



Algorithmentheorie

03 - Randomisierung

Prof. Dr. S. Albers
Prof. Dr. Th. Ottmann

- Klassen von randomisierten Algorithmen
- Randomisierter Quicksort
- Randomisierter Primzahltest
- Kryptographie

1. Klassen von randomisierten Algorithmen

- **Las Vegas Algorithmen**

immer korrekt; erwartete Laufzeit

Beispiel: randomisierter Quicksort

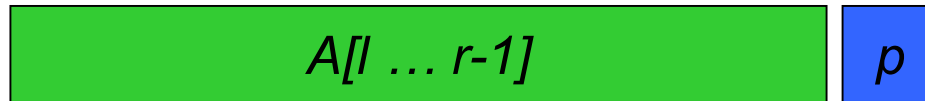
- **Monte Carlo Algorithmen** (**m**ost **c**orrectly):

wahrscheinlich korrekt; garantierte Laufzeit

Beispiel: randomisierter Primzahltest

2. Quicksort

Unsortierter Bereich $A[l,r]$ in einem Array A



Quicksort

Quicksort

Quicksort

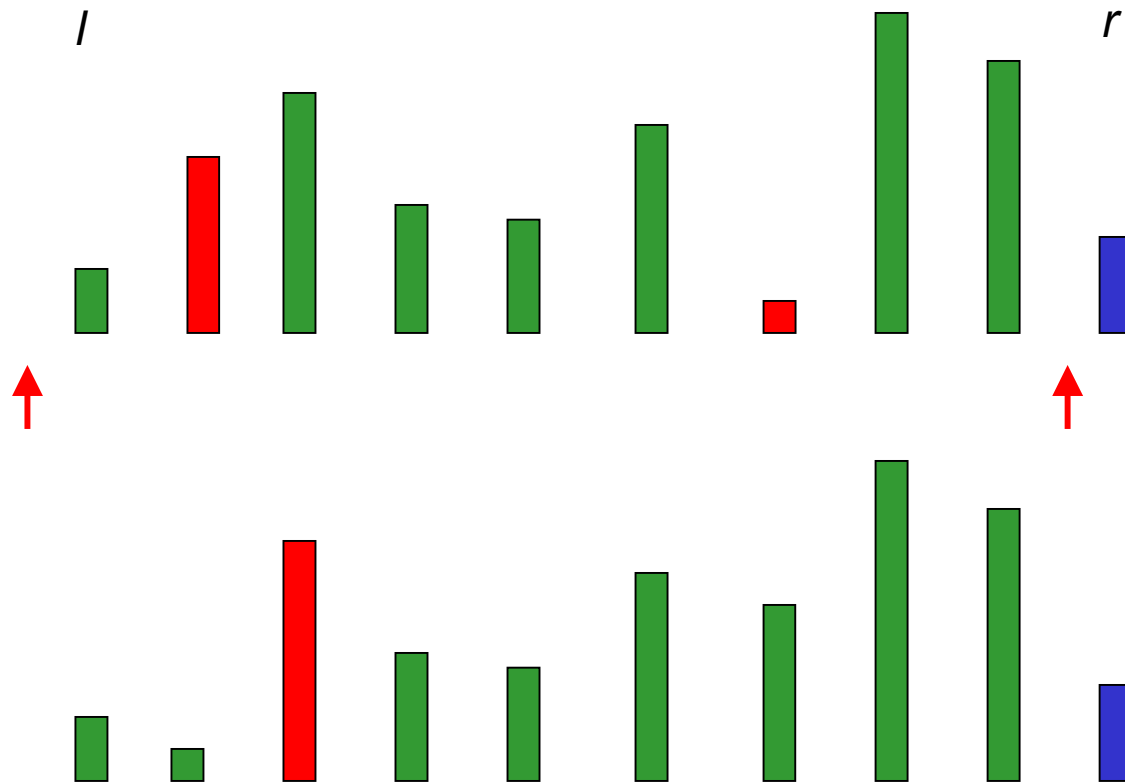
Algorithmus: *Quicksort*

Input: unsortierter Bereich $[l, r]$ in Array A

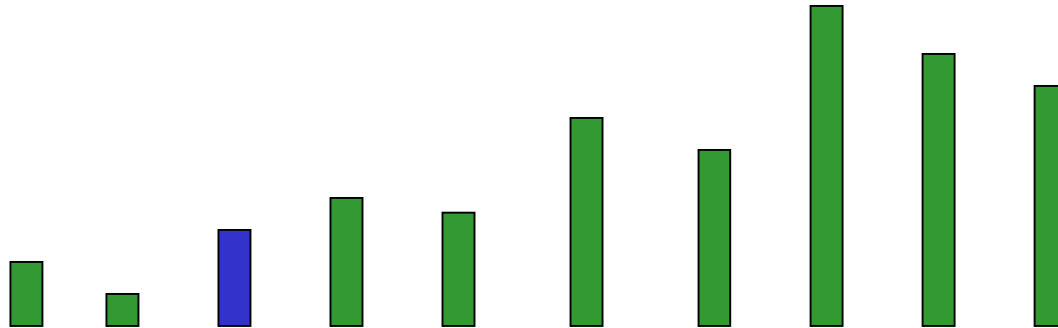
Output: sortierter Bereich $[l, r]$ in Array A

```
1  if  $r > l$ 
2    then wähle Pivotelement  $p = A[r]$ 
3     $m = \text{divide}(A, l, r)$ 
      /* Teile  $A$  bzgl.  $p$  auf:
       $A[l], \dots, A[m - 1] \leq p \leq A[m + 1], \dots, A[r]$ 
      */
4    Quicksort( $A, l, m - 1$ )
   Quicksort( $A, m + 1, r$ )
```

Der Aufteilungsschritt



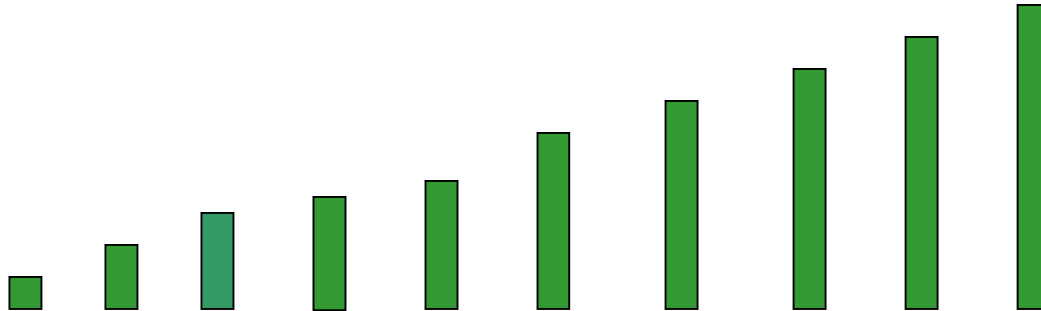
Der Aufteilungsschritt



$\text{divide}(A, l, r)$:

- liefert den Index des Pivotelements in A
- ausführbar in Zeit $O(r - l)$

Worst-Case-Eingabe



n Elemente:

Laufzeit: $(n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2$

3. Randomisierter Quicksort

Algorithmus: Quicksort

Input: unsortierter Bereich $[l, r]$ in Array A

Output: sortierter Bereich $[l, r]$ in Array A

```
1  if  $r > l$ 
2      then wähle zufälliges Pivotelement  $p = A[i]$ 
           im Bereich  $[l, r]$ 
3      tausche  $A[i]$  mit  $A[r]$ 
4       $m = \text{divide}(A, l, r)$ 
           /* Teile  $A$  bzgl.  $p$  auf:
            $A[l], \dots, A[m - 1] \leq p \leq A[m + 1], \dots, A[r]$ 
           */
5      Quicksort( $A, l, m - 1$ )
6      Quicksort( $A, m + 1, r$ )
```

Analyse 1

n Elemente; sei S_i das i -t kleinste Element

Mit WSK $1/n$ ist S_1 Pivotelement:

Teilprobleme der Größen 0 und $n-1$

-
-
-

Mit WSK $1/n$ ist S_k Pivotelement:

Teilprobleme der Größen $k-1$ und $n-k$

-
-
-

Mit WSK $1/n$ ist S_n Pivotelement:

Teilprobleme der Größen $n-1$ und 0

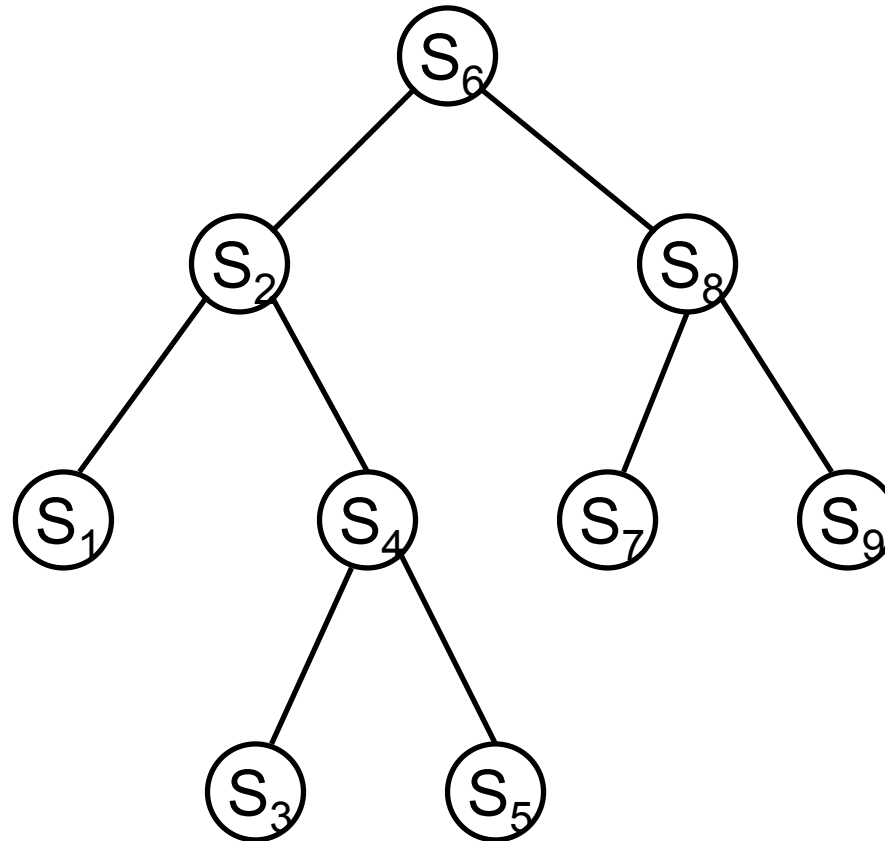
Erwartungswert für die Laufzeit:

$$T(n) = \frac{1}{n} \sum_{k=1}^n (T(k-1) + T(n-k)) + \Theta(n)$$

$$= \frac{2}{n} \sum_{k=1}^n T(k-1) + \Theta(n)$$

$$= O(n \log n)$$

Analyse 2: Darstellung von QS als Baum



$$\pi = S_6 S_2 S_8 S_1 S_4 S_7 S_9 S_3 S_5$$

Erwartete Anzahl der Vergleiche:

$$X_{ij} = \begin{cases} 1 & \text{falls } S_i \text{ mit } S_j \text{ verglichen wird} \\ 0 & \text{sonst} \end{cases}$$

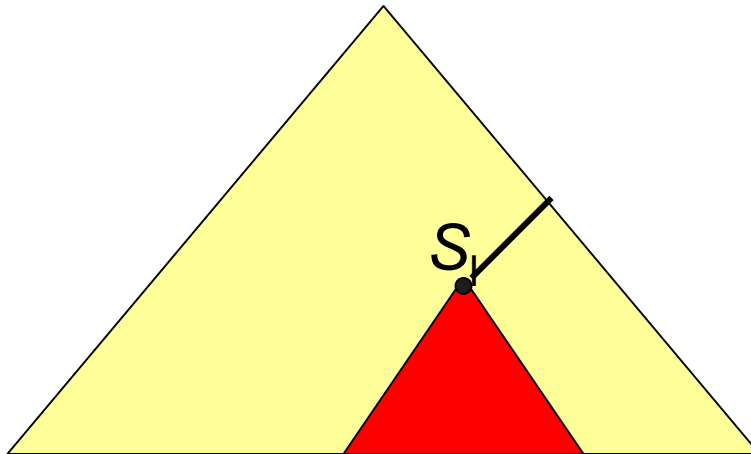
$$E \left[\sum_{i=1}^n \sum_{j>i} X_{ij} \right] = \sum_{i=1}^n \sum_{j>i} E[X_{ij}]$$

p_{ij} = WSK S_i wird mit S_j verglichen

$$E[X_{ij}] = 1 \times p_{ij} + 0 \times (1 - p_{ij}) = p_{ij}$$

Berechnung von p_{ij}

- S_i wird mit S_j verglichen gdw S_i oder S_j in π vor allen anderen S_l , $i < l < j$, als Pivotelement gewählt wird.
 $\{S_i \dots S_l \dots S_j\}$
- Jedes der Elemente S_i, \dots, S_j wird mit gleicher WSK als erstes als Pivotelement gewählt.



$$p_{ij} = 2/(j-i+1)$$

$\{\dots S_i \dots S_l \dots S_j \dots\}$

Erwartete Anzahl der Vergleiche:

$$\begin{aligned}\sum_{i=1}^n \sum_{j>i} p_{ij} &= \sum_{i=1}^n \sum_{j>i} \frac{2}{j-i+1} \\ &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\ &\leq 2 \sum_{i=1}^n \sum_{k=1}^n \frac{1}{k} \\ &= 2n \sum_{k=1}^n \frac{1}{k}\end{aligned}$$

$$H_n = \sum_{k=1}^n 1/k \approx \ln n$$

4. Primzahltest

Definition:

Zahl $p \geq 2$ ist genau dann **prim**, wenn aus $a \mid p$ folgt $a = 1$ oder $a = p$.

Algorithmus: Deterministischer Primzahltest (naiv)

Input: eine ganze Zahl $n \geq 2$

Output: Antwort auf die Frage: Ist n prim?

```
if  $n = 2$  then return true
if  $n$  gerade then return false
for  $i = 1$  to  $\sqrt{n}/2$  do
    if  $2i + 1$  teilt  $n$ 
        then return false
return true
```

Laufzeit: $\Theta(\sqrt{n})$

Primzahltest

Ziel:

Randomisiertes Verfahren

- in polynomieller Zeit ausführbar
- falls Ausgabe “nicht prim”, dann ist n nicht prim
- falls Ausgabe “prim”, dann ist n nicht prim höchstens mit Wahrscheinlichkeit $p > 0$

k Iterationen: n ist nicht prim mit Wahrscheinlichkeit p^k

Primzahltest

Beobachtung:

Jede ungerade Primzahl p teilt $2^{p-1} - 1$.

Beispiele: $p = 17, 2^{16} - 1 = 65535 = 17 * 3855$
 $p = 23, 2^{22} - 1 = 4194303 = 23 * 182361$

Einfacher Primzahltest:

- 1 Berechne $z = 2^{n-1} \bmod n$
- 2 **if** $z = 1$
- 3 **then** n ist möglicherweise prim
- 4 **else** n ist definitiv nicht prim

Vorteil: Benötigt nur polynomielle Zeit

Einfacher Primzahltest

Definition:

n ist eine **Pseudoprimzahl** zur Basis 2, wenn n nicht prim und
$$2^{n-1} \bmod n = 1.$$

Beispiel: $n = 11 * 31 = 341$

$$2^{340} \bmod 341 = 1$$

Randomisierter Primzahltest

Satz: (kleiner Fermat)

Ist p prim und $0 < a < p$, dann ist

$$a^{p-1} \bmod p = 1.$$

Definition:

n ist eine **Pseudoprimzahl** zur Basis a , wenn n nicht prim und

$$a^{n-1} \bmod n = 1.$$

Beispiel: $n = 341$, $a = 3$

$$3^{340} \bmod 341 = 56 \neq 1$$

Randomisierter Primzahltest

Algorithmus: Randomisierter Primzahltest 1

- 1 Wähle a im Bereich $[2, n-1]$ zufällig
- 2 Berechne $a^{n-1} \bmod n$
- 3 **if** $a^{n-1} \bmod n = 1$
- 4 **then** n ist möglicherweise prim
- 5 **else** n ist definitiv nicht prim

Prob(n ist nicht prim, aber $a^{n-1} \bmod n = 1$) ?

Carmichael- Zahlen

Problem: Carmichael-Zahlen

Definition: Eine Zahl n heißt eine **Carmichael-Zahl**, falls

$$a^{n-1} \bmod n = 1$$

für alle a mit $\text{ggT}(a, n) = 1$.

Beispiel:

Kleinste Carmichael-Zahl: $561 = 3 * 11 * 17$

Randomisierter Primzahltest 2

Satz:

Ist p prim und $0 < a < p$, dann hat die Gleichung

$$a^2 \bmod p = 1$$

genau die Lösungen $a = 1$ und $a = p - 1$.

Definition:

a heißt **nichttriviale Quadratwurzel** mod n , falls

$$a^2 \bmod n = 1 \text{ und } a \neq 1, n - 1.$$

Beispiel: $n = 35$

$$6^2 \bmod 35 = 1$$

Schnelle Exponentiation

Idee:

Teste während der Berechnung von a^{n-1} , $0 < a < n$ zufällig, ob es eine nichtriviale Quadratwurzel mod n gibt.

Verfahren zur Berechnung von a^n :

Fall 1: [n ist gerade]

$$a^n = a^{n/2} * a^{n/2}$$

Fall 2: [n ist ungerade]

$$a^n = a^{(n-1)/2} * a^{(n-1)/2} * a$$

Schnelle Exponentiation

Beispiel:

$$a^{62} = (a^{31})^2$$

$$a^{31} = (a^{15})^2 * a$$

$$a^{15} = (a^7)^2 * a$$

$$a^7 = (a^3)^2 * a$$

$$a^3 = (a)^2 * a$$

Laufzeit: $O(\log^2 a^n \log n)$

Schnelle Exponentiation

```
boolean isProbablyPrime;
```

```
power(int a, int p, int n){
```

```
    /* berechnet  $a^p \bmod n$  und prüft, ob bei der Berechnung  
       ein  $x$  auftritt mit  $x^2 \bmod n = 1$  und  $x \neq 1, n-1$  */
```

```
    if (p == 0) return 1;
```

```
    x = power(a, p/2, n)
```

```
    result = (x * x) % n;
```

Schnelle Exponentiation

```
/* prüfe ob  $x^2 \bmod n = 1$  und  $x \neq 1, n-1$  */
```

```
if (result == 1 && x != 1 && x != n - 1 )  
    isProbablyPrime = false;
```

```
if (p % 2 == 1)  
    result = (a * result) % n;
```

```
return result;  
}
```

Laufzeit: $O(\log^2 n \log p)$

Randomisierter Primzahltest 2

```
primeTest(int n) {  
    /* führt den randomisierten Primzahltest für ein zufälliges a aus */  
  
    a = random(2, n-1);  
  
    isProbablyPrime = true;  
  
    result = power(a, n-1, n);  
  
    if (result != 1 || !isProbablyPrime)  
        return false;  
    else return true;  
  
}
```

Randomisierter Primzahltest 2

Satz:

Ist n nicht prim, so gibt es höchstens

$$\frac{n-9}{4}$$

Zahlen $0 < a < n$, für die Algorithmus **primeTest** versagt.