Algorithmentheorie

Daniel Neuen (Universität Bremen)

WiSe 2023/24

Rekursionsgleichungen

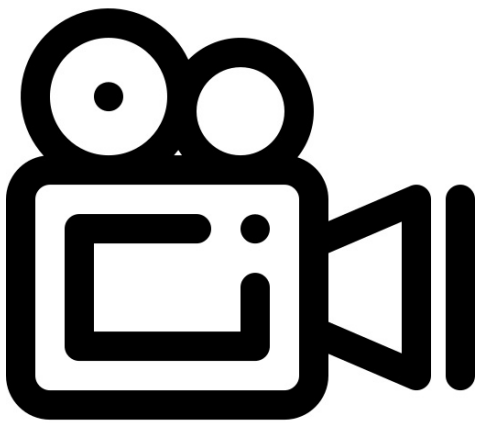
Master Theorem und Laufzeit von Sortierverfahren

3. Vorlesung

Aufzeichnung der Vorlesung

Diese Vorlesung wird aufgezeichnet und live gestreamt.

- Aufzeichnungen nur der Lehrenden durch sich selbst.
- Bei Rückfragen aus dem Auditorium und Diskussion bitte deutlich anzeigen, falls das Mikro stumm geschaltet werden soll.



Organisatorisches

Vorlesung:

- Nächste Vorlesung (09. Nov) wird von Prof. Siebertz gehalten.

Organisatorisches

Vorlesung:

- Nächste Vorlesung (09. Nov) wird von Prof. Siebertz gehalten.

Übung:

- Einige Einzel-Abgaben bei Blatt 1
- Bitte bilden Sie Gruppen von 3-4 Studierenden

Wiederholung: Algorithmendesignprinzip

Divide-and-Conquer (Teile-und-herrsche)

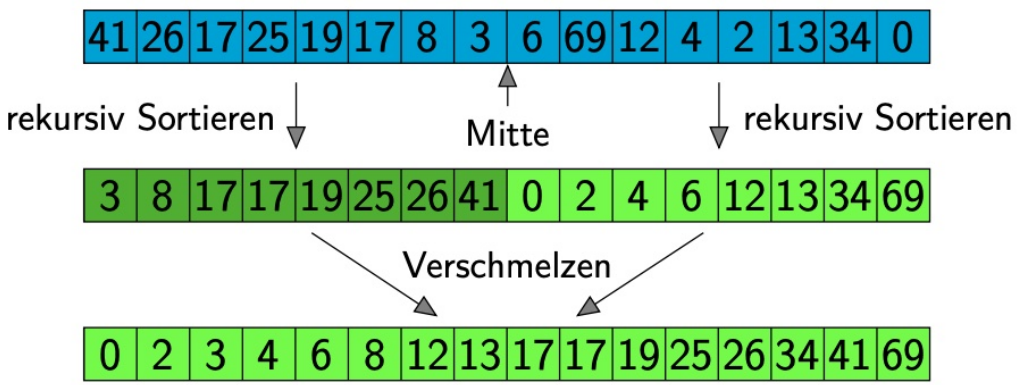
- Divide: Teile eine Probleminstanz in kleinere Teilinstanzen auf.
- Conquer: Löse die Teilinstanzen rekursiv, bis sie so klein sind, dass sie triviale Lösungen erlauben
- Combine: Kombiniere die Lösungen der Teilinstanzen zu einer Lösung des Gesamtproblems.

Wiederholung: Algorithmendesignprinzip

Divide-and-Conquer (Teile-und-herrsche)

- Divide: Teile eine Probleminstanz in kleinere Teilinstanzen auf.
- Conquer: Löse die Teilinstanzen rekursiv, bis sie so klein sind, dass sie triviale Lösungen erlauben
- Combine: Kombiniere die Lösungen der Teilinstanzen zu einer Lösung des Gesamtproblems.

Beispiel: Sortieralgorithmus MergeSort



© G. Woeginger

Wiederholung: MergeSort

```
1 MERGESORT(int[] a, int ℓ, int r)
  /* Eingabe: Feld a[0,...,n-1], ℓ ≤ r                    */
  /* Nach Ausführung ist a[ℓ,...,r] sortiert.             */
2 if ℓ < r then
3     m = ℓ + (r − ℓ)/2
4     MERGESORT(a, ℓ, m)
5     MERGESORT(a, m + 1, r)
6     MERGE(a, ℓ, m, r)
```

- Aufruf mit $\textsc{Mergesort}(\mathtt{a}, 0, \mathtt{n}-1)$.

Wiederholung: Merge

```
1  MERGE(int[] A, int ℓ, int m, int r)
2  n_1 := m − ℓ
3  n_2 := r − m − 1
4  copy A[ℓ, ..., m] to new array L[0, ..., n_1 + 1] with
     L[n_1 + 1] := ∞
5  copy A[m + 1, ..., r] to new array R[0, ..., n_2 + 1]
     with R[n_2 + 1] := ∞
6  i := 0, j := 0
7  for k = ℓ to r do
8      if L[i] ≤ R[j] then
9          A[k] := L[i]
10         i := i + 1
11     else
12         A[k] := R[j]
13         j := j + 1
```

Wiederholung: MergeSort

```
1 MERGESORT(int[] a, int ℓ, int r)
  /* Eingabe: Feld a[0,...,n−1], ℓ ≤ r                    */
  /* Nach Ausführung ist a[ℓ,...,r] sortiert.             */
2 if ℓ < r then
3     m = ℓ + (r − ℓ)/2
4     MERGESORT(a, ℓ, m)
5     MERGESORT(a, m + 1, r)
6     MERGE(a, ℓ, m, r)
```

- Aufruf mit $\textsc{Mergesort}(\mathtt{a}, 0, \mathtt{n}-1)$.
- Alg. $\textsc{Merge}$ in letzter VL besprochen; per Induktion gezeigt:

Satz

$\textsc{Merge}$ arbeitet korrekt und hat lineare Laufzeit $\mathcal{O}(r-\ell) = \mathcal{O}(n)$.

Wiederholung: MergeSort

```
1 MERGESORT(int[] a, int ℓ, int r)
  /* Eingabe: Feld a[0,...,n-1], ℓ ≤ r                    */
  /* Nach Ausführung ist a[ℓ,...,r] sortiert.             */
2 if ℓ < r then
3     m = ℓ + (r − ℓ)/2
4     MERGESORT(a, ℓ, m)
5     MERGESORT(a, m + 1, r)
6     MERGE(a, ℓ, m, r)
```

- Aufruf mit $\textsc{Mergesort}(\mathtt{a}, 0, \mathtt{n}-1)$.
- Alg. $\textsc{Merge}$ in letzter VL besprochen; per Induktion gezeigt:

Satz

$\textsc{Merge}$ arbeitet korrekt und hat lineare Laufzeit $\mathcal{O}(r-\ell) = \mathcal{O}(n)$.

Noch offen: Laufzeitanalyse MergeSort

Analyse rekursiver Algorithmen

Rekursionsgleichungen

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.
- Conquer:

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.
- Conquer:
  - Zeit $a \cdot T(\frac{n}{b})$ für rekursive Lösung.

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.
- Conquer:
  - Zeit $a \cdot T(\frac{n}{b})$ für rekursive Lösung.
  - Für $n \leq c$, d.h. kleine Instanzen: Zeit $T(c) \in \mathcal{O}(1)$.

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.
- Conquer:
  - Zeit $a \cdot T(\frac{n}{b})$ für rekursive Lösung.
  - Für $n \leq c$, d.h. kleine Instanzen: Zeit $T(c) \in \mathcal{O}(1)$.
- Combine: Zeit $C(n)$

Analyse rekursiver Algorithmen

Rekursionsgleichungen

- $T(n)$: Laufzeit für Instanzen der Größe n.
- Divide:
  - Teilung in a Teilinstanzen der Größe $\frac{1}{b}$ der Originalinstanz.
  - Zeit $D(n)$ um die Instanz zu teilen.
- Conquer:
  - Zeit $a \cdot T(\frac{n}{b})$ für rekursive Lösung.
  - Für $n \leq c$, d.h. kleine Instanzen: Zeit $T(c) \in \mathcal{O}(1)$.
- Combine: Zeit $C(n)$

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n \leq c \\ a \cdot T(\frac{n}{b}) + D(n) + C(n) & \text{sonst.} \end{cases}$$

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3 |   m = ℓ + (r − ℓ)/2
4 |   MERGESORT(a, ℓ, m)
5 |   MERGESORT(a, m + 1, r)
6 |   MERGE(a, ℓ, m, r)
```

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3     m = ℓ + (r − ℓ)/2
4     MERGESORT(a, ℓ, m)
5     MERGESORT(a, m + 1, r)
6     MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3 |   m = ℓ + (r − ℓ)/2
4 |   MERGESORT(a, ℓ, m)
5 |   MERGESORT(a, m + 1, r)
6 |   MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.
- Divide: Berechne m in $\mathcal{O}(1)$.

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3 |   m = ℓ + (r − ℓ)/2
4 |   MERGESORT(a, ℓ, m)
5 |   MERGESORT(a, m + 1, r)
6 |   MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.
- Divide: Berechne m in $\mathcal{O}(1)$.
- Conquer:

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3 |   m = ℓ + (r − ℓ)/2
4 |   MERGESORT(a, ℓ, m)
5 |   MERGESORT(a, m + 1, r)
6 |   MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.
- Divide: Berechne m in $\mathcal{O}(1)$.
- Conquer:
  - Löse 2 Teilinstanzen der Größe $\frac{n}{2}$ in Laufzeit $2 \cdot T(\frac{n}{2})$.

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3 |   m = ℓ + (r − ℓ)/2
4 |   MERGESORT(a, ℓ, m)
5 |   MERGESORT(a, m + 1, r)
6 |   MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.
- Divide: Berechne m in $\mathcal{O}(1)$.
- Conquer:
  - Löse 2 Teilinstanzen der Größe $\frac{n}{2}$ in Laufzeit $2 \cdot T(\frac{n}{2})$.
  - Basisfall: für $n = 1$ hat Laufzeit $\mathcal{O}(1)$.

Laufzeitanalyse Mergesort

```
1 MERGESORT(int[] a, int ℓ, int r)
2 if ℓ < r then
3     m = ℓ + (r − ℓ)/2
4     MERGESORT(a, ℓ, m)
5     MERGESORT(a, m + 1, r)
6     MERGE(a, ℓ, m, r)
```

- Annahme $n = 2^\ell$ für ein ℓ.
- Divide: Berechne m in $\mathcal{O}(1)$.
- Conquer:
  - Löse 2 Teilinstanzen der Größe $\frac{n}{2}$ in Laufzeit $2 \cdot T(\frac{n}{2})$.
  - Basisfall: für $n = 1$ hat Laufzeit $\mathcal{O}(1)$.
- Combine: Merge in $\mathcal{O}(n)$.

Laufzeitanalyse Mergesort

Rekursionsgleichung für Mergesort:

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n) & \text{sonst.} \end{cases}$$

Laufzeitanalyse Mergesort

Rekursionsgleichung für Mergesort:

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n) & \text{sonst.} \end{cases}$$

Mit geeignet gewählter Konstante c:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Laufzeitanalyse Mergesort

Rekursionsgleichung für Mergesort:

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n) & \text{sonst.} \end{cases}$$

Mit geeignet gewählter Konstante c:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Wie Rekursionsgleichung in geschlossenen Ausdruck bringen?

Laufzeitanalyse Mergesort: Rekursionsbäume

Rekursionsbaum:

- Baum mit einem Knoten für jedes rekursive Teilproblem.

Laufzeitanalyse Mergesort: Rekursionsbäume

Rekursionsbaum:

- Baum mit einem Knoten für jedes rekursive Teilproblem.
- Wert eines Knotens: Zeit für Divide and Combine.

Laufzeitanalyse Mergesort: Rekursionsbäume

Rekursionsbaum:

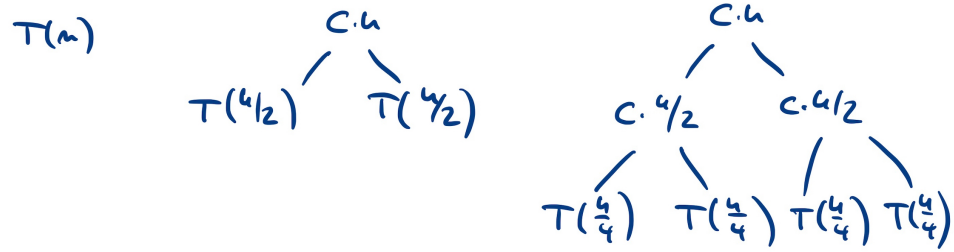
- Baum mit einem Knoten für jedes rekursive Teilproblem.
- Wert eines Knotens: Zeit für Divide and Combine.
- Gesamtlaufzeit: Summe der Werte aller Knoten des Baumes.

Laufzeitanalyse Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

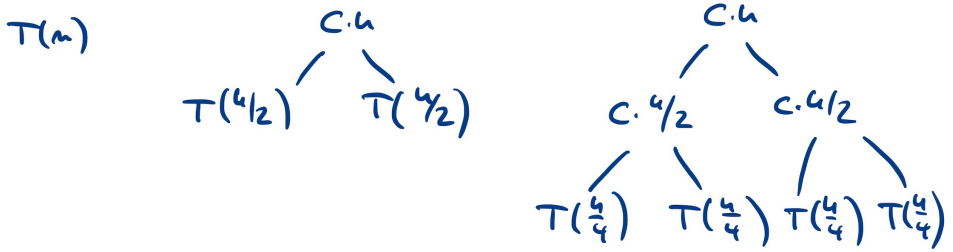
Laufzeitanalyse Mergesort

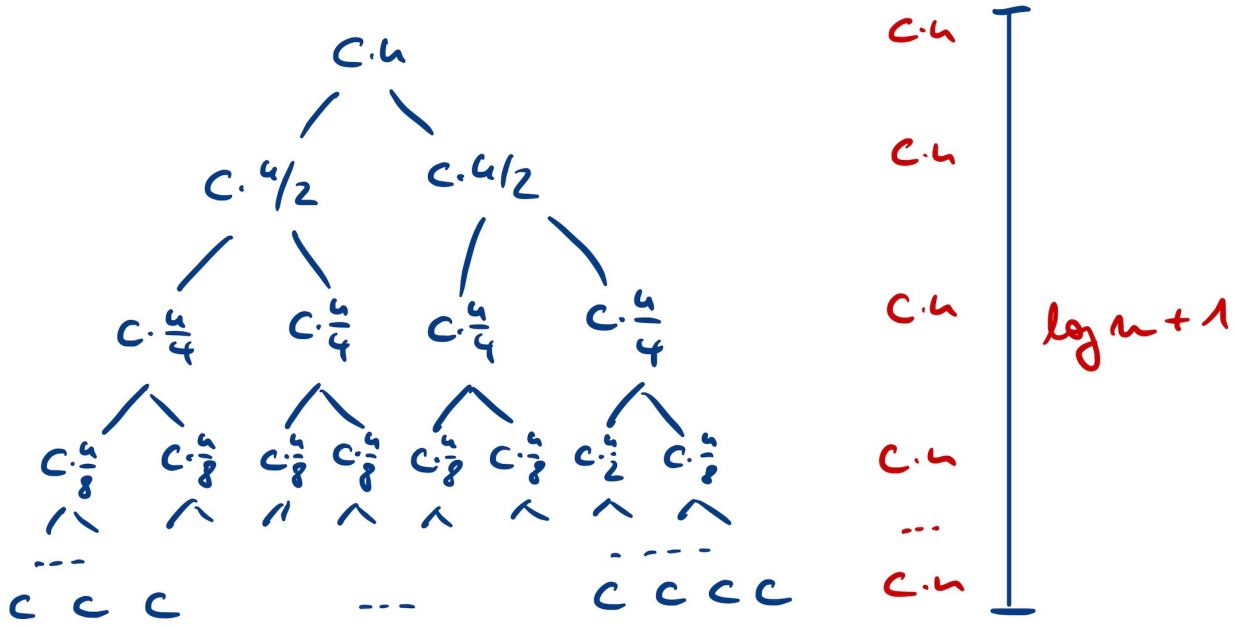
$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$



Laufzeitanalyse Mergesort

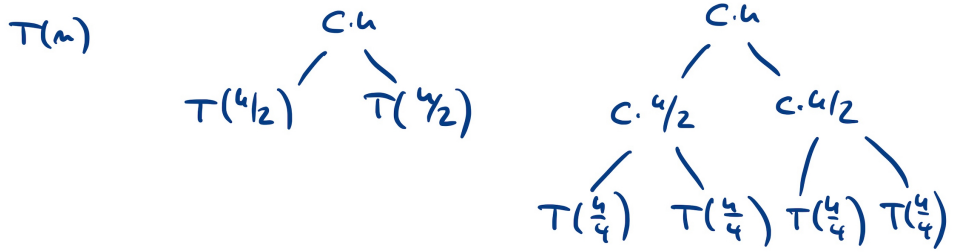
$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

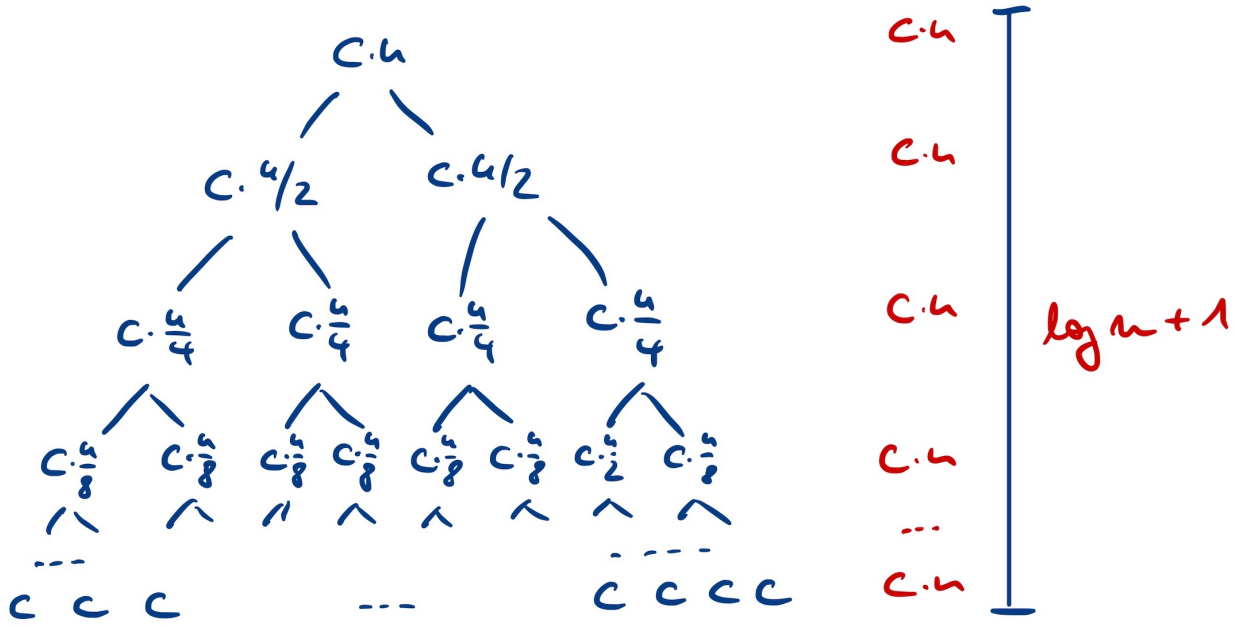




Laufzeitanalyse Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$





Laufzeit Mergesort: $T(n) = c \cdot n \cdot \log n + c \cdot n \in \mathcal{O}(n \log n)$.

Alternativ: Laufzeitanalyse Mergesort mit Induktion

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Wir beweisen per Induktion $T(n) = c \cdot n \cdot \log n + c \cdot n \in \mathcal{O}(n \log n)$.

Alternativ: Laufzeitanalyse Mergesort mit Induktion

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Wir beweisen per Induktion $T(n) = c \cdot n \cdot \log n + c \cdot n \in \mathcal{O}(n \log n)$.

- Induktionsanfang: Für $n = 1$ gilt $T(n) = c = c \cdot \log(1) + c$.

Alternativ: Laufzeitanalyse Mergesort mit Induktion

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Wir beweisen per Induktion $T(n) = c \cdot n \cdot \log n + c \cdot n \in \mathcal{O}(n \log n)$.

- Induktionsanfang: Für $n = 1$ gilt $T(n) = c = c \cdot \log(1) + c$.
- Induktionsschritt: Wir betrachten $n \geq 2$.

Alternativ: Laufzeitanalyse Mergesort mit Induktion

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + c \cdot n & \text{sonst.} \end{cases}$$

Wir beweisen per Induktion $T(n) = c \cdot n \cdot \log n + c \cdot n \in \mathcal{O}(n \log n)$.

- Induktionsanfang: Für $n = 1$ gilt $T(n) = c = c \cdot \log(1) + c$.
- Induktionsschritt: Wir betrachten $n \geq 2$. Dann ist

$$\begin{aligned} T(n) &= 2 \cdot T(\frac{n}{2}) + c \cdot n \\ &\overset{\text{I.V.}}{=} 2 \cdot \left(c \cdot \frac{n}{2} \cdot \log \frac{n}{2} + c \cdot \frac{n}{2} \right) + c \cdot n \\ &= c \cdot n \cdot (\log n - 1) + c \cdot n + c \cdot n \\ &= c \cdot n \cdot \log n + c \cdot n \end{aligned}$$

Insertionsort vs. Mergesort

- Laufzeit Insertionsort $\mathcal{O}(n^2)$.

Insertionsort vs. Mergesort

- Laufzeit Insertionsort $\mathcal{O}(n^2)$.
- Laufzeit Mergesort $\mathcal{O}(n \log n)$.

Insertionsort vs. Mergesort

- Laufzeit Insertionsort $\mathcal{O}(n^2)$.
- Laufzeit Mergesort $\mathcal{O}(n \log n)$.
- Für große Eingaben ist Mergesort wesentlich schneller als Insertionsort.

Insertionsort vs. Mergesort

- Laufzeit Insertionsort $\mathcal{O}(n^2)$.
- Laufzeit Mergesort $\mathcal{O}(n \log n)$.
- Für große Eingaben ist Mergesort wesentlich schneller als Insertionsort.
- Aber die Konstanten in der Laufzeit von Mergesort sind größer: Arrays kopieren, Rekursion hat weiteren Overhead.

Insertionsort vs. Mergesort

- Laufzeit Insertionsort $\mathcal{O}(n^2)$.
- Laufzeit Mergesort $\mathcal{O}(n \log n)$.
- Für große Eingaben ist Mergesort wesentlich schneller als Insertionsort.
- Aber die Konstanten in der Laufzeit von Mergesort sind größer: Arrays kopieren, Rekursion hat weiteren Overhead.
- In der Praxis: Insertionsort bis ca. $n = 50$ schneller als Mergesort.

Das Master Theorem

Auflösen von Rekursionsgleichungen

Rekursionsgleichungen auflösen

Verschiedene Methoden:

Rekursionsgleichungen auflösen

Verschiedene Methoden:

- Subsitutionsmethode: Lösung raten und per Induktion Korrektheit beweisen.

Rekursionsgleichungen auflösen

Verschiedene Methoden:

- Subsitutionsmethode: Lösung raten und per Induktion Korrektheit beweisen.
- Rekursionsbaum analysieren.

Rekursionsgleichungen auflösen

Verschiedene Methoden:

- Subsitutionsmethode: Lösung raten und per Induktion Korrektheit beweisen.
- Rekursionsbaum analysieren.
- Mastertheorem für Rekursionsgleichungen der Form

$$T(n) = a \cdot T(\frac{n}{b}) + f(n).$$

Mastertheorem

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

Mastertheorem

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

Mastertheorem

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

Mastertheorem

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

Mastertheorem für Mergesort

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + cn & \text{sonst.} \end{cases}$$

Mastertheorem für Mergesort

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + cn & \text{sonst.} \end{cases}$$

- $a = 2, b = 2, \log_b a = 1$.

Mastertheorem für Mergesort

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + cn & \text{sonst.} \end{cases}$$

- $a = 2, b = 2, \log_b a = 1$.
- $f(n) = cn \in \Theta(n^{\log_b a})$.

Mastertheorem für Mergesort

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

Mergesort

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2 \cdot T(\frac{n}{2}) + cn & \text{sonst.} \end{cases}$$

- $a = 2, b = 2, \log_b a = 1$.
- $f(n) = cn \in \Theta(n^{\log_b a})$.
- Es gilt nach Mastertheorem $T(n) \in \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$.

Mastertheorem Beispielanwendungen I

Mastertheorem (Teil 1)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$.

Mastertheorem Beispielanwendungen I

Mastertheorem (Teil 1)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$.
- $a = 9, b = 3, \log_b a = 2$.

Mastertheorem Beispielanwendungen I

Mastertheorem (Teil 1)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$.
- $a = 9, b = 3, \log_b a = 2$.
- $f(n) = n \in \mathcal{O}(n^{\log_b a - \epsilon})$ für $\epsilon = 1$.

Mastertheorem Beispielanwendungen I

Mastertheorem (Teil 1)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$.
- $a = 9, b = 3, \log_b a = 2$.
- $f(n) = n \in \mathcal{O}(n^{\log_b a - \epsilon})$ für $\epsilon = 1$.
- Es gilt nach Mastertheorem $T(n) \in \Theta(n^2)$.

Mastertheorem Beispielanwendungen I

Mastertheorem (Teil 1)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$.
- $a = 9, b = 3, \log_b a = 2$.
- $f(n) = n \in \mathcal{O}(n^{\log_b a - \epsilon})$ für $\epsilon = 1$.
- Es gilt nach Mastertheorem $T(n) \in \Theta(n^2)$.

Mastertheorem Beispielanwendungen II

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- $T(n) = T(2n/3) + 1$.

Mastertheorem Beispielanwendungen II

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- $T(n) = T(2n/3) + 1$.
- $a = 1, b = 3/2, \log_b a = 0$.

Mastertheorem Beispielanwendungen II

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- $T(n) = T(2n/3) + 1$.
- $a = 1, b = 3/2, \log_b a = 0$.
- $f(n) \in \Theta(n^0) = \Theta(1)$.

Mastertheorem Beispielanwendungen II

Mastertheorem (Teil 2)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- $T(n) = T(2n/3) + 1$.
- $a = 1, b = 3/2, \log_b a = 0$.
- $f(n) \in \Theta(n^0) = \Theta(1)$.
- Es gilt nach Mastertheorem $T(n) \in \Theta(\log n)$.

Mastertheorem Beispielanwendungen III

Mastertheorem (Teil 3)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \log n$.

Mastertheorem Beispielanwendungen III

Mastertheorem (Teil 3)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \log n$.
- $a = 3, b = 4, \log_4 3 \approx 0.793$.

Mastertheorem Beispielanwendungen III

Mastertheorem (Teil 3)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \log n$.
- $a = 3, b = 4, \log_4 3 \approx 0.793$.
- $f(n) \in \Omega(n^{\log_4 3 + \epsilon})$, für $\epsilon \approx 0.2$.

Mastertheorem Beispielanwendungen III

Mastertheorem (Teil 3)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \log n$.
- $a = 3, b = 4, \log_4 3 \approx 0.793$.
- $f(n) \in \Omega(n^{\log_4 3 + \epsilon})$, für $\epsilon \approx 0.2$.
- Für großes n gilt
  $af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n \log n = cf(n)$ für $c = 3/4$.

Mastertheorem Beispielanwendungen III

Mastertheorem (Teil 3)

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \log n$.
- $a = 3, b = 4, \log_4 3 \approx 0.793$.
- $f(n) \in \Omega(n^{\log_4 3 + \epsilon})$, für $\epsilon \approx 0.2$.
- Für großes n gilt
  $af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n \log n = cf(n)$ für $c = 3/4$.
- Es gilt nach Mastertheorem $T(n) \in \Theta(n \log n)$.

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.
- $f(n) \in \Omega(n^{\log_b a})$

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.
- $f(n) \in \Omega(n^{\log_b a})$ aber $f(n) \notin \Omega(n^{\log_b a + \epsilon})$ für alle $\epsilon > 0$.

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.
- $f(n) \in \Omega(n^{\log_b a})$ aber $f(n) \notin \Omega(n^{\log_b a + \epsilon})$ für alle $\epsilon > 0$.
- $f(n) \notin \mathcal{O}(n^{\log_b a - \epsilon})$ für alle $\epsilon > 0$

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.
- $f(n) \in \Omega(n^{\log_b a})$ aber $f(n) \notin \Omega(n^{\log_b a + \epsilon})$ für alle $\epsilon > 0$.
- $f(n) \notin \mathcal{O}(n^{\log_b a - \epsilon})$ für alle $\epsilon > 0$
- $f(n) \notin \Theta(n^{\log_b a})$

Mastertheorem Beispielanwendungen IV

- $T(n) = 2T(n/2) + n \log n$.
- $a = 2, b = 2, \log_b a = 1$.
- $f(n) \in \Omega(n^{\log_b a})$ aber $f(n) \notin \Omega(n^{\log_b a + \epsilon})$ für alle $\epsilon > 0$.
- $f(n) \notin \mathcal{O}(n^{\log_b a - \epsilon})$ für alle $\epsilon > 0$
- $f(n) \notin \Theta(n^{\log_b a})$
- Also ist das Mastertheorem nicht anwendbar!

Mastertheorem

Seien $a \geq 1, b > 1$ Konstanten, $f : \mathbb{N} \to \mathbb{N}$ und $T(n) = aT(\frac{n}{b}) + f(n)$.

- Wenn $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ für ein $\epsilon > 0$, dann gilt

$$T(n) \in \Theta(n^{\log_b a}).$$

- Wenn $f(n) \in \Theta(n^{\log_b a})$, dann gilt

$$T(n) \in \Theta(n^{\log_b a} \log n).$$

- Wenn $f(n) \in \Omega(n^{\log_b a + \epsilon})$ für ein $\epsilon > 0$, und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n, dann gilt

$$T(n) \in \Theta(f(n)).$$

Gibt es schnellere Sortieralgorithmen?

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?
- Ohne zusätzliche Strukturinformation nicht!

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?
- Ohne zusätzliche Strukturinformation nicht!
- Untere Schranke für die Laufzeit vergleichsbasierter Sortieralgorithmen $\Omega(n \log n)$.

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?
- Ohne zusätzliche Strukturinformation nicht!
- Untere Schranke für die Laufzeit vergleichsbasierter Sortieralgorithmen $\Omega(n \log n)$.
  - Vergleichsbasiert: Informationen können nur durch Vergleiche von jeweils zwei Objekten gewonnen werden.

Untere Schranken für Sortieralgorithmen

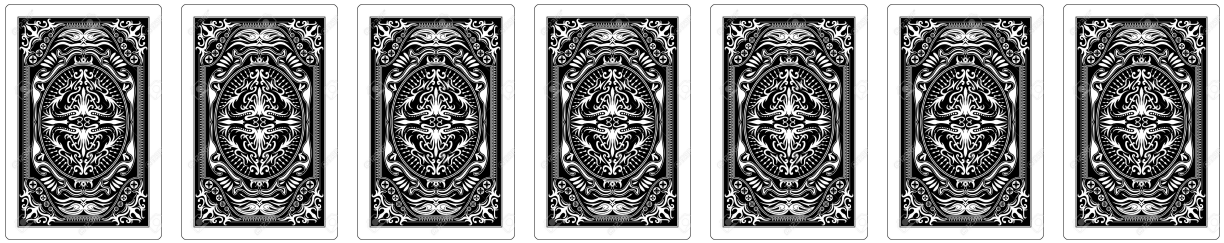
- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?
- Ohne zusätzliche Strukturinformation nicht!
- Untere Schranke für die Laufzeit vergleichsbasierter Sortieralgorithmen $\Omega(n \log n)$.
  - Vergleichsbasiert: Informationen können nur durch Vergleiche von jeweils zwei Objekten gewonnen werden.
  - Mögliche zusätzliche Strukturinformation: es sollen Zahlen aus einem festen Zahlenbereich sortiert werden.

Untere Schranken für Sortieralgorithmen

- Insertionsort, ... laufen in Zeit $\mathcal{O}(n^2)$.
- Mergesort, Quicksort, ... laufen in Zeit $\mathcal{O}(n \log n)$.
- Geht es noch schneller?
- Ohne zusätzliche Strukturinformation nicht!
- Untere Schranke für die Laufzeit vergleichsbasierter Sortieralgorithmen $\Omega(n \log n)$.
  - Vergleichsbasiert: Informationen können nur durch Vergleiche von jeweils zwei Objekten gewonnen werden.
  - Mögliche zusätzliche Strukturinformation: es sollen Zahlen aus einem festen Zahlenbereich sortiert werden.
  - Beispiel: Bucketsort $\mathcal{O}(n + r)$ falls Zahlenbereich $\{1, \ldots, r\}$.

Intuition für Vergleichsbasierte Algorithmen

Vergleichsbasierte Sortieralgorithmen können wir uns wie folgt vorstellen.



- Karten mit Zahlen beschriftet verdeckt auf dem Tisch.
- Wir können einen Helfer fragen, ob Karte an Position i größer ist als Karte an Position j.
- Wir erfahren dabei nicht den Wert der Karten.
- Wir wollen Karten durch Vertauschen sortieren.
- Wie viele Anfragen an den Helfer brauchen wir?

Untere Schranke Beweis

Satz

Jeder vergleichsbasierte Sortieralgorithmus benötigt zum Sortieren von Arrays der Länge n im Worst Case $\Omega(n \log n)$ Vergleiche. Damit ist seine Worst-Case-Laufzeit $\Omega(n \log n)$.

Untere Schranke Beweis

Satz

Jeder vergleichsbasierte Sortieralgorithmus benötigt zum Sortieren von Arrays der Länge n im Worst Case $\Omega(n \log n)$ Vergleiche. Damit ist seine Worst-Case-Laufzeit $\Omega(n \log n)$.

- Wir beweisen die Aussage nur für deterministische Algorithmen (die keine zufälligen Entscheidungen treffen).

Untere Schranke Beweis

Satz

Jeder vergleichsbasierte Sortieralgorithmus benötigt zum Sortieren von Arrays der Länge n im Worst Case $\Omega(n \log n)$ Vergleiche. Damit ist seine Worst-Case-Laufzeit $\Omega(n \log n)$.

- Wir beweisen die Aussage nur für deterministische Algorithmen (die keine zufälligen Entscheidungen treffen).
- Sie gilt aber auch für randomisierte Algorithmen.

Untere Schranke Beweis

Satz

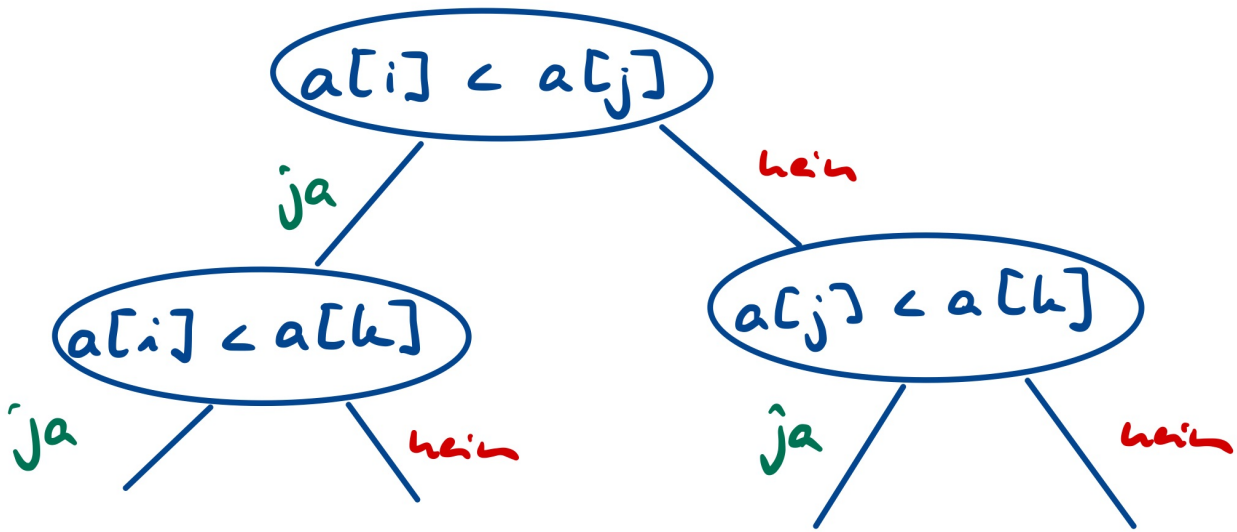
Jeder vergleichsbasierte Sortieralgorithmus benötigt zum Sortieren von Arrays der Länge n im Worst Case $\Omega(n \log n)$ Vergleiche. Damit ist seine Worst-Case-Laufzeit $\Omega(n \log n)$.

- Wir beweisen die Aussage nur für deterministische Algorithmen (die keine zufälligen Entscheidungen treffen).
- Sie gilt aber auch für randomisierte Algorithmen.
- O.B.d.A. nehmen wir an, dass keine Zahl in der Eingabe mehrmals vorkommt.

Beweis der unteren Schranke

Betrachte vergleichsbasierte Sortieralgorithmen:

- abstrahiere von allen Operationen außer Vergleichen
- repräsentiere Algorithmus als Entscheidungsbaum
  - jeder Knoten repräsentiert einen Vergleich
  - Kinder repräsentieren Algorithmusentscheidung je nach Ergebnis des Vergleichs
- Sortieren verschiedener Eingabe-Arrays ergibt verschiedene Pfade im Baum.



Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt $=$ maximale Baumhöhe

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt = maximale Baumhöhe

- Anzahl Blätter = Anzahl Permutationen von n Zahlen = $n!$

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt = maximale Baumhöhe

- Anzahl Blätter = Anzahl Permutationen von n Zahlen = $n!$
- Innere Knoten haben Grad 2 (binäre Vergleiche)

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt = maximale Baumhöhe

- Anzahl Blätter = Anzahl Permutationen von n Zahlen = $n!$
- Innere Knoten haben Grad 2 (binäre Vergleiche)
- Für Baumhöhe h gilt $2^h \geq n!$;

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt = maximale Baumhöhe

- Anzahl Blätter = Anzahl Permutationen von n Zahlen = $n!$
- Innere Knoten haben Grad 2 (binäre Vergleiche)
- Für Baumhöhe h gilt $2^h \geq n!$;Also $h \geq \log_2(n!) = \Omega(n \log n)$.

Beweis der unteren Schranke II

Für jeden Knoten betrachte alle Permutationen des Inputs, die an diesem Knoten realisiert sein können.

(a) Wenn es ≥ 2 mögliche Permutationen gibt, dann ist mindestens ein weiterer Vergleich nötig.

(b) Wenn nur eine Permutation übrig (Blatt), Permutation ausgeben.

Worst case: längster Weg von Wurzel zu Blatt = maximale Baumhöhe

- Anzahl Blätter = Anzahl Permutationen von n Zahlen = $n!$
- Innere Knoten haben Grad 2 (binäre Vergleiche)
- Für Baumhöhe h gilt $2^h \geq n!$;Also $h \geq \log_2(n!) = \Omega(n \log n)$.

$$\log_2(n!) \geq \log_2(n(n-1)(n-2)\cdots n/2)$$

$$= \sum_{k=1}^{n/2} \log_2(n-k) \geq (n/2)\log_2(n/2) \in \Omega(n \log n).$$

□

Zusammenfassung

- MergeSort hat Laufzeit $\mathcal{O}(n \log n)$

Zusammenfassung

- MergeSort hat Laufzeit $\mathcal{O}(n \log n)$
- Rekursionsgleichungen

Zusammenfassung

- MergeSort hat Laufzeit $\mathcal{O}(n \log n)$
- Rekursionsgleichungen
- Master Theorem

Zusammenfassung

- MergeSort hat Laufzeit $\mathcal{O}(n \log n)$
- Rekursionsgleichungen
- Master Theorem
- Untere Schranke von $\Omega(n \log n)$ an die Laufzeit von vergleichsbasierten Sortierverfahren