Wiederholung - Average-Case Laufzeit

Average-case Laufzeit:

- Betrachten alle Permutationen der *n* Eingabezahlen.
- Berechnen für jede Permutation Laufzeit des Algorithmus bei dieser Permutation.
- Average-case Laufzeit ist dann der Durchschnitt über all diese Laufzeiten.
- Average-case Laufzeit ist die erwartete Laufzeit einer zufällig und gleichverteilt gewählten Permutation aus der Menge aller Permutationen der *n* Eingabezahlen.

Average-Case – Insertion-Sort

Satz 4.8: Insertion-Sort besitzt average-case Laufzeit $\Theta(n^2)$.

Beweis: Wir zeigen, mit Wahrscheinlichkeit 1/2 hat man mindestens $n^2/16$ Vergleiche (Annahme: n ist gerade).

- Sei $L_{n/2}$ die Menge der $n/2$ kleinsten Zahlen
- Sei $U_{n/2}$ die Menge der $n/2$ größten Zahlen
- A_i: Ereignis, dass in einer zufälligen Permutation der n Zahlen genau i Elemente aus $U_{n/2}$ in der ersten Hälfte der Permutation platziert sind.
- B_i: Ereignis, dass in einer zufälligen Permutation der n Zahlen genau i Elemente aus $L_{n/2}$ in der ersten Hälfte der Permutation platziert sind.

Average-Case – Insertion-Sort

5 2 4 7 1 3 2 6

$L_{n/2}$ 1 2 2 3 **$U_{n/2}$ 4 5 6 7**

Ereignis A_3

- $\Pr[\bigcup_{n/4 \leq i \leq n/2} A_i] = \sum_{n/4 \leq i \leq n/2} \Pr[A_i]$
- $\Pr[A_i] = \Pr[B_i] \rightarrow \Pr[A_i] = \Pr[A_{n/2-i}]$
- $\sum_{0 \leq i \leq n/2} \Pr[A_i] = 1$
- $\sum_{n/4 \leq i \leq n/2} \Pr[A_i] = 1 - \sum_{0 \leq i < n/4} \Pr[A_i] = \Pr[A_{n/4}] + 1 - \sum_{n/4 \leq i \leq n/2} \Pr[A_i]$
- $2 \sum_{n/4 \leq i \leq n/2} \Pr[A_i] = 1 + \Pr[A_{n/4}] > 1 \rightarrow \sum_{n/4 \leq i \leq n/2} \Pr[A_i] > ½$
- Mit Wahrscheinlichkeit 1/2 befindet sich mindestens die Hälfte von $U_{n/2}$ in der ersten Hälfte und mindestens die Hälfte von $L_{n/2}$ in der zweiten Hälfte einer zufälligen Permutation.

Average-Case Laufzeit

InsertionSort(Array A)

1. **for** j ← 2 **to** length(A) **do**
2. key ← A[j]
3. i ← j-1
4. **while** i>0 and A[i]>key **do**
5. A[i+1] ← A[i]
6. i ← i-1
7. A[i+1] ← key

Mit Wahrscheinlichkeit ½ gibt es mindestens $n/4$ Elemente A[j] in $L_{n/2}$ mit j ≥ $n/2$

Sei j ≥ $n/2$ und A[j] in $L_{n/2}$
Dann wird die **while**-Schleife mindestens $n/4$ mal durchlaufen.

$n^2/16$ Vergleiche

Im Durchschnitt produziert Insertion-Sort mehr als $n^2/32$ Vergleiche

5. Divide & Conquer – Quicksort

- Quicksort ist wie Merge-Sort ein auf dem Divide&Conquer-Prinzip beruhender Sortieralgorithmus.

- Von Quicksort existieren unterschiedliche Varianten, von denen einige in der Praxis besonders effizient sind.

- Die worst-case Laufzeit von Quicksort ist $\Theta(n^2)$.

- Die durchschnittliche Laufzeit ist jedoch $\Theta(n\log(n))$.

- Eine randomisierte Version von Quicksort besitzt erwartete Laufzeit $\Theta(n\log(n))$.

Quicksort - Idee

Eingabe: Ein zu sortierendes Teilarray $A[p\ldots r]$.

Teilungsschritt: Berechne einen Index $q, p \leq q \leq r$ und vertausche die Reihenfolge der Elemente in $A[p\ldots r]$, so dass die Element in $A[p\ldots q-1]$ nicht größer und die Elemente in $A[q+1\ldots r]$ nicht kleiner sind als $A[q]$.

Eroberungsschritt: Sortiere rekursiv die beiden Teilarrays $A[p\ldots q-1]$ und $A[q+1\ldots r]$.

Kombinationsschritt: Entfällt, da nach Eroberungsschritt das Array $A[p\ldots r]$ bereits sortiert ist.

Quicksort - Pseudocode

Quicksort(A,p,r)

1. **if** $p < r$
2. **then** $q \leftarrow$ Partition(A,p,r)
3. Quicksort$(A,p,q\text{-}1)$
4. Quicksort$(A,q+1,r)$

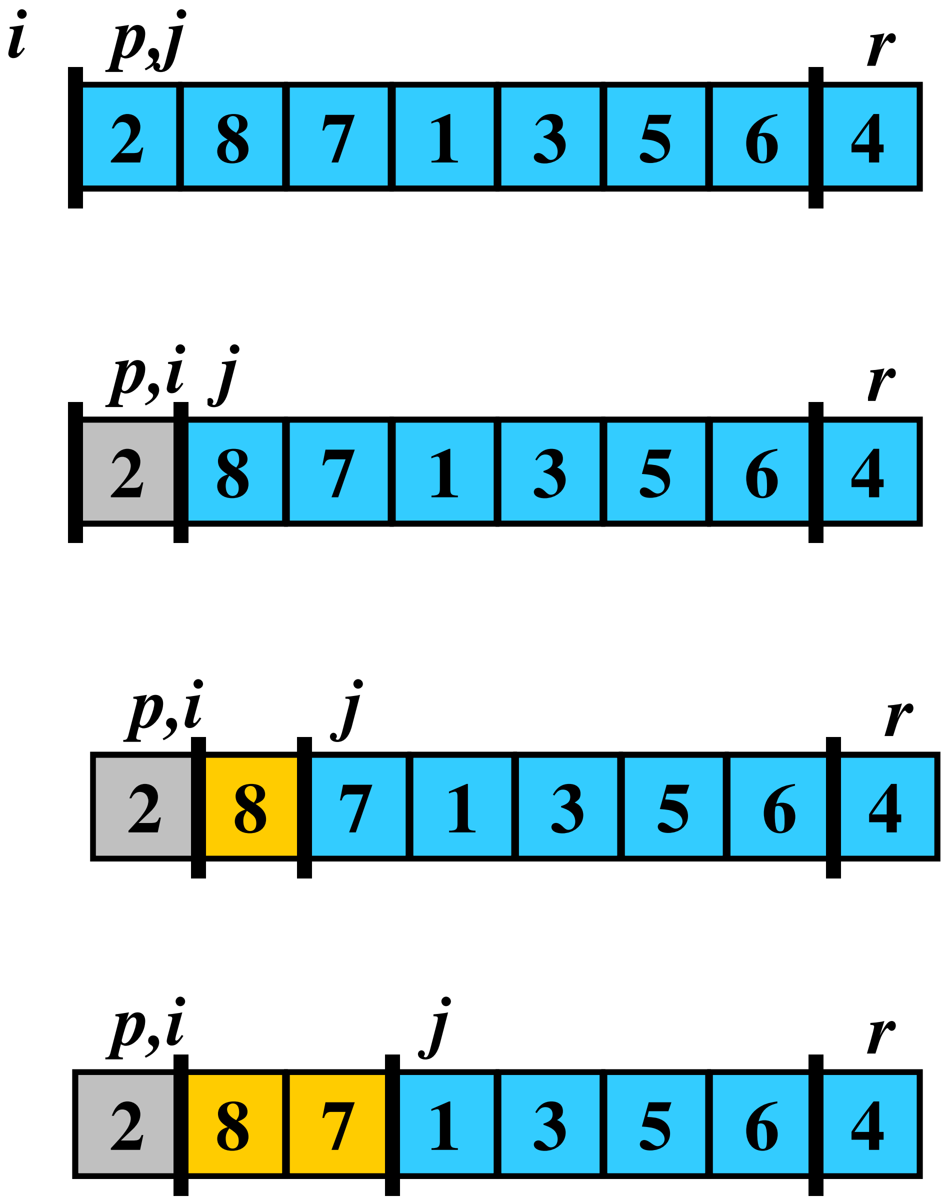
Aufruf, um Array A zu sortieren: Quicksort$(A,1,length[A])$

Partition - Pseudocode

Partition(A,p,r)

1. $x \leftarrow A[r]$
2. $i \leftarrow p$-1
3. **for** $j \leftarrow p$ **to** r-1
4.   **do if** $A[j] \leq x$
5.   **then** $i \leftarrow i+1$
6. $A[i] \leftrightarrow A[j]$
7. $A[i+1] \leftrightarrow A[r]$
8. **return** $i+1$

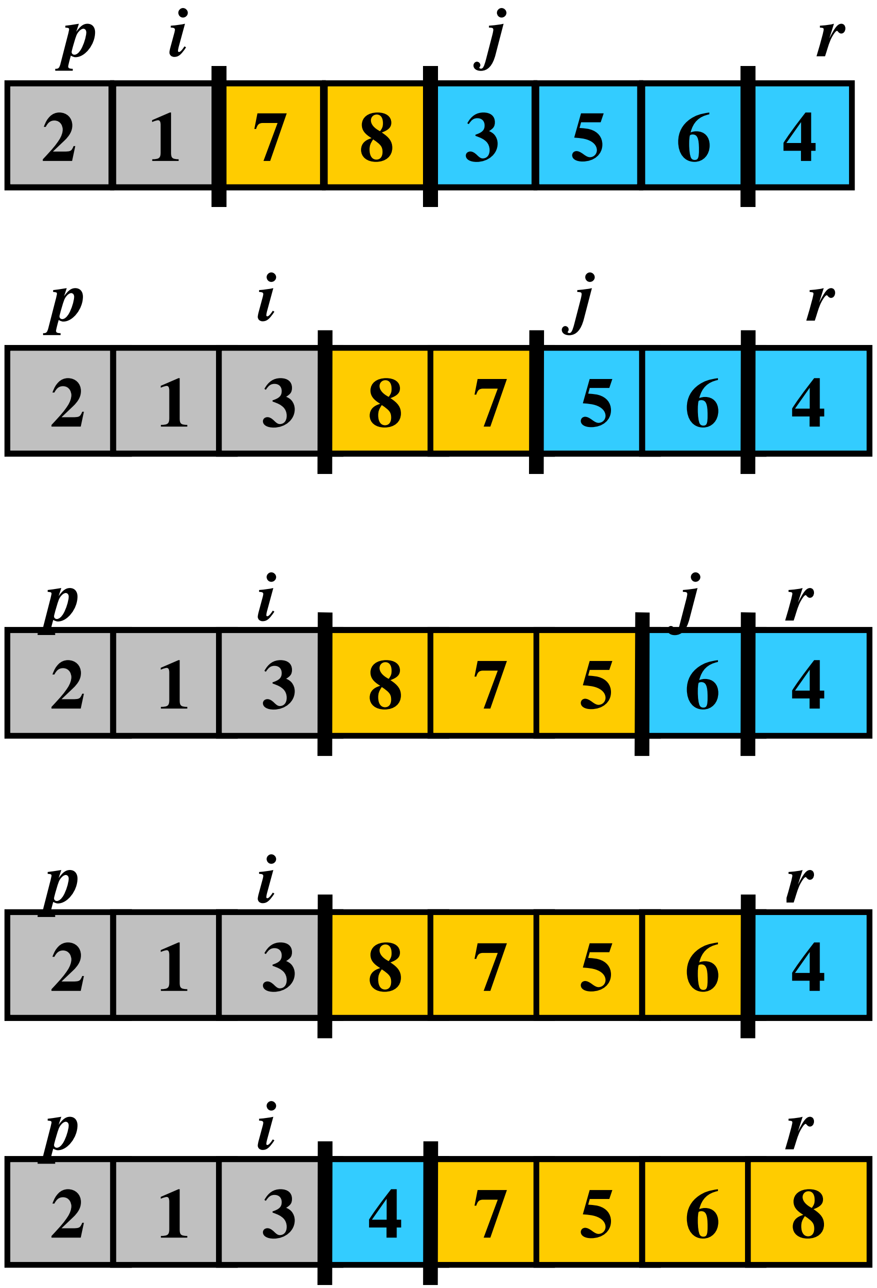
Illustration von Partition (1)



Partition(A,p,r)

1. $x \leftarrow A[r]$
2. $i \leftarrow p$-1
3. **for** $j \leftarrow p$ **to** r-1
4.   **do if** $A[j] \leq x$
5.   **then** $i \leftarrow i+1$
6. $A[i] \leftrightarrow A[j]$
7. $A[i+1] \leftrightarrow A[r]$
8. **return** $i+1$

Illustration von Partition (2)



Partition(A,p,r)

```
x ← A[r]
i ← p-1
for j ← p to r-1
      do if A[j] ≤ x
            then i ← i+1
                  A[i] ↔ A[j]
A[i+1] ↔ A[r]
return i+1
```

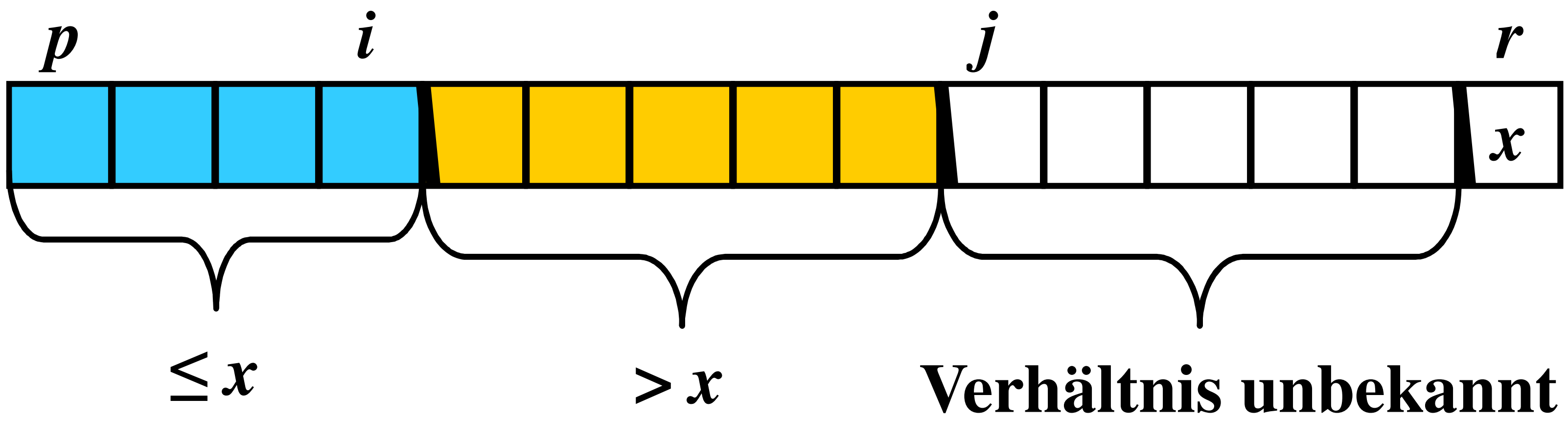
Korrektheit von Partition - Invariante

Invariante: Vor Durchlauf der Schleife in Zeilen 3-6 mit Index *j* gilt für jeden Index *k*:

1. Falls $p \le k \le i$, dann ist $A[k] \le x$.

2. Falls $i+1 \le k \le j-1$, dann ist $A[k] > x$.

3. Falls $k = r$, dann ist $A[k] = x$.



Korrektheit von Partition (1)

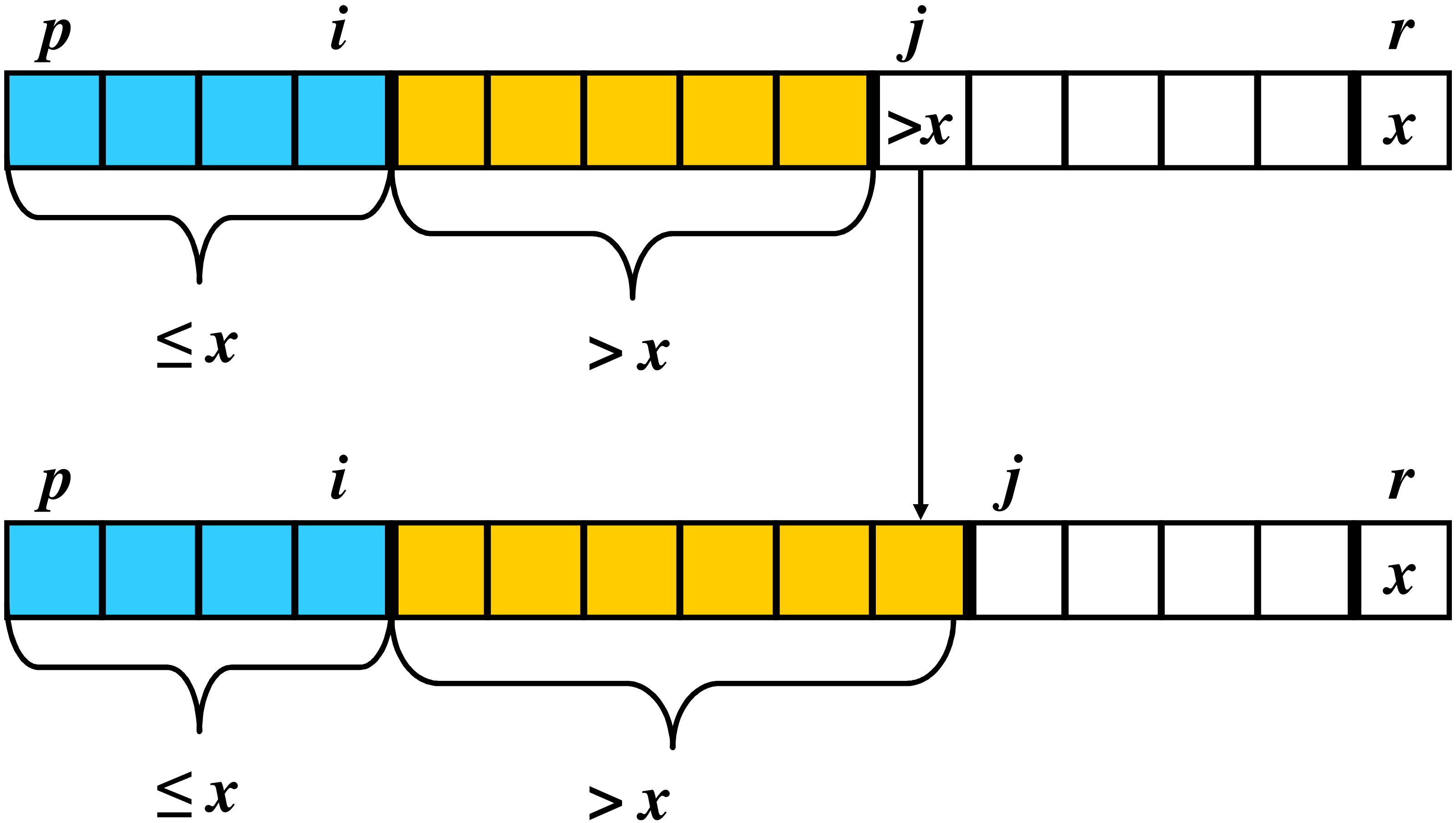
Initialisierung: Vor dem ersten Schleifendurchlauf gilt $i = p-1$ und $j = p$. Daher gibt es in diesem Fall keine Indizes zwischen p und i (1.Bedingung) bzw. zwischen $i+1$ und $j-1$ (2.Bedingung). Die erste Zeile sorgt dafür, dass die 3.Bedingung ebenfalls erfüllt ist.

Erhaltung: Unterscheiden zwei Fälle

1. $A[j] > x$
2. $A[j] \leq x$.

1. Fall: Nur j wird erhöht. Damit wird dann 2.Bedingung auch für $k=j$ erfüllt.

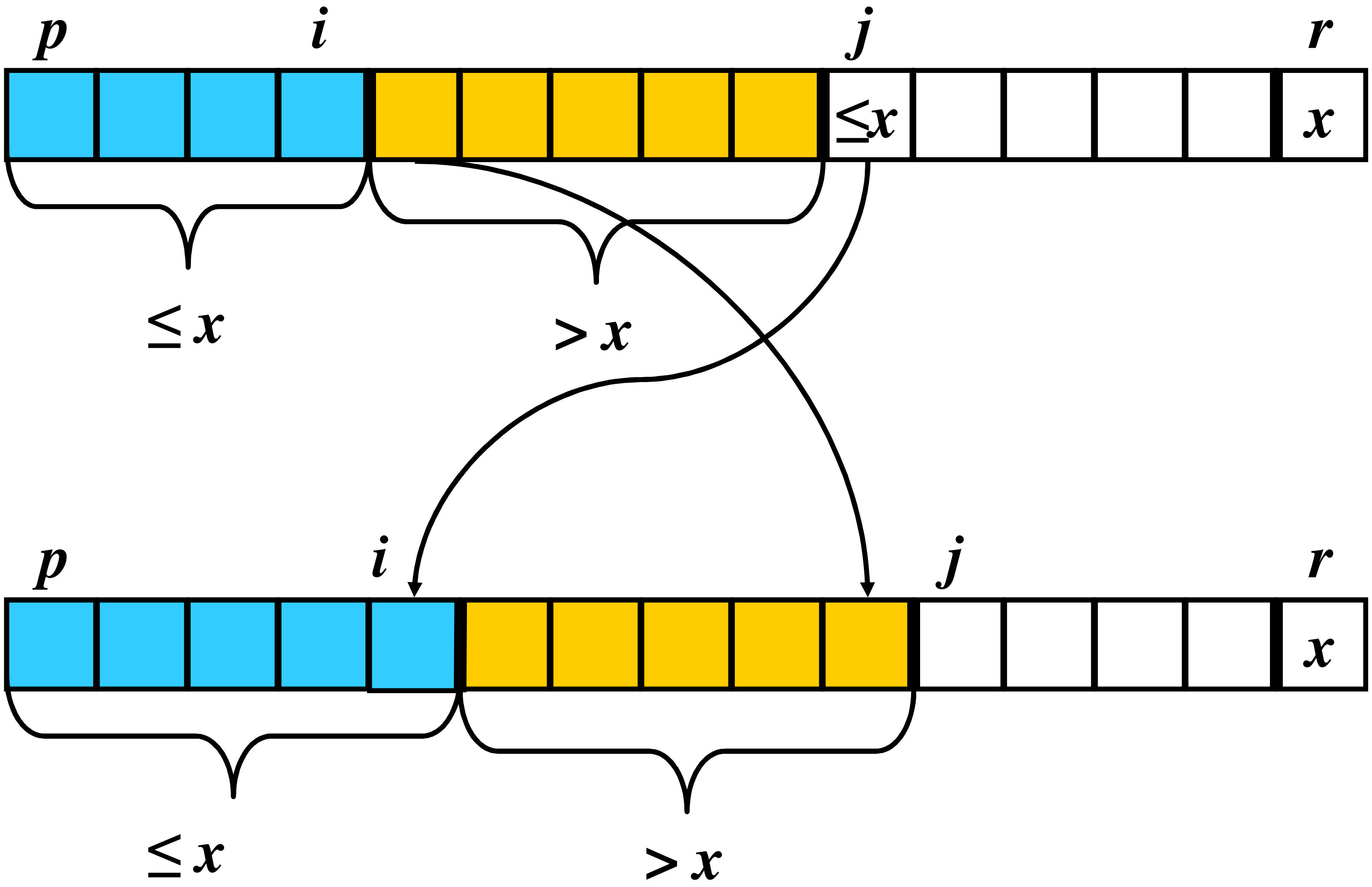
Erhaltung – 1.Fall



Korrektheit von Partition (2)

2. Fall $A[j] \leq x$*:* Element $A[j]$ kommt an Position i. Da $A[j] \leq x$ ist 1.Bedingung weiter erfüllt. Für neues $A[j-1]$ gilt nach Voraussetzung $A[j-1] > x$. Damit ist 2. Bedingung erfüllt auch.

Erhaltung – 2.Fall



Korrektheit von Partition (3)

Terminierung: Es gilt $j=r$ und alle Elemente des Arrays wurden mit x verglichen. Zeile 9 stellt nun sicher, dass x zwischen die Elemente echt kleiner als x und die Element echt größer als x plaziert wird. Damit genügt die von Partition berechnet Aufteilung immer den Anforderungen von Quicksort.

Laufzeit von Partition

Partition(A,p,r)

1. $x \leftarrow A[r]$
2. $i \leftarrow p\text{-}1$
3. **for** $j \leftarrow p$ **to** r-1
4. **do if** $A[j] \leq x$
5. **then** $i \leftarrow i+1$
6. $A[i] \leftrightarrow A[j]$
7. $A[i+1] \leftrightarrow A[r]$
8. **return** $i+1$

- **Pro Zeile konstante Zeit.**
- **Schleife Zeilen 3-6 wird *n*=*r*-*p*-mal durchlaufen.**

Satz 5.1: **Partition hat Laufzeit $\Theta(n)$ bei Eingabe eines Teilarrays mit n Elementen.**

Quicksort – Analyse

Satz 5.2: Es gibt ein $c>0$, so dass für alle n und alle Eingaben der Größe n Quicksort mindestens Laufzeit $cn\log(n)$ besitzt.

Satz 5.3: Quicksort besitzt worst-case Laufzeit $\Theta(n^2)$.

Satz 5.4: Quicksort besitzt *average-case* Laufzeit $O(n\log(n))$.

Average-case Laufzeit: Betrachten alle Permutationen der n Eingabezahlen. Berechnen für jede Permutation Laufzeit von Quicksort bei dieser Permutation. Average-case Laufzeit ist dann der Durchschnitt über all diese Laufzeiten.

Quicksort – Analyse

- Linearität des Erwartungswertes: Seien $X : S_1 \to N$, $Y : S_2 \to N$ zwei Zufallsvariablen. Dann gilt:

$$\sum_{x \in N} x \Pr[X + Y = x] = E[X + Y] = E[X] + E[Y].$$

- Sei X_n die Zufallsvariable, die die Laufzeit (Anzahl der Vergleiche) von Quicksort für eine zufällige Permutation von $\{1, \ldots, n\}$ angibt. Sei A_i das Ereignis $A[n] = i$.

Quicksort – Analyse

- $E[X_n] = \sum_{x \in N} x \cdot \Pr[X_n = x]$
- $= \sum_{x \in N} x \sum_{i=1}^{n} \Pr[A_i] \cdot \Pr[X_{i-1} + X_{n-i} = x - (n-1)]$
- $= \frac{1}{n} \sum_{i=1}^{n} \sum_{x \in N} (x - (n-1) + (n-1)) \cdot Pr[X_{i-1} + X_{n-i} = x - (n-1)]$
- $= \frac{1}{n} \sum_{i=1}^{n} \Bigg(\underbrace{\sum_{x \in N} (x - (n-1)) \cdot Pr[X_{i-1} + X_{n-i} = x - (n-1)]}_{E[X_{i-1}+X_{n-i}]} +$
- $\underbrace{(n-1) \sum_{x \in N} Pr[X_{i-1} + X_{n-i} = x - (n-1)] \Bigg)}_{=1}$
- $= \frac{1}{n} \sum_{i=1}^{n} (E[X_{i-1}] + E[X_{n-i}]) + (n-1)$

Quicksort – Analyse

- Sei $Q_E(n)$ die erwartete Laufzeit von Quicksort für eine zufällige Permutation der Länge n, wobei alle Permutationen gleichwahrscheinlich sind.
- Die Zahl $A[n]$ ist die i kleinste Zahl mit Wahrscheinlichkeit $1/n$ für alle $i \in \{1, \ldots, n\}$.
- $Q_E(n) \leq \frac{1}{n} \sum_{i=1}^{n} (Q_E(i-1) + Q_E(n-i)) + cn$
- $\sum_{i=1}^{n} Q_E(i-1) = \sum_{k=0}^{n-1} Q_E(k) = \sum_{i=1}^{n} Q_E(n-i)$

Quicksort – Analyse

- $Q_E(n) \leq \frac{2}{n} \sum_{k=0}^{n-1} Q_E(k) + cn$
- $nQ_E(n) \leq 2 \sum_{k=0}^{n-1} Q_E(k) + cn^2$
- $(n-1)Q_E(n-1) \leq 2 \sum_{k=0}^{n-2} Q_E(k) + c(n-1)^2$
- $nQ_E(n) - (n-1)Q_E(n-1) \leq 2Q_E(n-1) + c(2n-1)$
- $nQ_E(n) = (n+1)Q_E(n-1) + c(2n-1)$

Quicksort – Analyse

- $\frac{Q_E(n)}{n+1} \leq \frac{Q_E(n-1)}{n} + c\,\frac{2n-1}{n(n+1)} \leq \frac{Q_E(n-1)}{n} + \frac{2c}{n}$
- $\frac{Q_E(n)}{n+1} \leq \frac{Q_E(n-2)}{n-1} + \frac{2c}{n} + \frac{2c}{n-1} \leq \cdots \leq \frac{Q_E(1)}{2} + 2c\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$
- Wir wissen $\sum_{i=2}^{n} \leq \ln(n)$. Dann gilt

$$\frac{Q_E(n)}{n+1} \leq \frac{Q_E(1)}{2} + 2c\ln(n) \leq 2c(\ln(n) + 1)$$

Randomisiertes Quicksort (1)

- Schlechte Eingaben für Quicksort können vermieden werden durch *Randomisierung*, d.h. der Algorithmus wirft gelegentlich eine Münze, um sein weiteres Vorgehen zu bestimmen.

- Worst-case Laufzeit bei ungünstigen Münzwürfen immer noch $\Theta(n^2)$.

- Es gibt keine schlechten Eingaben. Dies sind Eingaben, bei denen Quicksort bei allen Münzwürfen Laufzeit $\Theta(n^2)$ besitzt.

- Laufzeit ist in diesem Modell erwartete Laufzeit, wobei Erwartungswert über Münzwürfe genommen wird. Erwartete Laufzeit ist $\Theta(n\log(n))$.

Randomisiertes Quicksort (2)

$$\text{Randomized - Partition}(A,p,r)$$

1. $i \leftarrow \text{Random}(p,r)$
2. $A[r] \leftrightarrow A[i]$
3. $\textbf{return}\ \text{Partition}(A,p,r)$

Hierbei ist Random eine Funktion, die zufällig einen Wert aus $[\mathrm{p}\ldots\mathrm{r}]$ wählt. Dabei gilt für alle $i \in [p \ldots r]$:

$$\Pr(\text{Random}(p,r) = i) = \frac{1}{r-p+1}.$$

Randomisiertes Quicksort (3)

$\text{Randomized-Quicksort}(A,p,r)$

1. **if** $p < r$
2. **then** $q \leftarrow \text{Randomized-Partition}(A,p,r)$
3. $\text{Randomized-Quicksort}(A,p,q\text{-}1)$
4. $\text{Randomized-Quicksort}(A,q+1,r)$

Satz 5.10: Die erwartete Laufzeit von Randomized-Quicksort ist $\Theta(n\log(n))$. Dabei ist der Erwartungswert über die Zufallsexperimente in Randomized-Partition genommen.

Median-Quicksort (1)

- Verbesserung der Güte von Aufteilungen, indem nicht ein festes Element zur Aufteilung benutzt wird, sondern z.B. das mittlere von drei Elementen Zur Aufteilung benutzt wird.

- Können etwa drei zufällige Elemente wählen oder $A[p], A[q], A[r]$ mit $q := \lfloor (p+r)/2 \rfloor$.

- Beide Varianten in der Praxis erfolgreich. Aber nur zufällige Variante kann gut analysiert werden: $\Theta(n\log(n))$ erwartete Laufzeit.

Median-Quicksort (2)

$\text{Median}(A,i,j,k)$

1. **if** $(A[i] \leq A[j]) \wedge (A[k] \leq A[i])$
2. **then return** i
3. **else if** $(A[i] \leq A[j]) \wedge (A[k] \geq A[j])$
4. **then return** j
5. **else return** k

$\text{Median-Partition}(A,p,r)$

1. $i \leftarrow \text{Median}(p,\lfloor (p+r)/2 \rfloor,r)$
2. $A[r] \leftrightarrow A[i]$
3. **return** $\text{Partition}(A,p,r)$

Median-Quicksort (3)

$\text{Median-Quicksort}(A,p,r)$

1. **if** $p < r$
2.   **then** $q \leftarrow \text{Median-Partition}(A,p,r)$
3. $\text{Median-Quicksort}(A,p,q\text{-}1)$
4. $\text{Median-Quicksort}(A,q+1,r)$

6. Rekursionen

- Laufzeiten insbesondere von Divide&Conquer Algorithmen werden häufig durch Rekursionsgleichungen beschrieben.

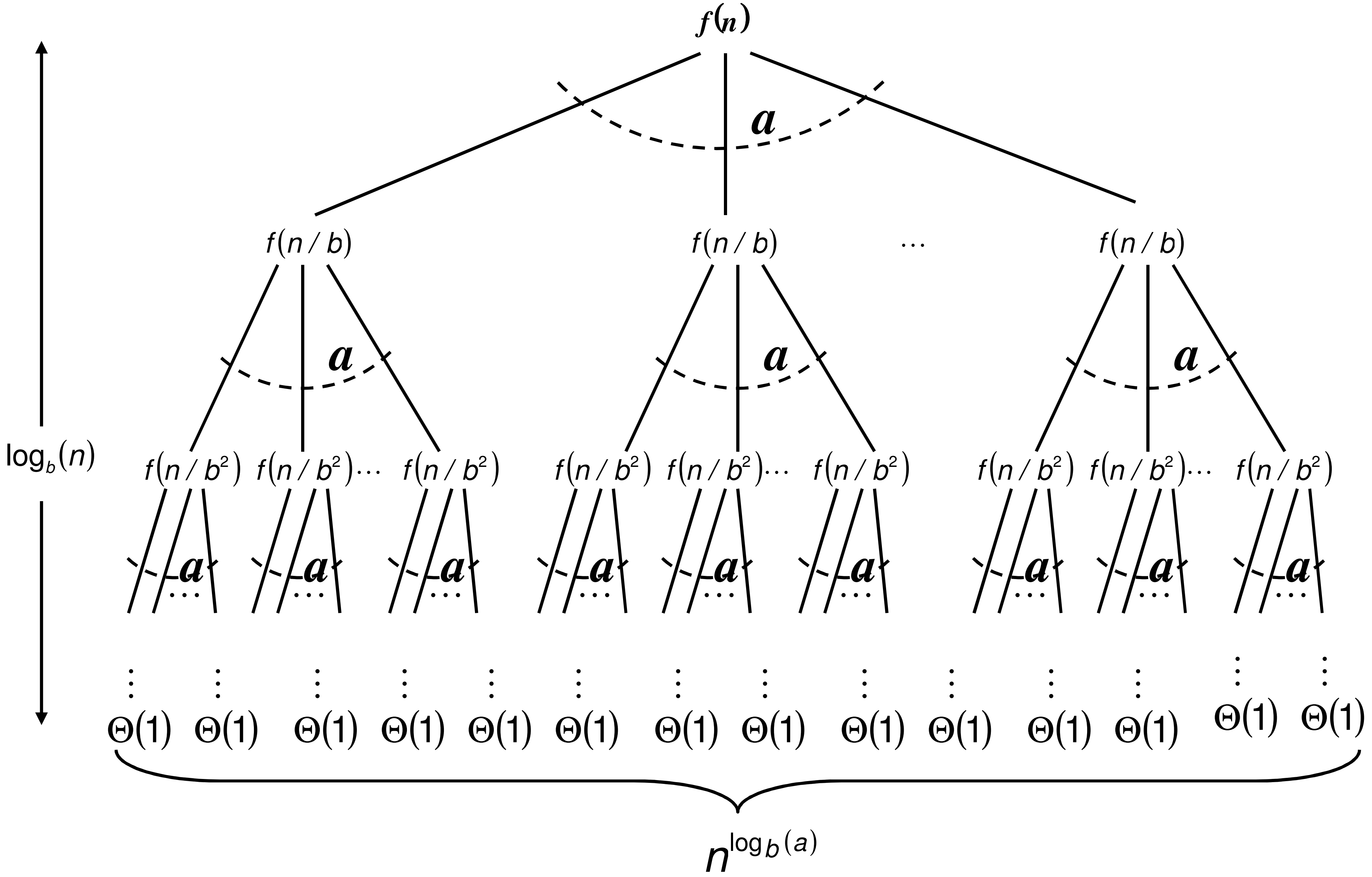
- Werden eine Methode kennenlernen, um solche Gleichungen zu lösen, die Rekursionsbaum-Methode.

- Diese Methode kann verfeinert werden, um das Master Theorem für Rekursionsgleichungen zu beweisen.

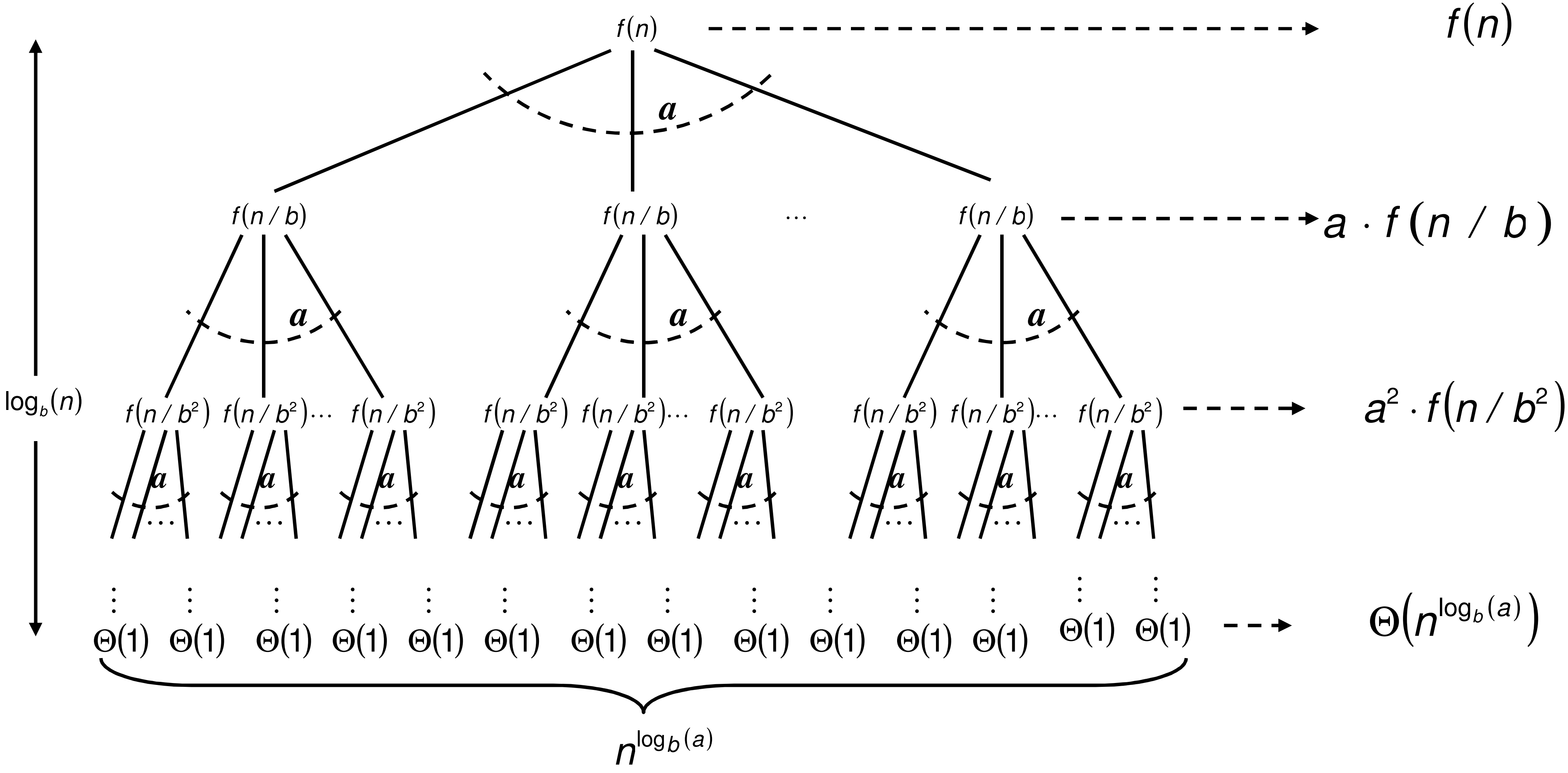
- Formulieren das Master Theorem nur.

- Anwendung des Master Theorems mit großer Vorsicht!

Rekursionsbaum für $T(n)=aT(n/b)+f(n)$



Rekursionsbaum und Summation



$$\text{Insgesamt}: \Theta\left(n^{\log_b(a)}\right) + \sum_{j=0}^{\log_b(n)-1} a^j f\left(n / b^j\right)$$

Lösen durch Rekursionsbäume - Beispiel

- Betrachten $T(n)=\begin{cases}3T(\lfloor n/4\rfloor)+cn^2, & n>1\\ c, & n=1\end{cases}, c>0.$

- Erhalten durch Ignorieren der Abrundung und mit Rekursionsbaum

$$T(n)=cn^2+\frac{3}{16}cn^2+\cdots+\left(\frac{3}{16}\right)^{\log_4(n)-1}cn^2+cn^{\log_4(3)}$$

$$=\sum_{i=0}^{\log_4(n)-1}\left(\frac{3}{16}\right)^i cn^2+cn^{\log_4(3)} .$$

$$=\frac{(3/16)^{\log_4(n)}-1}{(3/16)-1}cn^2+cn^{\log_4(3)}=O\left(n^2\right)$$

Das Master-Theorem

Satz 6.1 (Master Theorem für Rekursionsgleichungen):

Seien $a,b \geq 1$ Konstanten, sei $f(n)$ eine Funktion und sei $T(n)$ definiert durch die Rekursionsgleichung

$$T(n) = aT(n/b) + f(n).$$

Hierbei kann n/b auch durch $\lfloor n/b \rfloor$ oder $\lceil n/b \rceil$ ersetzt werden. Dann kann $T(n)$ folgendermaßen abgeschätzt werden.

1. Ist $f(n) = O\left(n^{\log_b(a)-\varepsilon}\right)$ für $\varepsilon > 0$, dann gilt $T(n) = O\left(n^{\log_b(a)}\right)$.
2. Ist $f(n) = \Theta\left(n^{\log_b(a)}\right)$, dann gilt $T(n) = O\left(n^{\log_b(a)} \cdot \log(n)\right)$.
3. Ist $f(n) = \Omega\left(n^{\log_b(a)+\varepsilon}\right)$ für $\varepsilon > 0$, und ist $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und $n \to \infty$, dann gilt $T(n) = \Theta(f(n))$.