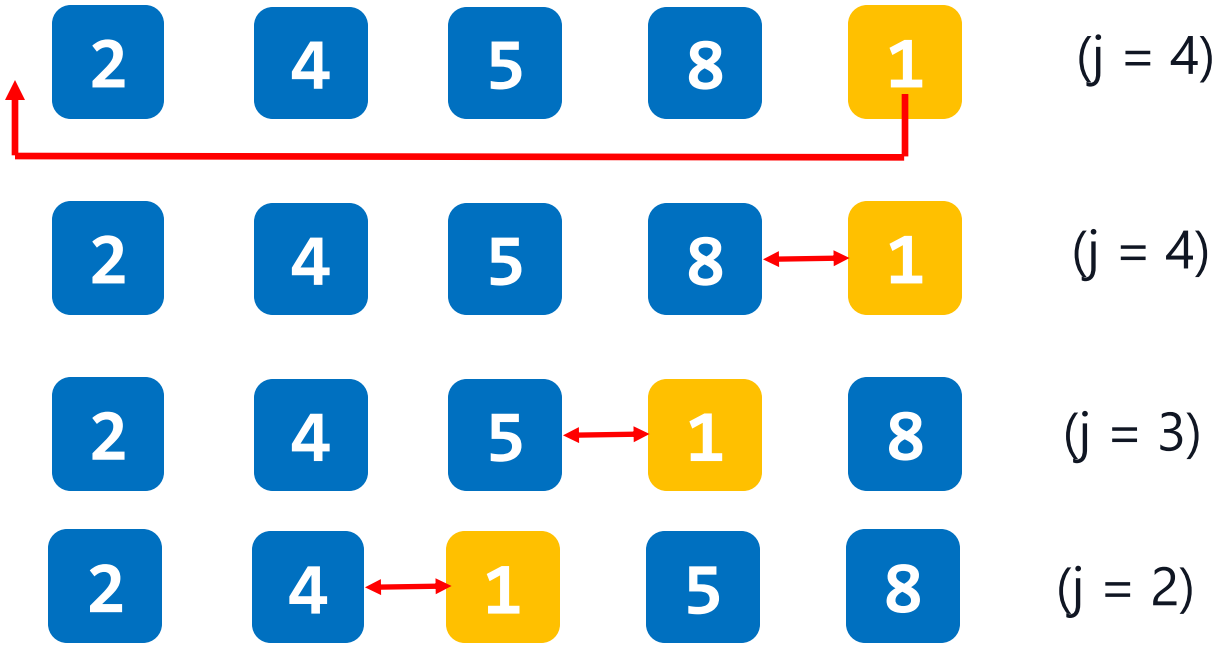
- Betrachten wir Iteration i = 4
- Setze Variable j = i
- Vergleiche li[j] und li[j-1] und tausche, falls li[j-1] > li[j].

Einfügen durch Austauschen



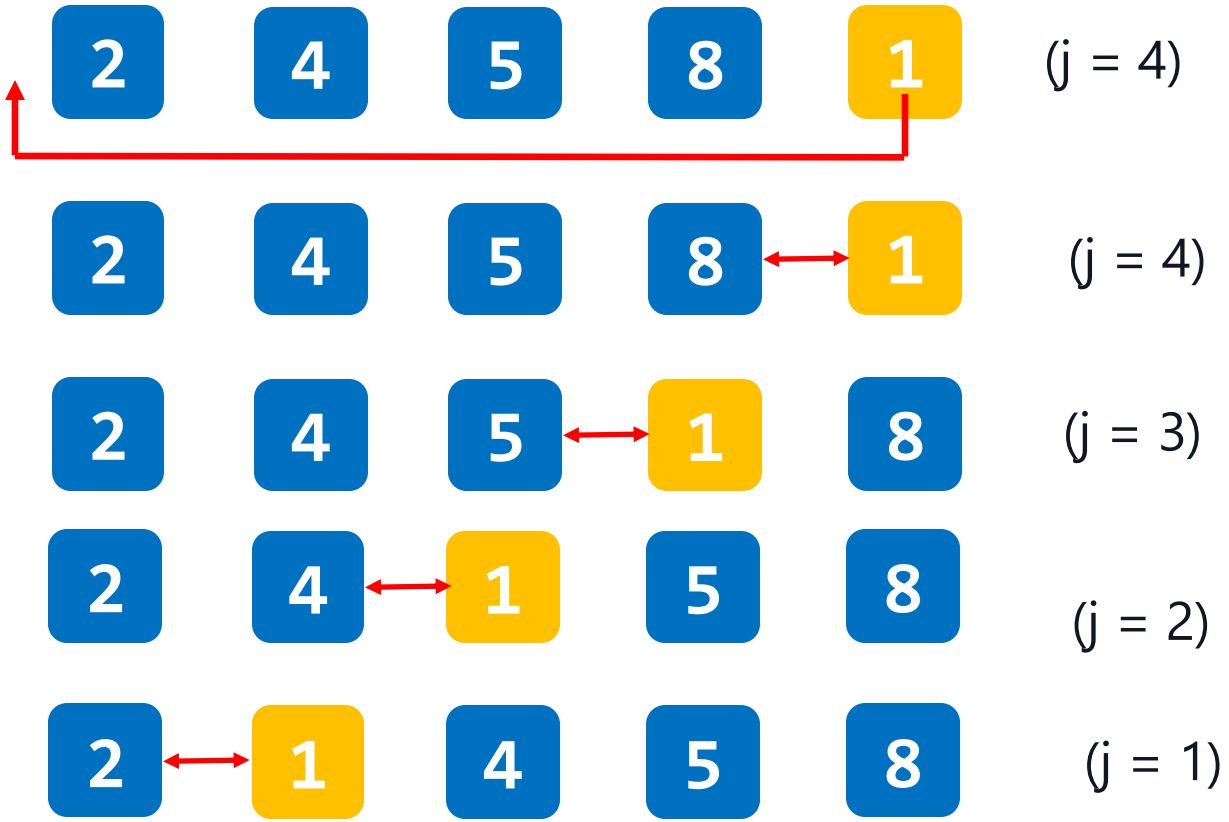
- Betrachten wir Iteration i = 4
- Setze Variable j = i
- Vergleiche li[j] und li[j-1] und tausche, falls li[j-1] > li[j].
- Wiederhole bis j = 0 oder li[j-1] <= li[j]

Einfügen durch Austauschen



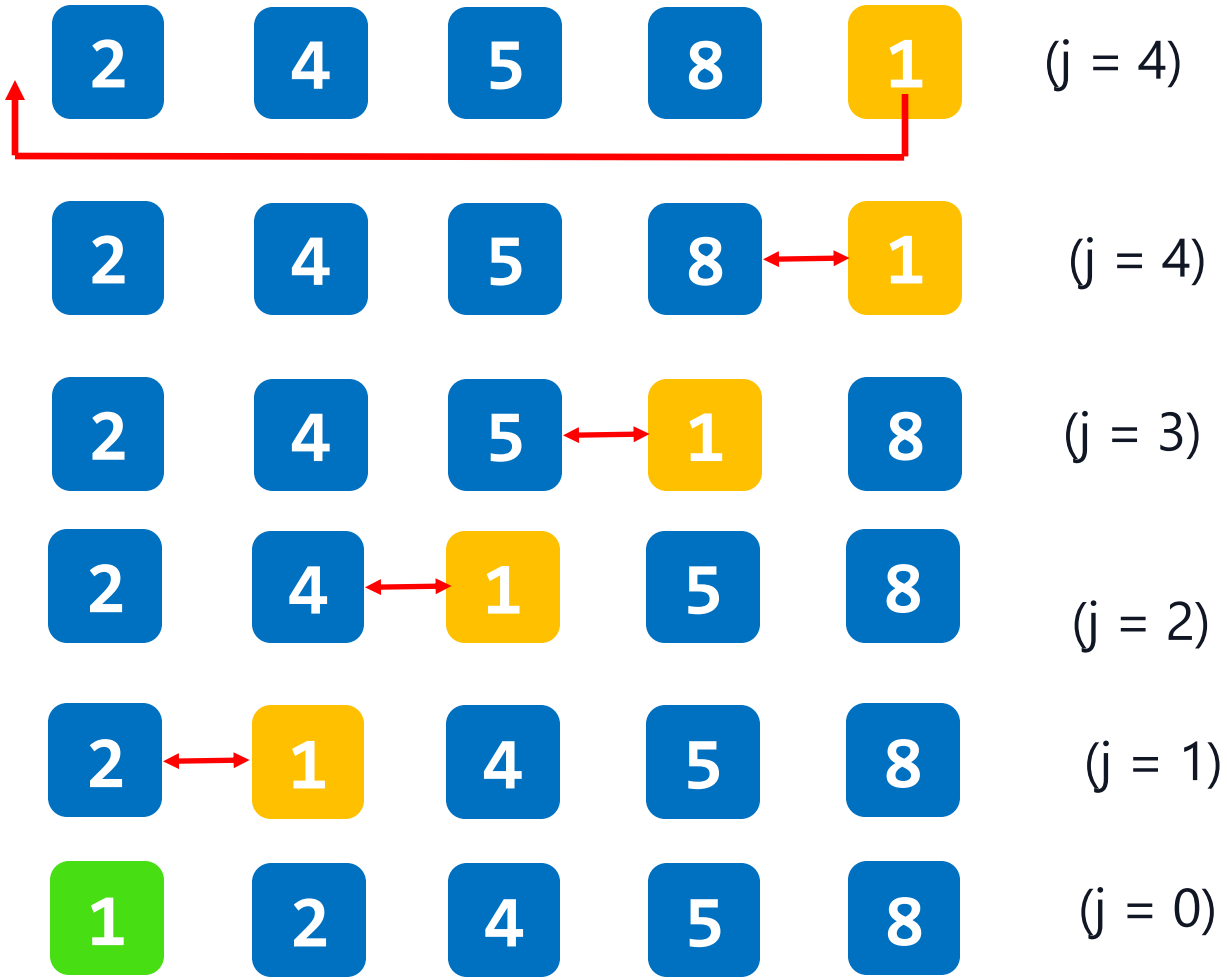
- Betrachten wir Iteration i = 4
- Setze Variable j = i
- Vergleiche li[j] und li[j-1] und tausche, falls li[j-1] > li[j].
- Wiederhole bis j = 0 oder li[j-1] <= li[j]

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 oder li[j-1] <= li[j]

Einfügen durch Austauschen



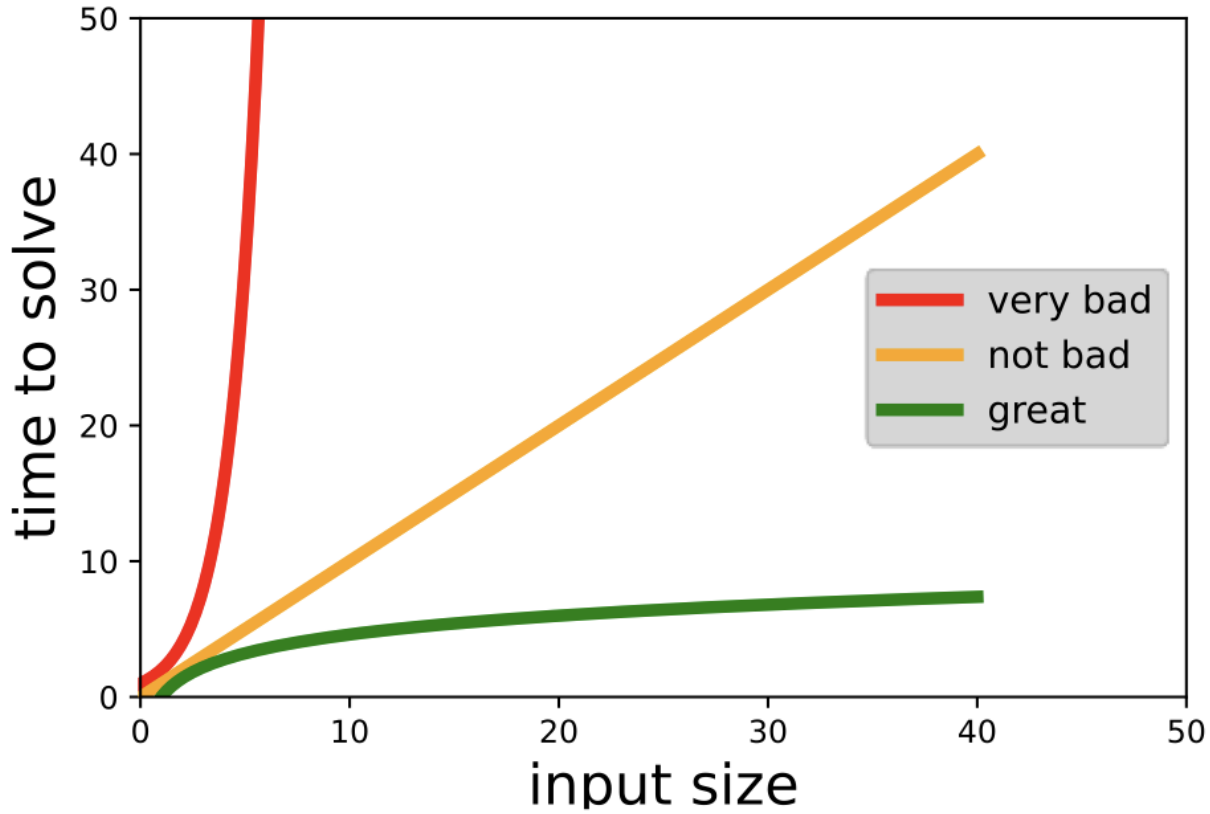
- Betrachten wir Iteration i = 4
- Setze Variable j = i
- Vergleiche li[j] und li[j-1] und tausche, falls li[j-1] > li[j].
- Wiederhole bis j = 0 oder li[j-1] <= li[j]

3. Asymptotisches Verhalten & Laufzeit

(kommt sehr wahrscheinlich an der Prüfung) 

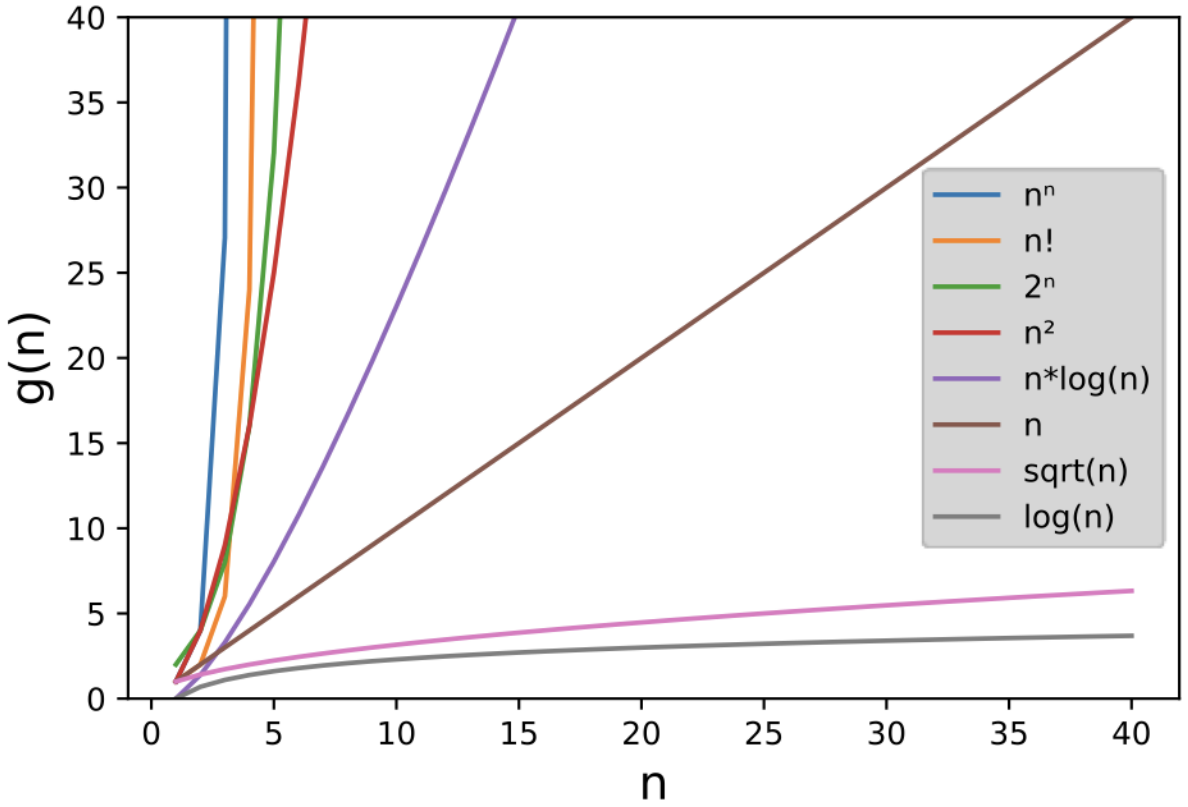
Was ist ein guter Algorithmus?

- Gutes Verhältnis von Eingabegrösse vs Rechenzeit



Idee: Algorithmus vergleichen

- Wir vergleichen die Rechenzeit unseres Algorthmus mit einer Funktion g(n)



Asymptotisches Verhalten: Notation

Was sind $O(g)$, $\Omega(g)$, $\Theta(g)$?

→ Funktionen!

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

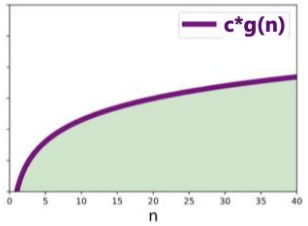
Aber was bedeutet das intuitiv gesehen? 🤔

Asymptotisches Verhalten: Intuition

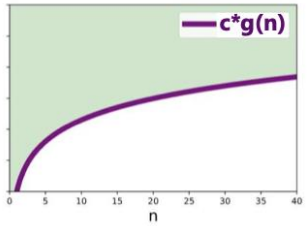
Also was sind $O(g), \Omega(g)$ oder $\Theta(g)$? → Asymptotische Ober- und Untergrenzen!

"*der Graph von f bleibt immer im grünen Bereich ab einem Wert n*"

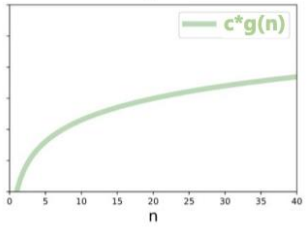
$f \in O(g)$: f wächst **nicht schneller** als $c \cdot g(n)$.



$f \in \Omega(g)$: f wächst **nicht langsamer** als $c \cdot g(n)$.

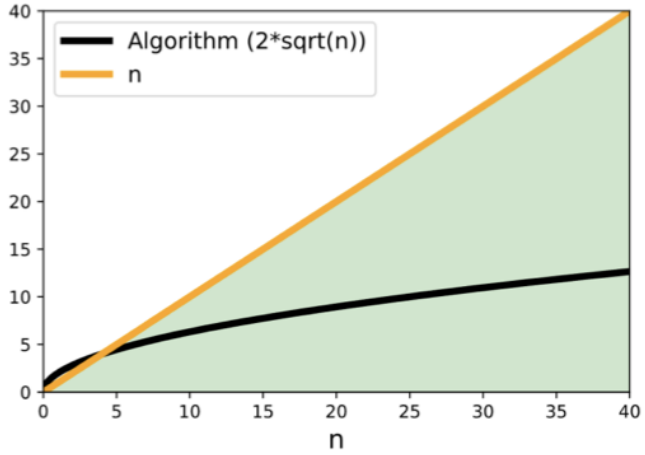


$f \in \Theta(g)$: f wächst **etwa gleich** wie $c \cdot g(n)$.

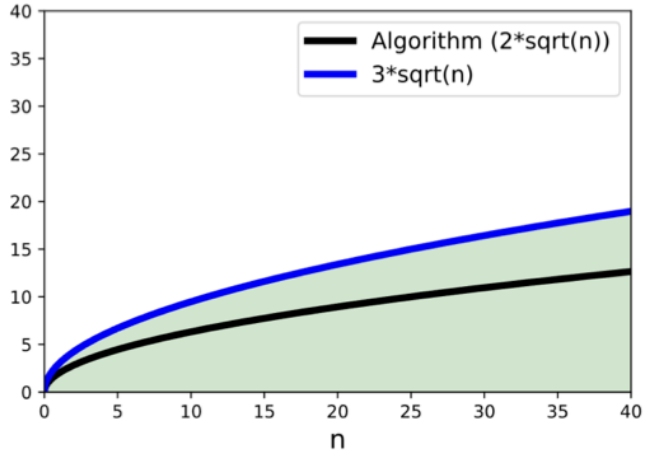


Asymptotisches Verhalten – Beispiel $O(g)$

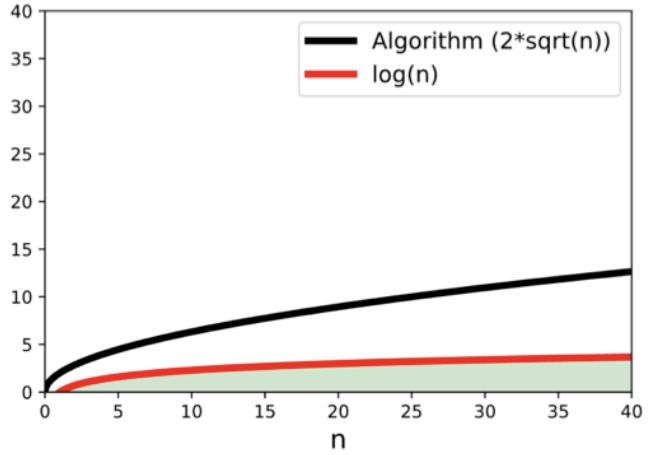
Laufzeit vom Algorithmus ist $f = 2 * \sqrt{n}$



$O(n)$ ✅



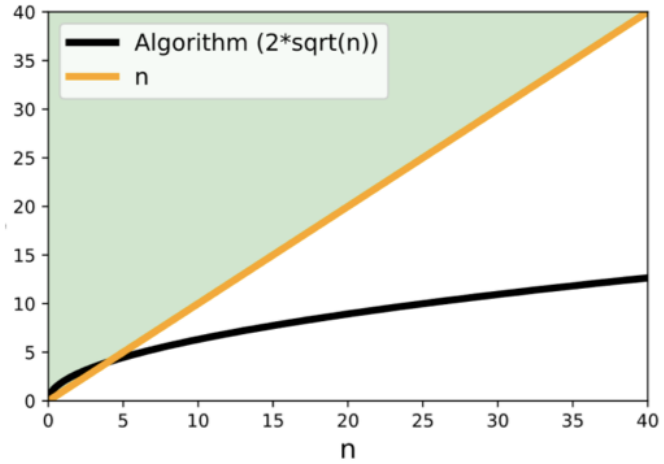
$O(\sqrt{n})$ ✅



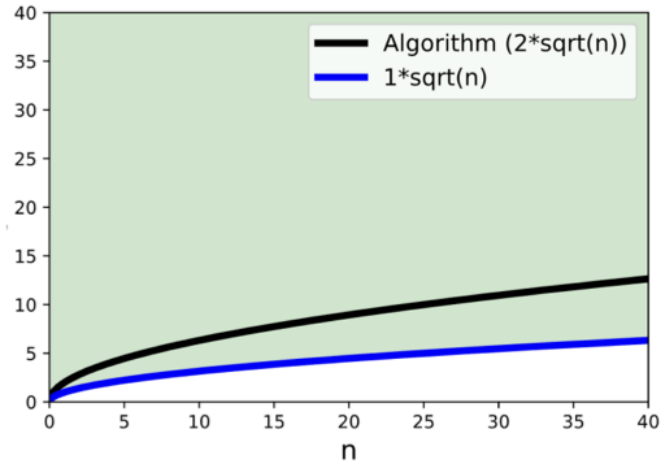
$O(\log(n))$ ❌

Asymptotisches Verhalten – Beispiel Ω(g)

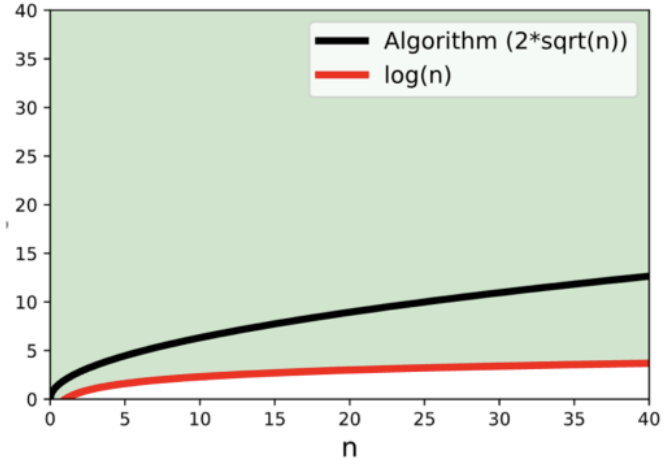
Laufzeit vom Algorithmus ist $f = 2 * \sqrt{n}$



$\Omega(n)$❌



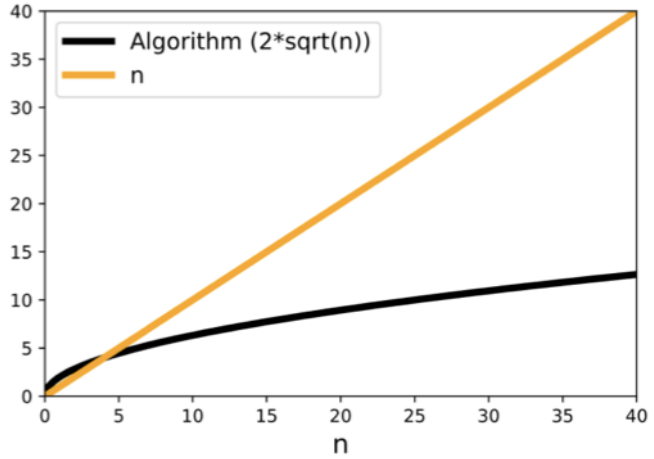
$\Omega(\sqrt{n})$✅



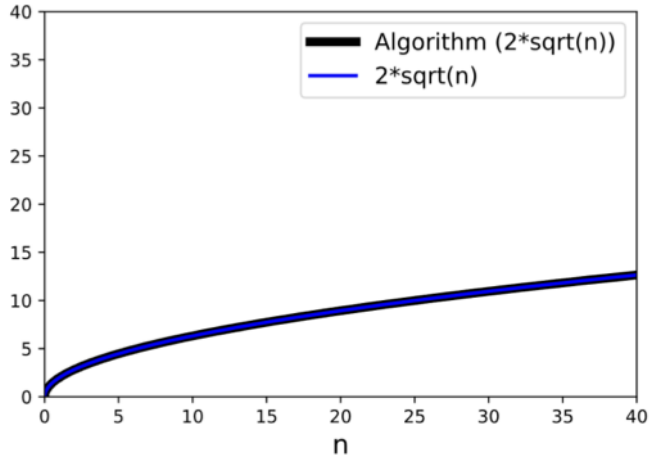
$\Omega(\log(n))$✅

Asymptotisches Verhalten – Beispiel $\Theta(g)$

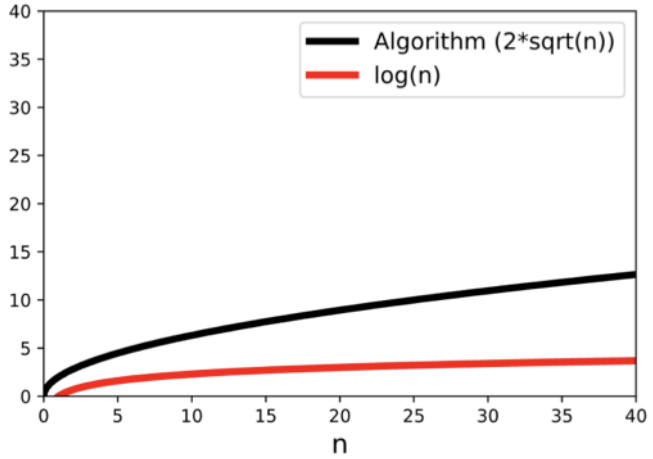
Laufzeit vom Algorithmus ist $f = 2 * \sqrt{n}$



$\Theta(n)$ ❌



$\Theta(\sqrt{n})$ ✅



$\Theta(\log(n))$ ❌

Asymptotisches Verhalten: Nützliche Formeln

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

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

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

(muss man nicht auswendig wissen)

In-Class Exercise: Laufzeiten mit Θ

- Entweder auf Code Expert: **Nicht-rekursive Snippets Laufzeiten**

https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples

- Wer per Hand mitschreiben will: separate Datei in meiner Polybox unter 'Zusatzmaterial'

4. In-Class Exercise

In-class Exercise: Insertion Sort Implementierung

CodeExpert in-class task:

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?

5. Hausaufgaben

Exercise 5: Algorithms and Efficiency

Auf https://expert.ethz.ch/mycourses/SS25/mavt2/exercises

- Asymptotische Laufzeit
- Längstes gemeinsames Präfix
- Dutch flag
- k-kleinstes Element

Fällig bis Montag 31.03.2025, 20:00 CET

NO HARDCODING

Feedback?

Zu schnell? Zu langsam? Weniger Theorie, mehr Aufgaben?
Dankbar für Feedback am besten mir direkt sagen oder Mail schreiben

Credits

Die Slide(-templates) stammen ursprünglich von Julian Lotzer und Daniel Steinhauser, vielen Dank!

→Checkt ihre Websites ab für zusätzliches Material in Informatik I, Informatik II und Stochastik & Machine Learning.

- https://n.ethz.ch/~jlotzer/
- https://n.ethz.ch/~dsteinhauser/