Westfälische Hochschule

Grundlagen der Informatik und Programmierung 1

Sortieren

Ordne das Chaos.

Prof. Dr. Tom Vierjahn

Fachbereich Wirtschaft und Informationstechnik
Westfälische Hochschule – Campus Bocholt

Wintersemester 2019/20

Lineare Suche

in unsortierten Daten

Daten:

finde die 13:

- betrachte das erste Element
- ist es die 13?
  - ja: fertig
- wiederhole mit nächstem Element

Aufwand:

- $\mathcal{O}(n)$

Binäre Suche

in sortierten Daten

Daten:

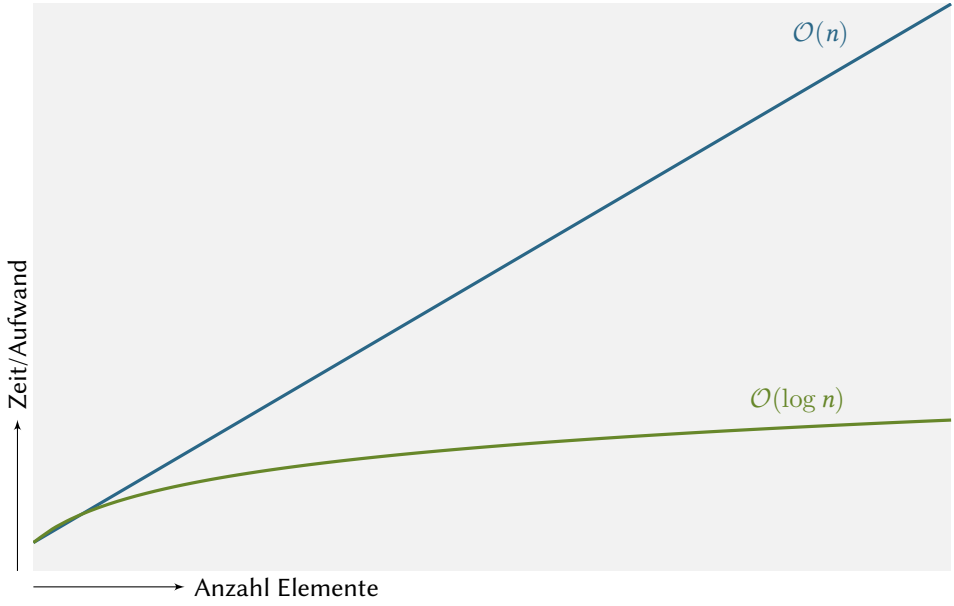
finde die 13:

- betrachte das mittlere Element aller Elemente
- ist es die 13?
  - ja: fertig
- ist es kleiner als 13?
  - ja: wiederhole in der rechten Hälfte
  - nein: wiederhole in der linken Hälfte

Aufwand:

- $\mathcal{O}(\log n)$

Laufzeit-Komplexität



Monkeysort

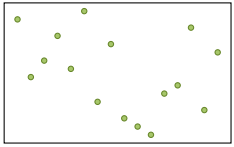
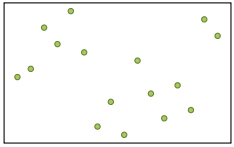
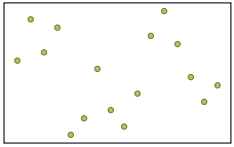
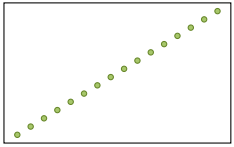
auch: Bogosort, Stupidsort, Permutationsort, Shotgunsort

Daten:

initial:	14	7	9	12	8	15	4	11	2	1	0	5	6	13	3	10
$i = 1$:	3	11	1	14	0	9	5	4	8	7	15	6	13	10	12	2
$i = 2$:	7	8	13	11	15	10	1	4	0	9	5	2	6	3	14	12
⋮																
$i = j$:	9	14	10	13	0	2	8	3	1	5	12	15	11	7	4	6
⋮																
$i = z$:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Monkeysort

auch: Bogosort, Stupidsort, Permutationsort, Shotgunsort

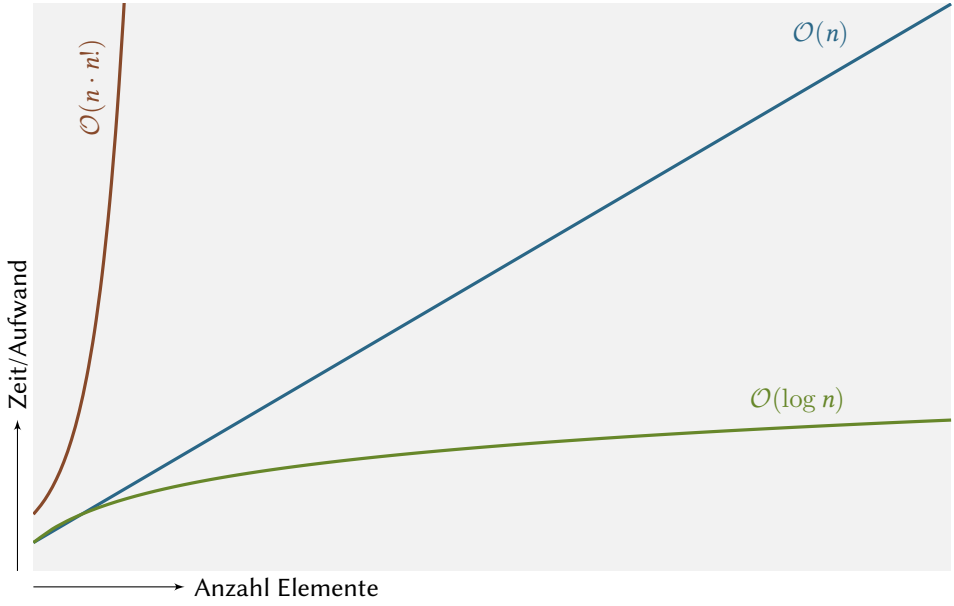
Ablauf:

- Sortiert?
  - ja: fertig
- wiederhole mit neuer, zufälliger Ordnung

Aufwand:

- bestenfalls: $\mathcal{O}(n)$
- im Mittel: $\mathcal{O}(n \cdot n!)$
- schlimmstenfalls: unbeschränkt

Laufzeit-Komplexität



Bubblesort

Daten:

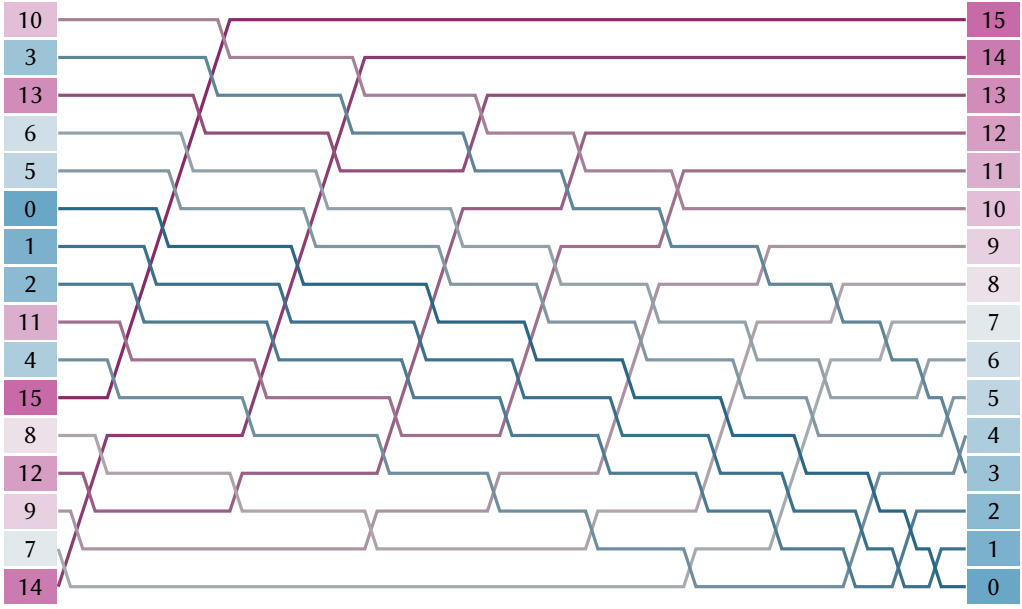
Ablauf:

- durchlaufe Array von vorne nach hinten
- vertausche unsortierte Wertpaare
- wiederhole solange Vertauschungen nötig sind

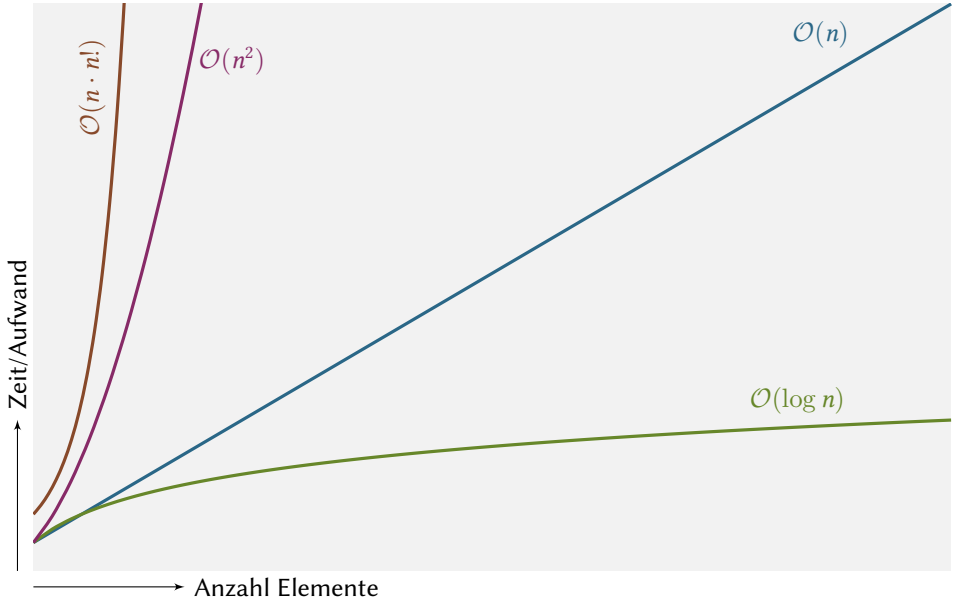
Aufwand:

- bestenfalls: $\mathcal{O}(n)$ Vergleiche, $\mathcal{O}(1)$ Tausche,
- im Mittel: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n^2)$ Tausche
- schlimmstenfalls: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n^2)$ Tausche

Bubblesort



Laufzeit-Komplexität



Selectionsort

Daten:

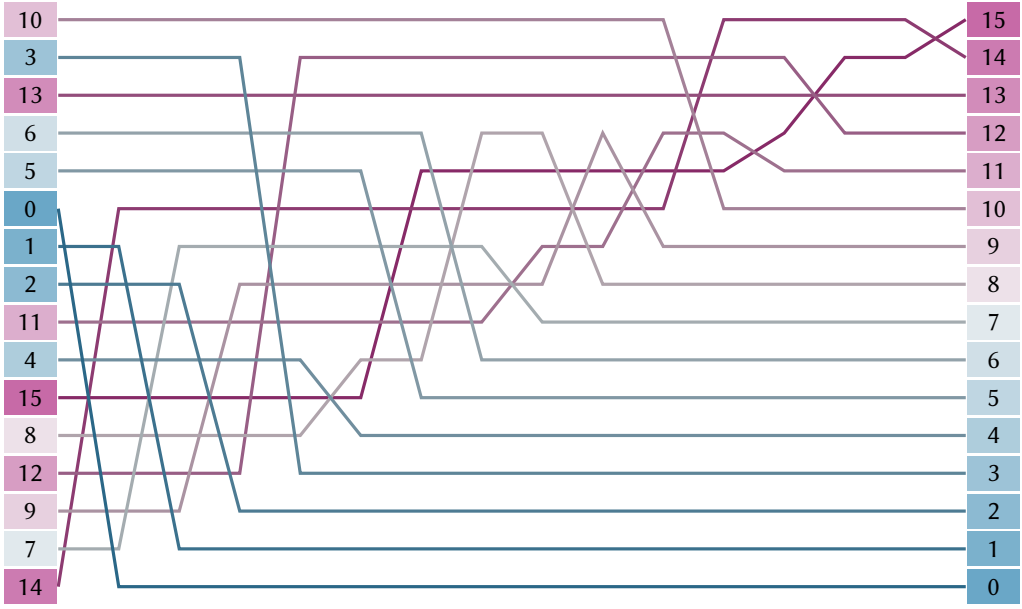
Ablauf:

- suche vorne beginnend das Minimum
- vertausche es mit dem ersten Element
- wiederhole ab dem zweiten Element

Aufwand:

- bestenfalls: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n)$ Tausche,
- im Mittel: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n)$ Tausche
- schlimmstenfalls: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n)$ Tausche

Selectionsort



Insertionsort

Daten:

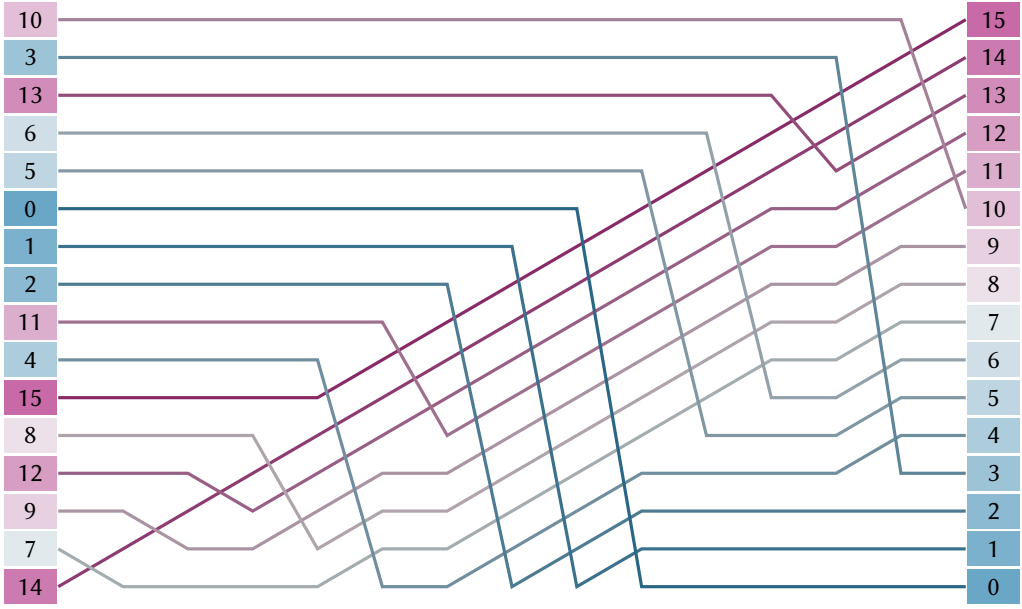
Ablauf:

- erstes Element ist sortiert
- tausche zweites Element nach links bis zur richtigen Position
- wiederhole mit dem nächsten Element

Aufwand:

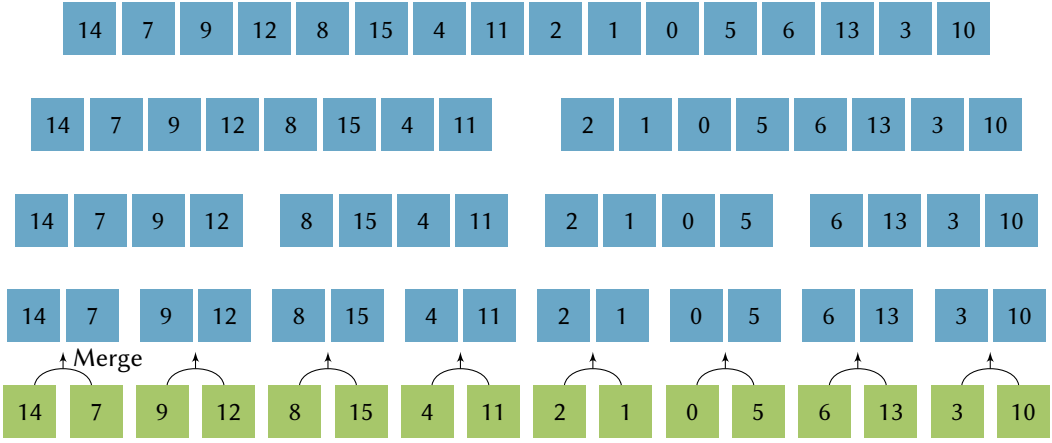
- bestenfalls: $\mathcal{O}(n)$ Vergleiche, $\mathcal{O}(1)$ Tausche,
- im Mittel: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n^2)$ Tausche
- schlimmstenfalls: $\mathcal{O}(n^2)$ Vergleiche, $\mathcal{O}(n^2)$ Tausche

Insertionsort



Mergesort

Daten:



Mergesort

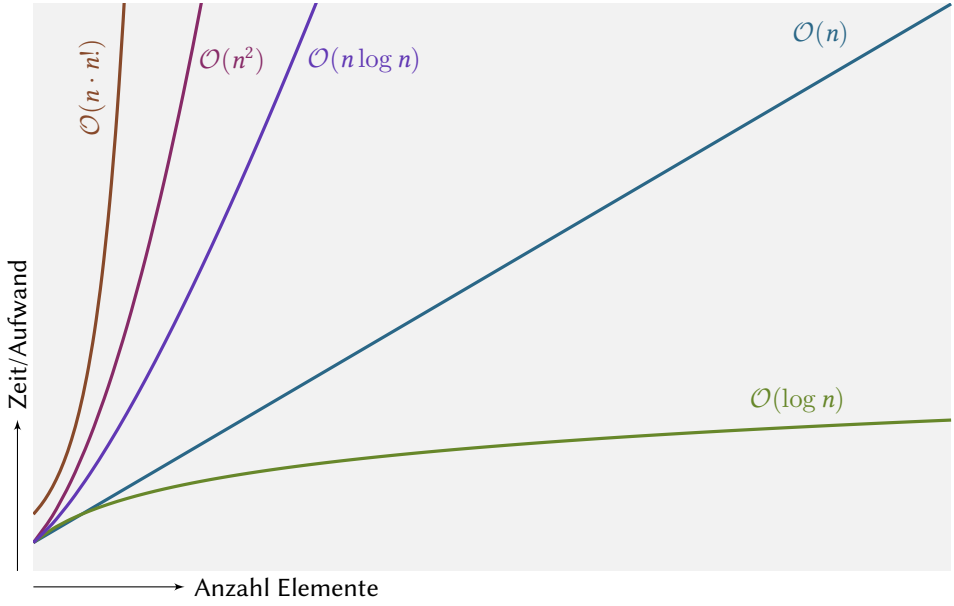
Ablauf:

- teile das Array rekursiv in zwei Hälften
- bleibt ein/kein Element über, ist der Teil sortiert
- sortiere jeweils beim Zusammenfügen

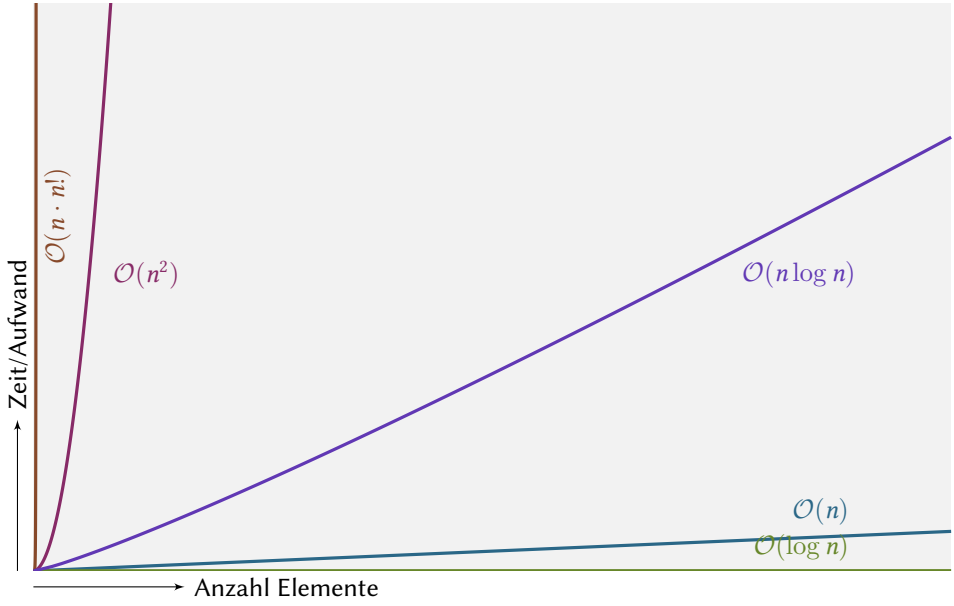
Aufwand:

- bestenfalls: $\mathcal{O}(n \log n)$
- im Mittel: $\mathcal{O}(n \log n)$
- schlimmstenfalls: $\mathcal{O}(n \log n)$

Laufzeit-Komplexität



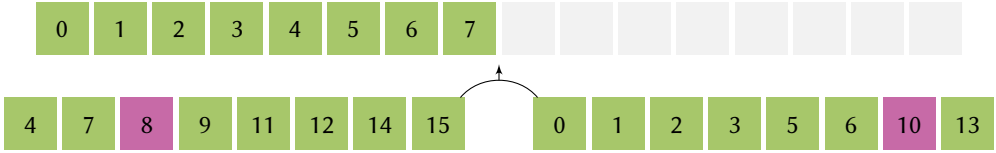
Laufzeit-Komplexität



Mergesort

sortiere beim Zusammenfügen

Daten:



Ablauf:

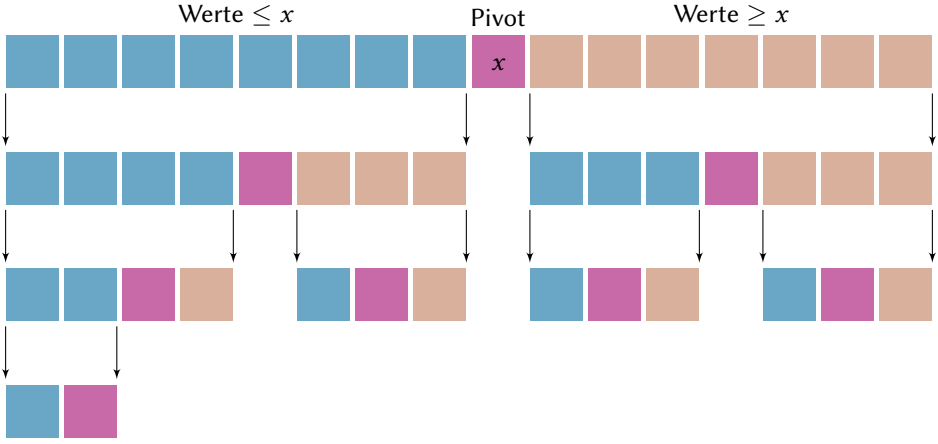
- betrachte in beiden Teilen den Anfang
- übertrage das kleinere Element
- mache nur in diesem Teil einen Schritt
- wiederhole

Aufwand:

- $\mathcal{O}(n)$

Quicksort

Prinzip



Quicksort

Lomuto-Partition

Daten:

Ablauf:

- wähle Pivot-Element
- laufe von vorn nach hinten
- tausche dabei Inversionen
- wiederhole rekursiv für beide Teile

Aufwand:

- bestenfalls: $\mathcal{O}(n \log n)$
- im Mittel: $\mathcal{O}(n \log n)$
- schlimmstenfalls: $\mathcal{O}(n^2)$

Quicksort

Hoare-Partition

Daten:

Ablauf:

- wähle Pivot-Element
- laufe von außen nach innen
- tausche dabei Inversionen
- wiederhole rekursiv für beide Teile

Aufwand:

- bestenfalls: $\mathcal{O}(n \log n)$
- im Mittel: $\mathcal{O}(n \log n)$
- schlimmstenfalls: $\mathcal{O}(n^2)$

Minimaler Aufwand

Mögliche Sortierungen

- n Elemente können in $n!$ Permutationen angeordnet werden.

Beispiel:

- 4 Elemente: 24 Permutationen

1	2	3	4	2	1	3	4	3	1	2	4	4	1	2	3
1	2	4	3	2	1	4	3	3	1	4	2	4	1	3	2
1	3	2	4	2	3	1	4	3	2	1	4	4	2	1	3
1	3	4	2	2	3	4	1	3	2	4	1	4	2	3	1
1	4	2	3	2	4	1	3	3	4	1	2	4	3	1	2
1	4	3	2	2	4	3	1	3	4	2	1	4	3	2	1

Minimaler Aufwand

Entscheidungen à la „Wer bin ich?“

$a_0 < a_1 \wedge a_0 < a_2 \wedge a_0 < a_3 \wedge a_1 < a_2 \wedge a_1 < a_3 \wedge a_2 < a_3$															
1	2	3	4	2	1	3	4	3	1	2	4	4	1	2	3
1	2	4	3	2	1	4	3	3	1	4	2	4	1	3	2
1	3	2	4	2	3	1	4	3	2	1	4	4	2	1	3
1	3	4	2	2	3	4	1	3	2	4	1	4	2	3	1
1	4	2	3	2	4	1	3	3	4	1	2	4	3	1	2
1	4	3	2	2	4	3	1	3	4	2	1	4	3	2	1

Minimaler Aufwand

Herleitung

im Beispiel:

- 24 Permutationen: 6 Vergleiche
- mindestens $\log_2 n!$ Vergleiche?

allgemeiner:

- starte mit $n!$ Permutationen
- Spieler nennt Vergleichspaar
- Gegner wählt Vergleichsoperator
  - möglichst viele Elemente übrig bleiben
  - maximal die Hälfte der Elemente wird aussortiert
- mindestens $\log_2 n!$ Vergleiche erforderlich

Minimaler Aufwand

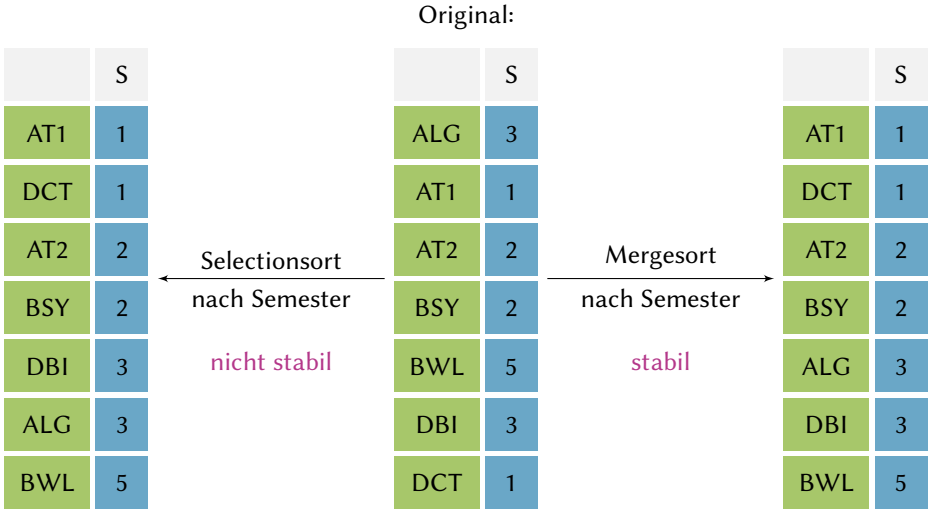
Herleitung

formaler:

- der Entscheidungsbaum hat $n!$ Blätter
- ein Binärbaum der Höhe h hat 2^h Blätter
- für einen passenden Binärbaum:

$$
\begin{aligned}
2^h \geq n! \Longrightarrow h \geq \log_2 n! = \sum_{i=1}^{n} \log_2 i &= \sum_{i=1}^{\frac{n}{2}-1} \log_2 i + \sum_{i=\frac{n}{2}}^{n} \log_2 i \\
&\geq \sum_{i=\frac{n}{2}}^{n} \log_2 \frac{n}{2} \\
&= \frac{n}{2} \cdot \log_2 \frac{n}{2} \\
&\in \mathcal{O}(n \log n)
\end{aligned}
$$

Stabilität



- ein stabiles Sortierverfahren erhält relative Ordnung

Zusammenfassung

	best.	mittel	schlecht.	stabil
Monkeysort	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot n!)$	?	nein
Bubblesort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	ja
Selectionsort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	nein
Insertionsort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	ja
Mergesort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	ja
Quicksort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	nein