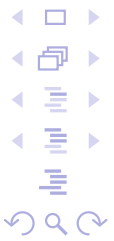
Algorithmentechnik — Übung 1

http://i11www.ira.uka.de/teaching/WS_0607/algotech

Robert Görke (rgoerke@ira.uka.de)

WS 0607



Übersicht
Informationen
Aufgabe 1
- Teil (a)
- Teil (b)
- Teil (c)

Aufgabe 2
- Teil (a)
- Teil (b)
- Teil (c)

Aufgabe 3
- Teil (a)
- Teil (b)

Aufgabe 4
- Teil (a)
- Teil (b)

Ende

Übungsblätter, Termine, Klausurbonus, etc.

- Algorithmentechnik (Hauptklausur): 01.03.07
- Algorithmentechnik (Wiederholerklausur): 12.04.07

- Übungsblätter erscheinen Dienstags (14-tg.)
- Abgabe Mittwochs bis 15:30 im 3. Stock Fasaneng. Ostflügel
- *erfolgreich* = mind. 50% Punkte = 0.5 Klausurpunkte (v. 60)
- maximal ein Teilnotenschritt
- alle Blätter *erfolgreich* $\Rightarrow$ ein Teilnotenschritt
- Zweiergruppen ausdrücklich erwünscht

- Inhalt der Übungen: Lösungen der Blätter + Diverses

Änderung auf Übungsblatt 2

Aufgabe 3:

- Findmax() in $O(k)$ (nicht in ~~$O(\log k)$~~)
- $k \in O(n)$
- Titel: „Alternative zu Heaps"'(als Hinweis)

Sonstige Hinweise:

- Aufgabe 1: Erstmal Algorithmus klar machen!
- Aufgabe 2: Problem verstehen $\Rightarrow$ Algorithmus einfach

Änderung auf Übungsblatt 2

Aufgabe 3:

- Findmax() in $O(k)$ (nicht in ~~$O(\log k)$~~)
- $k \in O(n)$
- Titel: „Alternative zu Heaps"'(als Hinweis)

Sonstige Hinweise:

- Aufgabe 1: Erstmal Algorithmus klar machen!
- Aufgabe 2: Problem verstehen $\Rightarrow$ Algorithmus einfach

Änderung auf Übungsblatt 2

Aufgabe 3:

- Findmax() in $O(k)$ (nicht in ~~$O(\log k)$~~)
- $k \in O(n)$
- Titel: „Alternative zu Heaps"'(als Hinweis)

Sonstige Hinweise:

- Aufgabe 1: Erstmal Algorithmus klar machen!
- Aufgabe 2: Problem verstehen $\Rightarrow$ Algorithmus einfach

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- Pseudocode!
- $O(), \Omega(), \Theta()$ $\neq$ Best-case, Avg-case, Worst-case!
- $3\frac{1}{2}$ Punkte sind auch gut...

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- **Pseudo**code!
- $O(), \Omega(), \Theta()$ $\neq$ Best-case, Avg-case, Worst-case!
- $3\frac{1}{2}$ Punkte sind auch gut...

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- **Pseudo**code!
- $O(), \Omega(), \Theta()$ $\neq$ Best-case, Avg-case, Worst-case!
- $3\frac{1}{2}$ Punkte sind auch gut...

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- **Pseudo**code!
- $O(), \Omega(), \Theta()$ $\neq$ Best-case, Avg-case, Worst-case!
- $3\frac{1}{2}$ Punkte sind auch gut...

Average-case-Laufzeit vs. Worst-case-Laufzeit

Betrachten Sie das folgende Suchproblem **lineare Suche**:

- **Eingabe:** Sequenz von n ganzen Zahlen $A[1, \ldots, n]$, Wert v
- **Ausgabe:** Index i, mit $v = A[i]$ falls v in A, sonst NIL

(a) Pseudocode, Korrektheit

(b) Average-case, und die Worst-case Laufzeit asymptotisch scharf (Vergleichsoperationen)

(c) Untere Schranke für lineare Suche

Pseudocode

Schreiben Sie ein Programm in Pseudocode, und beweisen Sie, dass Ihr Algorithmus korrekt ist.

Algorithmus 1 : LineareSuche(A, v)

Eingabe : Array A der Länge n, Wert v
Ausgabe : Index i mit $A[i] = v$

```
1 Für i ← 1 ... n
2     Wenn A[i] = v
3         return i
4 return NIL
```

Korrektheit

- $v \in A$
  Sei $A[k] = v$ das linkeste Vorkommen von v $\Rightarrow$ Zeile 3 wird ausgeführt
  $\Rightarrow$ Return k ist korrekt.
- $v \notin A$
  Für keinen Index wird Zeile 3 ausgeführt
  $\Rightarrow$ Return NIL$\Rightarrow$ korrekt

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \quad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \quad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$
- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$
- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Laufzeiten

Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

Worst-case:

- $v \notin A \Rightarrow T_{v \notin A}(n) = n$
- Maximale Anzahl Schleifendurchläufe $\Rightarrow$ Worst–case Laufzeit $T^{\text{worst}}(n) = n \in \Theta(n)$

Average-Case:

- **Annahme:** Gleichverteilung der Zahlen $\Rightarrow |\mathbb{Z}| = \aleph_0 = \infty$
- $\Rightarrow \Pr(A[k] = v) = 0 \qquad$ für $1 \leq k \leq n$
- $\Rightarrow T^{\text{avg}}(n) = T^{\text{worst}}(n) = n \in \Theta(n)$

- Annahme dass Zahlen $\in [1, \ldots, n]$ sind liefert auch $\Theta(n)$

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Untere Schranke

Im Worst-Case ($v \notin A$):

Jede Zahl wird verglichen!

$\Rightarrow$ untere Schranke für Worst-case $\Omega(n)$
(Untere Schranke für Avg-case: $\Omega(n)$)
(Untere Schranke für Best-case: $\Omega(1)$)

Kein Info $\Rightarrow$ möglichst scharf, Worst-case

Rekursion

(a) Asymptotisch scharfe Schranken für mit Mastertheorem

(b) *Rekursives Sortieren durch Einfügen*
Array $A[1, \ldots, n]$ ordnen, indem $A[1, \ldots, n-1]$ rekursiv geordnet wird und $A[n]$ danach einsetzen.
Pseudocode, Rekursionsgleichung, asymptotisch scharfe Worst-case Laufzeit.

(c) Array $A[1, \ldots, n]$ enthält alle ganzen Zahlen von 0 bis n außer einer.
Binär dargestellt, einzige Operation: *rufe das j-te Bit von $A[i]$ auf.*
Pseudocode mit Wort-case Laufzeit $O(n)$. Möglichst scharfe obere Schranke für Laufzeit.

Mastertheorem

(i) $T(n) = 4T(n/2) + n$

(ii) $T(n) = 4T(n/2) + n^2$

(iii) $T(n) = 4T(n/2) + n^3$

Wende die allgemeinere Form des Master-Theorems an:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Mastertheorem

(i) $T(n) = 4T(n/2) + n$

(ii) $T(n) = 4T(n/2) + n^2$

(iii) $T(n) = 4T(n/2) + n^3$

Wende die allgemeinere Form des Master-Theorems an:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1 \quad \Rightarrow c = 2$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1 \quad \Rightarrow c = 2$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1 \quad \Rightarrow c = 2$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (i)

$T(n) = 4T(n/2) + n$ mit Hilfe von: $T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n \in \Theta(n)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 1 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot \frac{1}{2} = 2 > 1$
- $\Rightarrow T(n) \in \Theta(n^c)$
- wobei $\sum_{i=1}^{m} \alpha_i^c = 1 \quad \Rightarrow c = 2$
- $T(n) \in \Theta(n^2)$.

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:

- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:

- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:

- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (ii)

$T(n) = 4T(n/2) + n^2$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n^2)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^2 \in \Theta(n^2)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 2 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^2 = 1$
- $\Rightarrow T(n) \in \Theta(n^k \log n)$
- $\Rightarrow T(n) \in \Theta(n^2 \log n)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Mastertheorem (iii)

$T(n) = 4T(n/2) + n^3$ mit Hilfe von:
$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$

Beobachtungen:
- $m = 4$
- $\alpha_i = \frac{1}{2}$
- $f(n) = n^3 \in \Theta(n^3)$

Vorraussetzungen:
- $0 < \alpha_i < 1$
- $m \geq 1$
- $f(n) \in \Theta(n^k), k = 3 \geq 0$

Lösung:
- $\sum_{i=1}^{m} \alpha_i^k = 4 \cdot (\frac{1}{2})^3 = \frac{1}{2} < 1$
- $\Rightarrow T(n) \in \Theta(n^k)$
- $\Rightarrow T(n) \in \Theta(n^3)$

Rekursives Sortieren durch Einfügen

Array $A[1, \ldots, n]$ ordnen, indem $A[1, \ldots, n-1]$ rekursiv geordnet wird und $A[n]$ danach einsetzen.

Pseudocode, Rekursionsgleichung, asymptotisch scharfe Worst-case Laufzeit.

(aufsteigend)

Rekursives Sortieren durch Einfügen

Algorithmus 2 : RecursiveInsertSort(A, j)

Eingabe : Array A, Index j
Ausgabe : sortiertes Array A

```
1  Wenn j == 1
2      return A
3  sonst
4      RecursiveInsertSort(A, j − 1)
5      i = 1
6      solange i < j tue
7          Wenn A[j] < A[i]
8              Fge A[j] an Stelle i ein und verschiebe Rest des Arrays um
               eine Stelle nach hinten
9              return A
10         sonst
11             i ← i + 1
12     return A
```

Analyse von RecursiveInsertSort(A, j)

- Worst-case: Bereits vorsortiertes Array!
- Rekursionsgleichung:

$$T(n) \in \begin{cases} \Theta(1), & \text{wenn } n = 1 \\ T(n-1) + \Theta(n), & \text{wenn } n > 1 \end{cases}$$

- iterative Formel in eine geschlossene Form bringen:

$$T(n) \in \sum_{i=1}^{n} \Theta(i) \Rightarrow T(n) \in \Theta(\tfrac{(n+1)\cdot n}{2}) = \Theta(n^2)$$

Analyse von RecursiveInsertSort(A, j)

- Worst-case: Bereits vorsortiertes Array!
- Rekursionsgleichung:

$$T(n) \in \begin{cases} \Theta(1), & \text{wenn } n = 1 \\ T(n-1) + \Theta(n), & \text{wenn } n > 1 \end{cases}$$

- iterative Formel in eine geschlossene Form bringen:
$T(n) \in \sum_{i=1}^{n} \Theta(i) \Rightarrow T(n) \in \Theta(\frac{(n+1)\cdot n}{2}) = \Theta(n^2)$

Analyse von RecursiveInsertSort(A, j)

- Worst-case: Bereits vorsortiertes Array!
- Rekursionsgleichung:

$$T(n) \in \begin{cases} \Theta(1), & \text{wenn } n = 1 \\ T(n-1) + \Theta(n), & \text{wenn } n > 1 \end{cases}$$

- iterative Formel in eine geschlossene Form bringen:
$T(n) \in \sum_{i=1}^{n} \Theta(i) \Rightarrow T(n) \in \Theta(\frac{(n+1)\cdot n}{2}) = \Theta(n^2)$

Fehlende Zahl

Array $A[1, \ldots, n]$ enthält alle ganzen Zahlen von 0 bis n außer einer.
Binär dargestellt, einzige Operation: *rufe das j-te Bit von $A[i]$ auf.*
Pseudocode mit Wort-case Laufzeit $O(n)$. Möglichst scharfe obere Schranke für Laufzeit.

Algorithmus 3 : FindMissingNumber(A, j)

```
1  Wenn A.length == 1
2      kippe das j-te Bit von A[0]
3      return A[0]
4  sonst
5      z_0 ← 0,   z_1 ← 0
6      Für i ∈ A
7          x ← das j-te Bit von A[i]
8          Wenn x == 0
9              B_0[z_0] = A[i] und z_0 ++
10         sonst
11             B_1[z_1] = A[i] und z_1 ++
12     j ++
13     Wenn z_0 < z_1
14         FindMissingNumber(B_0, j)
15     sonst
16         FindMissingNumber(B_1, j)
```

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$
- (Es geht auch mit einem oder keinem Hilfsarray!)

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$
- (Es geht auch mit einem oder keinem Hilfsarray!)

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$
- (Es geht auch mit einem oder keinem Hilfsarray!)

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$
- (Es geht auch mit einem oder keinem Hilfsarray!)

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow$ $T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$
- (Es geht auch mit einem oder keinem Hilfsarray!)

Laufzeit für die Fehlende Zahl

- Laufzeit: Die Zeilen 6- 11 werden n-mal durchlaufen
- $\Rightarrow$ Kosten $c \cdot n$, c konstant
- Im nächsten Rekursionsschritt wird nur kleinere Hälfte der Zahlen betrachtet!
- Die Rekurrenzgleichung lautet also: $T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + cn$
- Mastertheorem $\Rightarrow$ $T^{\text{worst}}(n) \in \Theta(n) \subseteq (n)$

- (Es geht auch mit einem oder keinem Hilfsarray!)

Amortisierte Analysen

(a) Datenstruktur, Sequenz von n Operationen.
i-te Operation hat Kosten i, wenn i eine Potenz von 2 ist, ansonsten Kosten 1
Ganzheitsmethode, Buchungsmethode

(b) Binärer Zähler mit Increment und Reset
Zähler als Bitfeld, so dass eine beliebige Folge von Increment- und Reset- Operationen in $O(n)$
Pseudocode, amortisierte Analyse mit Hilfe der Buchungsmethode.

Amortisierte Analysen

(a) Datenstruktur, Sequenz von n Operationen.
i-te Operation hat Kosten i, wenn i eine Potenz von 2 ist, ansonsten Kosten 1
Ganzheitsmethode, Buchungsmethode

(b) Binärer Zähler mit Increment und Reset
Zähler als Bitfeld, so dass eine beliebige Folge von Increment- und Reset- Operationen in $O(n)$
Pseudocode, amortisierte Analyse mit Hilfe der Buchungsmethode.

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

c_i die Kosten der i–ten Operation:

$$c_i = \begin{cases} i & \text{falls } i \text{ eine Potenz von 2 ist} \\ 1 & \text{sonst} \end{cases}$$

Die Werte von c_i für $i = 1, \ldots, 10$:

Operation i	Kosten c_i
1	1
2	2
3	1
4	4
5	1
6	1
7	1
8	8
9	1
10	1
⋮	⋮

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

c_i die Kosten der i–ten Operation:

$$c_i = \begin{cases} i & \text{falls } i \text{ eine Potenz von 2 ist} \\ 1 & \text{sonst} \end{cases}$$

Die Werte von c_i für $i = 1, \ldots, 10$:

Operation i	Kosten c_i
1	1
2	2
3	1
4	4
5	1
6	1
7	1
8	8
9	1
10	1
⋮	⋮

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

c_i die Kosten der i–ten Operation:

$$c_i = \begin{cases} i & \text{falls } i \text{ eine Potenz von 2 ist} \\ 1 & \text{sonst} \end{cases}$$

Die Werte von c_i für $i = 1, \ldots, 10$:

Operation i	Kosten c_i
1	1
2	2
3	1
4	4
5	1
6	1
7	1
8	8
9	1
10	1
⋮	⋮

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Kosten für n Operationen ergeben sich durch

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \leq n + (2n - 1) < 3n$$

Durchschnittlichen Kosten pro Operation:

$$\frac{\text{Gesamtkosten}}{\text{Anzahl Operationen}} < \frac{3n}{n} = 3$$

Eine Operation hat also amortisierte Kosten von $O(1)$ Eine Operation hat aber auch amortisierte Kosten von $\Omega(1)$

$\Rightarrow T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Kosten für n Operationen ergeben sich durch

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \leq n + (2n - 1) < 3n$$

Durchschnittlichen Kosten pro Operation:

$$\frac{\text{Gesamtkosten}}{\text{Anzahl Operationen}} < \frac{3n}{n} = 3$$

Eine Operation hat also amortisierte Kosten von $O(1)$ Eine Operation hat aber auch amortisierte Kosten von $\Omega(1)$
$\Rightarrow T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Kosten für n Operationen ergeben sich durch

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \leq n + (2n - 1) < 3n$$

Durchschnittlichen Kosten pro Operation:

$$\frac{\text{Gesamtkosten}}{\text{Anzahl Operationen}} < \frac{3n}{n} = 3$$

Eine Operation hat also amortisierte Kosten von $O(1)$ Eine Operation hat aber auch amortisierte Kosten von $\Omega(1)$
$\Rightarrow T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Kosten für n Operationen ergeben sich durch

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \leq n + (2n - 1) < 3n$$

Durchschnittlichen Kosten pro Operation:

$$\frac{\text{Gesamtkosten}}{\text{Anzahl Operationen}} < \frac{3n}{n} = 3$$

Eine Operation hat also amortisierte Kosten von $O(1)$ Eine

Operation hat aber auch amortisierte Kosten von $\Omega(1)$
$\Rightarrow T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Gut geraten: Veranschlagte Kosten pro Operation: $\hat{c}_i = 3$

Op. i	Amort. K. $\hat{c}_i$	Kosten c_i	Restkredit
1	3	1	2
2	3	2	3
3	3	1	5
4	3	4	4
5	3	1	6
6	3	1	8
7	3	1	10
8	3	8	5
9	3	1	7
10	3	1	9
⋮	⋮	⋮	⋮

!Müssen noch **zeigen**, dass Kredit niemals negativ!

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Gut geraten: Veranschlagte Kosten pro Operation: $\hat{c}_i = 3$

Op. i	Amort. K. $\hat{c}_i$	Kosten c_i	Restkredit
1	3	1	2
2	3	2	3
3	3	1	5
4	3	4	4
5	3	1	6
6	3	1	8
7	3	1	10
8	3	8	5
9	3	1	7
10	3	1	9
⋮	⋮	⋮	⋮

!Müssen noch **zeigen**, dass Kredit niemals negativ!

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Gut geraten: Veranschlagte Kosten pro Operation: $\hat{c}_i = 3$

Op. i	Amort. K. $\hat{c}_i$	Kosten c_i	Restkredit
1	3	1	2
2	3	2	3
3	3	1	5
4	3	4	4
5	3	1	6
6	3	1	8
7	3	1	10
8	3	8	5
9	3	1	7
10	3	1	9
⋮	⋮	⋮	⋮

!Müssen noch **zeigen**, dass Kredit niemals negativ!

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart: $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten – Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart:
  $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten – Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart: $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart: $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart: $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart:
  $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart:
  $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Kosten: $1, 2, 1, 4, 1, 1, 1, 8, 1, 1, \ldots$

Induktion:

Anfang: Fr $i = 1$ Kredit nicht negativ.

Annahme: Fr alle Schritte $< i$ Kredit nicht negativ.

Schritt: Betrachte Schritt i

Fall 1: i keine Zweierpotenz $\Rightarrow$ Restkredit wächst $\Rightarrow$ OK

Fall 2: i Potenz von 2 $\Rightarrow i = 2^k$ für $k \geq 1$.

- $\Rightarrow$ Kosten $c_i = i$
- $2^k = 2 \cdot 2^{k-1} \Rightarrow$ es gibt genau $2^{k-1} - 1$ Operationen seit der letzten Zweierpotenz!
- $\Rightarrow$ Haben gespart: $(2^{k-1} - 1) \cdot 2 + \hat{c}_i = (2^{k-1} - 1) \cdot 2 + 3 = 2^k + 1$
- $\Rightarrow$ Kredit von $2^k + 1$ genügt um i–te Operation zu bezahlen

$\Rightarrow \forall$ Operationen: Amortisierte Kosten $-$ Eigentliche Kosten ≥ 0

$\Rightarrow T^{\text{amort}}(n) \in O(n)$

$\Rightarrow$ Wie oben gilt also $T^{\text{amort}}(n) \in \Theta(n)$

Binärer Zähler mit Reset

Binärer Zähler mit Increment und Reset
Zähler als Bitfeld, so dass eine beliebige Folge von Increment- und Reset- Operationen in $O(n)$
Kippen eines Bits hat Kosten 1
Pseudocode, amortisierte Analyse mit Hilfe der Buchungsmethode.

Pseudocode - Increment

Algorithmus 4 : Increment(A)

Eingabe : Bitfeld A
Seiteneffekte : Inkrementieren um 1 und Anpassen von max[A]

```
1  i ← 0
2  solange i < |A| und A[i] = 1 tue
3  |   A[i] ← 0
4  |   i ← i + 1
5  Wenn i < |A|
6  |   A[i] ← 1
7  |   Wenn i > max[A]
8  |   |   max[A] ← i
9  sonst
10 |   max[A] ← −1
```

Pseudocode - Reset

Algorithmus 5 : Reset(A)

Eingabe : Bitfeld A
Seiteneffekte : Zurcksetzen von A und max[A]

```
1 Für i ← 0 ... max[A]
2     A[i] ← 0
3 max[A] ← −1
```

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse - Intuition

Negative Beobachtungen:

- (Increase) Im Worst-case kippen $\Omega(n)$ Bits
- (Reset) Im Worst-case kippen $\Omega(n)$ Bits

Positive Beobachtungen:

- (Increase) Bit kann nur $1 \longrightarrow 0$ kippen wenn zuvor $1 \longrightarrow 0$!
- (Reset) Bit kann nur $1 \longrightarrow 0$ kippen wenn es zuvor Max1 war!

Idee:
Vorausbezahlen!

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von max[A]
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)
- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)

- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)

- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Amortisierte Analyse

Veranschlage Kosten 4 pro Operation!

- Nutze 2 pro Operation für *traditionellen* Bitzähler
  Wie in Übung 1 vorgestellt.
  (ohne Reset)
- Nutze übrige Zahlung von 2 für Reset wie folgt:
- Bezahle 1 p.I. für Aktualisierung von $\max[A]$
- Buche Kredit 1 auf das neue maximale Bit im Zähler
- $\Rightarrow$ Bezahle damit den Reset dieses Bits
- (Reset habe Kosten $1 < 4$)

- Amortisierte Kosten: $4 \in \Theta(1)$ pro Operation
- $\Rightarrow$ n Operationen $\in O(n)$

Union Find

Union(i, j) arbeitet mit *balancing*
(kleinerer Baum wird an größeren Baum „angehängt“)

Gesucht: Prozedur Union(i, j) mit *h-balancing*
(niedrigerer Baum wird an höheren Baum „angehängt“)

(a) Union(i, j) mit *h-balancing*
Pseudocode

(b) Lemma: Anzahl Elemente mit der Hilfe der Höhe des Baumes beschränkbar
Gilt Lemma auch für *h-balancing*?

Union Find

Union(i, j) arbeitet mit *balancing*
(kleinerer Baum wird an größeren Baum „angehängt“)

Gesucht: Prozedur Union(i, j) mit *h-balancing*
(niedrigerer Baum wird an höheren Baum „angehängt“)

(a) Union(i, j) mit *h-balancing*
Pseudocode

(b) Lemma: Anzahl Elemente mit der Hilfe der Höhe des Baumes beschränkbar
Gilt Lemma auch für *h-balancing*?

Union Find

Union(i, j) arbeitet mit *balancing*
(kleinerer Baum wird an größeren Baum „angehängt“)

Gesucht: Prozedur Union(i, j) mit *h-balancing*
(niedrigerer Baum wird an höheren Baum „angehängt“)

(a) Union(i, j) mit *h-balancing*
Pseudocode

(b) Lemma: Anzahl Elemente mit der Hilfe der Höhe des Baumes beschränkbar
Gilt Lemma auch für *h-balancing*?

Union Find

Union(i, j) arbeitet mit *balancing*
(kleinerer Baum wird an größeren Baum „angehängt“)

Gesucht: Prozedur Union(i, j) mit *h-balancing*
(niedrigerer Baum wird an höheren Baum „angehängt“)

(a) Union(i, j) mit *h-balancing*
Pseudocode

(b) Lemma: Anzahl Elemente mit der Hilfe der Höhe des Baumes beschränkbar
Gilt Lemma auch für *h-balancing*?

Pseudocode

Algorithmus 6 : h-Union(i, j)

```
1 Wenn |Vor[i]| < |Vor[j]|
2     Vor[i] ← j
3 sonst, wenn |Vor[j]| < |Vor[i]|
4     Vor[j] ← i
5 sonst
6     Vor[i] ← j und Vor[j] ← Vor[j] − 1
```

Anmerkung: Wurzel r eines Baumes speichert negierte Höhe des Baumes in Vor$[r]$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?

Ja. Beweis durch Induktion ber h-Union-Operationen:

Anfang: Keine h-Union-Operation:

$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK

Annahme: Lemma gilt bis zur n-ten h-Union-Operation:

Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)

O.B.d.A. sei $h(T_j) \geq h(T_i)$.

$\Rightarrow$ T_i wird an Wurzel von T_j angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i, j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?

Ja. Beweis durch Induktion ber h-Union-Operationen:

Anfang: Keine h-Union-Operation:

$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK

Annahme: Lemma gilt bis zur n-ten h-Union-Operation:

Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)

O.B.d.A. sei $h(T_j) \geq h(T_i)$.

$\Rightarrow$ T_i wird an Wurzel von T_j angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i, j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?
Ja. Beweis durch Induktion ber h-Union-Operationen:
Anfang: Keine h-Union-Operation:
$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK
Annahme: Lemma gilt bis zur n-ten h-Union-Operation:
Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)
O.B.d.A. sei $h(T_j) \geq h(T_i)$.
$\Rightarrow$ T_i wird an Wurzel von T_J angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?
Ja. Beweis durch Induktion ber h-Union-Operationen:
Anfang: Keine h-Union-Operation:
$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK
Annahme: Lemma gilt bis zur n-ten h-Union-Operation:
Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)
O.B.d.A. sei $h(T_j) \geq h(T_i)$.
$\Rightarrow$ T_i wird an Wurzel von T_J angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?
Ja. Beweis durch Induktion ber h-Union-Operationen:
Anfang: Keine h-Union-Operation:
$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK
Annahme: Lemma gilt bis zur n-ten h-Union-Operation:
Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)
O.B.d.A. sei $h(T_j) \geq h(T_i)$.
$\Rightarrow$ T_i wird an Wurzel von T_J angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Gilt $h(T) \leq \log_2 |T|$?
Ja. Beweis durch Induktion ber h-Union-Operationen:
Anfang: Keine h-Union-Operation:
$\forall T$: $h(T) = 0$ und $|T| = 1 \Rightarrow h(T) = 0 \leq \log_2 1 = 0 \Rightarrow$ OK
Annahme: Lemma gilt bis zur n-ten h-Union-Operation:
Schritt: Betrachte n-te h-Union-Operation h-Union(i, j)
O.B.d.A. sei $h(T_j) \geq h(T_i)$.
$\Rightarrow$ T_i wird an Wurzel von T_J angehängt $\Rightarrow$ $T_{\text{h-Union}}(i, j)$ mit:

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j) + 1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Höhe beschränkt Größ des Baumes?

Kruskal 56, Prim 57

$h(T_j) \geq h(T_i)$.

$$h(T_{\text{h-Union}}(i,j)) = \begin{cases} h(t_j), & \text{wenn } h(T_j) > h(T_i) \\ h(t_j)+1, & \text{wenn } h(T_j) = h(T_i) \end{cases}$$

Induktionsannahme gilt für T_j und $h(T_i)$

- Falls $h(T_j) > h(T_i)$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j) \overset{IA}{\leq} \log_2(|T_j|) \leq \log_2(|T_{\text{h-Union}}(i,j)|)$$

- Falls $h(T_j) = h(T_i)$: O.B.d.A. sei $|T_j| \geq |T_i|$, dann ist

$$h(T_{\text{h-Union}}(i,j)) = h(T_j)+1 = h(T_i)+1$$

$$\overset{IA}{\leq} \log_2(|T_i|)+1 = \log_2(2|T_i|) \overset{|T_j|\geq|T_i|}{\leq} \log_2(|T_{\text{h-Union}}(i,j)|)$$

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- {Lokal minimale Kanten} $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- {Lokal minimale Kanten} $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- {Lokal minimale Kanten} $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- {Lokal minimale Kanten} $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Ein paralleler MST-Algorithmus (1926)

- Kruskal 1956, Prim 1957
- Jede lokal minimale Kante $\in$ MST ($\forall$ MST)
- $\{$Lokal minimale Kanten$\}$ $L(G)$ bilden Wald
- Boruvka-Phase: $L(G) \Rightarrow$ Kontraktion zu $B(G)$
- Laufzeit?
- $O(m+n)$
- Boruvka-Phase halbiert Knotenzahl (mindestens)
- $L(G) \bigcup$ MST von $B(G)$ ist MST von G (Hausaufgabe)
- Boruvka Phasen iterieren $\Rightarrow$ maximal $\log m$ mal
- Laufzeit $O(m \log n)$
- Parallelisierbar, Union Find

Termine

- Nächste Vorlesung: Dienstag, 21. November, 15:45 Uhr
- Übungsblatt Abgabe: Mittwoch, 22. November 15:30 Uhr
- Nächste Übung: Donnerstag, 30. November, 15.45 Uhr (D. Delling)