Entwurf und Analyse von Algorithmen

Inhalt:

1. Einführung
   - Asymptotik
   - Ein einfaches Beispiel
2. Zum Aufwärmen: Sortieren
   - *InsertionSort*
   - untere Schranke für vergleichsbasiertes Sortieren
   - *QuickSort*
3. *Divide-and-Conquer*-Verfahren
   - *MergeSort*
   - Multiplikation von Polynomen und Matrizen
   - Analyse von Divide-and-Conquer Algorithmen
4. *Greedy* Algorithmen
   - HUFFMANN-Codes, Datenkompression
   - Matroide
   - KRUSKAL's Algorithmus für Minimale Spannbäume
5. *String Matching*
   - Automatentheoretische Methoden
   - Algorithmus von KNUTH-MORRIS-PRATT

6. Probabilistische Algorithmen
   - Einführung
   - Las Vegas vs. Monte Carlo
   - Die Markov- und Tschebyscheff-Ungleichungen
     - Ein Selektionsalgorithmus
     - Wahrscheinlichkeitsverstärkung
   - *Fingerprinting*
     - Verifikation von Polynomgleichungen
     - Matchings in Graphen
     - Pattern Matching
   - Die „Probabilistische Methode“
     - MAX SAT
     - Expandergraphen
   - Markov-Ketten und Irrfahrten
     - 2 SAT
     - Irrfahrten auf Graphen
     - Erreichbarkeit
     - Wahrscheinlichkeitsverstärkung

Literatur:

1. T. H. Cormen, C. E Leiserson and R. L. Rivest: *Introduction to Algorithms*, MIT Press (1990)
2. R. Motwani and P. Raghavan: *Randomized Algorithms*, Cambridge University Press (1995)
3. D. E. Knuth: *The Art of Computer Programming*, Addison-Wesley, Vol. 1-2 (^{3}1997), Vol. 3 (^{2}1998)
4. U. Schöning: *Algorithmen - kurz gefasst*, Spektrum Akademischer Verlag (1997)

Asymptotische Notation

$$O(f(n)) := \{\, g(n)\,;\ \exists c, n_0 > 0\ \forall n \geq n_0\ 0 \leq g(n) \leq c \cdot f(n) \,\}$$

$$\Omega(f(n)) := \{\, g(n)\,;\ \exists c, n_0 > 0\ \forall n \geq n_0\ 0 \leq c \cdot f(n) \leq g(n) \,\}$$

Es gilt: $\quad g(n) \in O(f(n)) \Leftrightarrow f(n) \in \Omega(g(n))$.

$$\Theta(f(n)) := O(f(n)) \cap \Omega(f(n))$$

$$o(f(n)) := \{\, g(n)\,;\ \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0\ \ 0 \leq g(n) \leq c \cdot f(n) \,\}$$

$$\omega(f(n)) := \{\, g(n)\,;\ \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0\ \ 0 \leq c \cdot f(n) \leq g(n) \,\}$$

Für $f(n), g(n) > 0$:

$$f(n) \in o(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Algorithmus `maxfind`: #

Code	#
procedure maxfind($A[1..n]$: integer) : integer	
$max := A[1]$	1
for $i := 2$ **to** n **do**	$n-1$
if $A[i] \geq max$ **then**	$n-1$
$max := A[i]$	??
end	
return(max)	
end	

$\mathrm{lrm}(A) := \#\{\, 1 \leq i \leq n\,;\ \forall j < i\ A[j] < A[i]\,\}$

Also: ?? $= \mathrm{lrm}(A) - 1$

Zeitkomplexität des Algorithmus:

$$c_0 + c_1 \cdot n + c_2 \cdot \mathrm{lrm}(A)$$

Permutationen in S_4

σ	$\mathrm{lrm}(\sigma)$	σ	$\mathrm{lrm}(\sigma)$
$[1,2,3,4]$	4	$[3,1,2,4]$	2
$[1,2,4,3]$	3	$[3,1,4,2]$	2
$[1,3,2,4]$	3	$[3,2,1,4]$	2
$[1,3,4,2]$	3	$[3,2,4,1]$	2
$[1,4,2,3]$	2	$[3,4,1,2]$	2
$[1,4,3,2]$	2	$[3,4,2,1]$	2
$[2,1,3,4]$	3	$[4,1,2,3]$	1
$[2,1,4,3]$	2	$[4,1,3,2]$	1
$[2,3,1,4]$	3	$[4,2,1,3]$	1
$[2,3,4,1]$	3	$[4,2,3,1]$	1
$[2,4,1,3]$	2	$[4,3,1,2]$	1
$[2,4,3,1]$	2	$[4,3,2,1]$	1

Algorithmus `InsertionSort`:	#
`procedure InsertionSort(`$A[1..n]$ `: integer)`	
`for` $i := 2$ `to` n `do`	$n-1$
$k := A[i]$	$n-1$
$j := i-1$	$n-1$
`while` $j > 0$ `and` $A[j] > k$ `do`	$I_j(A)$
$A[j+1] := A[j]$	$I_j(A) - 1$
$j := j-1$	$I_j(A) - 1$
`end`	
$A[j+1] := k$	$n-1$
`end`	
`end`	

$$I_j(A) := \#\{\, i \,;\, 1 \le i < j \text{ und } A[i] > A[j] \,\}$$

Zeitkomplexität:

$$c_0 + c_1 \cdot n + c_2 \sum_{j=2}^{n} I_j(A)$$

Algorithmus `QuickSort`:

```
procedure QuickSort(A[1..n], p, r : integer)
   if p < r then
      q := Partition(A, p, r)
      QuickSort(A, p, q)
      QuickSort(A, q + 1, r)

procedure Partition(A[1..n], p, r : integer):integer
   x := A[p]
   i := p − 1
   j := r + 1
   while TRUE do
      repeat j := j − 1 until A[j] ≤ x
      repeat i := i + 1 until A[i] ≥ x
      if i < j
         then exchange(A[i], A[j])
         else return j
```

Algorithmus `RandomQuickSort`:

```
procedure RandomQuickSort(A[1..n], p, r : integer)
   if p < r then
      q := RandomPartition(A, p, r)
      RandomQuickSort(A, p, q)
      RandomQuickSort(A, q + 1, r)

procedure RandomPartition(A[1..n], p, r : integer):integer
   i := random(p, r)
   exchange(A[p], A[i])
   return Partition(A, p, r)
```

Algorithmus `MergeSort`:

```
procedure MergeSort(A[1..n], p, r : integer)
    if p = r then return;
    q := ⌊(p+r)/2⌋
    MergeSort(A, p, q)
    MergeSort(A, q + 1, r)
    Merge(A, p, q, r)

procedure Merge(A[1..n], p, q, r : integer)
    i := p
    j := q + 1
    k := 1
    while i ≤ q and j ≤ r do
       if A[i] < A[j]
          then  B[k] := A[i]
                i := i + 1
          else  B[k] := A[j]
                j := j + 1
       k := k + 1
    if i ≤ p
       then B[k .. (r − p + 1)] := A[i .. q]
       else B[k .. (r − p + 1)] := A[j .. r]
    A[p .. r] := B[1 .. (r − p + 1)]
```

Theorem: Seien $a \geq 1$ und $b > 1$ und $f : \mathbb{N} \to \mathbb{R}$ gegeben. Die Funktion T erfülle die Rekursionsgleichung

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

Dann gilt:

1. Falls $f(n) = O(n^{\log_b a - \epsilon})$, dann ist $T(n) = \Theta(n^{\log_b a})$.

2. Falls $f(n) = \Theta(n^{\log_b a})$, dann ist $T(n) = \Theta(n^{\log_b a} \log n)$.

3. Falls $f(n) = \Omega(n^{\log_b a + \epsilon})$, und $af(\frac{n}{b}) \leq cf(n)$ für ein $c < 1$ und fast alle n, dann ist $T(n) = \Theta(f(n))$.

Multiplikation von Polynomen

Linear:

$$(ax+b)(cx+d) = acx^2 + (ad+bc)x + bd$$

Kosten: 4 Multiplikationen, 1 Addition.

besseres Verfahren:

$$\begin{aligned}
&\mathrm{tmp} := (a+b)(c+d)\\
&A := ac\\
&C := bd\\
&B := \mathrm{tmp} - A - C\\
&\texttt{return}\ Ax^2 + Bx + C
\end{aligned}$$

Kosten: 3 Multiplikationen, 4 Additionen

Multiplikation $p \times q$ mit p, q vom Grad $< 2n$ (Karatsuba):

Schreibe

$$p = p_1 x^n + p_2$$
$$q = q_1 x^n + q_2$$

und berechne:

$$\begin{aligned}
&\mathtt{tmp} := (p_1 + p_2) \times (q_1 + q_2) && 2n \text{ Additionen} \\
&r_1 := p_1 \times q_1 && \\
&r_3 := p_2 \times q_2 && \\
&r_2 := \mathtt{tmp} - r_1 - r_3 && 4n \text{ Additionen} \\
&\mathbf{return}\ r_1 x^{2n} + r_2 x^n + r_3 && 2n \text{ Additionen}
\end{aligned}$$

Kosten $t(n)$ zur Multiplikation zweier Polynome vom Grad $< n$:

$$t(1) := 7$$
$$t(n) := 3t(\frac{n}{2}) + 8n \qquad \Rightarrow \qquad t(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1,585\ldots})$$

Multiplikation von $n \times n$-Matrizen:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$r := ae + bf$$

$$s := ag + bh$$

$$t := ce + df$$

$$u := cg + dh$$

8 Multiplikationen von $\frac{n}{2} \times \frac{n}{2}$-Matrizen:

$$t(n) = 8t(\frac{n}{2}) + \Theta(n^2)$$

Lösung : $t(n) := \Theta(n^3)$

Matrixmultiplikation nach Strassen:

$$\begin{aligned}
P_1 &:= a \cdot (g - h) && \underline{ag} - ah \\
P_2 &:= (a + b) \cdot h && ah + \underline{bh} \\
s &:= P_1 + P_2 && \\
P_3 &:= (c + d) \cdot e && \underline{ce} + de \\
P_4 &:= d \cdot (f - e) && \underline{df} - de \\
t &:= P_3 + P_4 && \\
P_5 &:= (a + d) \cdot (e + h) && \underline{ae} + ah + de + \underline{dh}
\end{aligned}$$

$$P_5 + P_4 - P_2 = \underline{ae} + dh + df - bh$$

$$\begin{aligned}
P_6 &:= (b - d) \cdot (f + h) && \underline{bf} + bh - df - dh \\
r &:= P_5 + P_4 - P_2 + P_6 &&
\end{aligned}$$

$$P_5 + P_1 - P_3 = ae + ag - ce + \underline{dh}$$

$$\begin{aligned}
P_7 &:= (a - c) \cdot (e + g) && ae + ag - ce - \underline{cg} \\
u &:= P_5 + P_1 - P_3 - P_7 &&
\end{aligned}$$

7 Multiplikationen von $\frac{n}{2} \times \frac{n}{2}$-Matrizen:

$$t(n) = 7t(\frac{n}{2}) + \Theta(n^2)$$

Lösung : $t(n) := \Theta(n^{\log_2 7}) = \Theta(n^{2,807\ldots})$.

Bester bekannter Algorithmus: $\Theta(n^{2,376})$.

Multiplikation $C = A \cdot B$:

$$
\begin{tikzcd}
\begin{array}{rl} A: & \langle a_0, \ldots, a_{n-1} \rangle \\ B: & \langle b_0, \ldots, b_{n-1} \rangle \end{array} \arrow[r, "\text{Multiplikation}", "\text{(Faltung)}"'] \arrow[d, "\text{Auswertung}"'] & C : \langle c_0, \ldots, c_{2n-1} \rangle \\
\begin{array}{l} A : [(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})] \\ B : [(x_0, z_0), \ldots, (x_{2n-1}, z_{2n-1})] \end{array} \arrow[r, "u_i = y_i z_i"'] & C : [(x_0, u_0), \ldots, (x_{2n-1}, u_{2n-1})] \arrow[u, "\text{Interpolation}"']
\end{tikzcd}
$$

Interpolation: $\quad y_k = A(x_k) \quad$ für $k = 0, \ldots, n-1$:

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}}_{=: V(x_0, \ldots, x_{n-1})} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$V(\vec{x})$ heißt Vandermonde-Matrix.

Falls $x_k \neq x_j$ für alle $k \neq j$, ist $V(\vec{x})$ invertierbar, denn:

$$\det V(x_0, \ldots, x_{n-1}) = \prod_{j<k<n} (x_k - x_j)$$

Also: Koeffizientenvektor $\vec{a} = V(\vec{x})^{-1}\vec{y}$.

Komplexität: $\Theta(n^3)$

LAGRANGE-Interpolation:

$$A(x) = \sum_{k=0}^{n-1} y_k \cdot \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Komplexität: $\Theta(n^2)$

Auswertung durch HORNER-Schema:

$y := a_{n-1}$
$\texttt{for } k := n-2 \texttt{ to } 0 \texttt{ by } -1 \texttt{ do}$
$\quad y := y \cdot x + a_k$

Komplexität (für n Werte): $\Theta(n^2)$

Für ein Polynom A vom Grad $< n$

$$A(x) := \sum_{j=0}^{n-1} a_j x^j$$

gegeben als Koeffizientenvektor $\langle a_0, a_1, \ldots, a_{n-1} \rangle$, definiere:

$$y_k := A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

Die Abbildung

$$\langle a_0, a_1, \ldots, a_{n-1} \rangle \mapsto (y_0, y_1, \ldots, y_{n-1})$$

heißt *Diskrete Fourier-Transformation*.

Schreibweise: $\vec{y} = \mathsf{DFT}_n(\vec{a})$.

Definiere

$$A_0(x) := a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{\frac{n}{2}-1}$$
$$A_1(x) := a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{\frac{n}{2}-1}$$

so dass

$$(*) \qquad A(x) = A_0(x^2) + x A_1(x^2)$$

Auswertung von $A(x)$ bei $\omega_n^0, \ldots, \omega_n^{n-1}$ reduziert zu:

- Auswertung von $A_0(x)$ und $A_1(x)$ bei $(\omega_n^0)^2, \ldots, (\omega_n^{n-1})^2$
- Kombination entsprechend $(*)$.

Die Werte $(\omega_n^0)^2, \ldots, (\omega_n^{n-1})^2$ sind $\omega_{\frac{n}{2}}^0, \ldots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1}$.

Schnelle Fourier-Transformation (FFT):

```
procedure FFT(A[0..n−1] : integer)
    if n = 1 then return A
    x := 1
    A_0 := [A[0], A[2], ... , A[n−2]]
    A_1 := [A[1], A[3], ... , A[n−1]]
    Y_0 := FFT(A_0)
    Y_1 := FFT(A_1)
    for k := 0 to n/2 − 1 do
        Y[k] := Y_0[k] + x · Y_1[k]
        Y[k + n/2] := Y_0[k] − x · Y_1[k]
        x := x · ω_n
    return Y
```

Komplexität: $T(n) = 2T(\frac{n}{2}) + \Theta(n) \qquad \Rightarrow \qquad T(n) = \Theta(n \log n)$.

Umkehrabbildung DFT^{-1}:

$$\vec{y} = \mathsf{DFT}_n(\vec{a}) = V_n \cdot \vec{a} \qquad \Rightarrow \qquad \vec{a} = V_n^{-1} \cdot \vec{y}$$

mit $V_n = V(\omega_n^0, \ldots, \omega_n^{n-1})$, also $\left(V_n\right)_{k,j} = \omega_n^{kj}$ für $k, j = 0, \ldots, n-1$.

Satz: Für $k, j = 0, \ldots, n-1$ ist $\left(V_n^{-1}\right)_{k,j} = \frac{1}{n}\omega_n^{-kj}$.

Berechnung von DFT^{-1} durch Modifikation von `FFT`:

- Ersetze in `FFT` ω_n durch $\omega_n^{-1} = \omega_n^{n-1}$.
- Ergänze am Schluss Division jedes Wertes durch n.

Also: Komplexität von DFT^{-1} ebenfalls $\Theta(n \log n)$.

Multiplikation von Polynomen vom Grad $< n$ in $\Theta(n \log n)$ Zeit:

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{ll} \vec{a}: & \langle a_0, \ldots, a_{n-1}, 0, \ldots, 0 \rangle \\ \vec{b}: & \langle b_0, \ldots, b_{n-1}, 0, \ldots, 0 \rangle \end{array} \arrow[r, "\text{Multiplikation}", "\text{(Faltung)}"'] \arrow[d, "\text{FFT}"', "\Theta(n \log n)"] & \vec{c}: \langle c_0, \ldots, c_{2n-1} \rangle \\
\begin{array}{l} \mathsf{DFT}(\vec{a}): (y_0, \ldots, y_{2n-1}) \\ \mathsf{DFT}(\vec{b}): (z_0, \ldots, z_{2n-1}) \end{array} \arrow[r, "u_i = y_i z_i", "\Theta(n)"'] & \vec{u}: (u_0, \ldots, u_{2n-1}) \arrow[u, "\text{FFT}^{-1}", "\Theta(n \log n)"']
\end{tikzcd}$$

Ein *Optimierungsproblem* ist gegeben durch:

- Eine Menge I von Instanzen.
- Für jede Instanz $x \in I$ eine Menge S_x von potentiellen Lösungen.
- Einer Maßfunktion m die jeder Instanz x und jeder Lösung $y \in S_x$ ein Maß $m(x, y)$ zuordnet.
- Ein Ziel $\text{goal} \in \{\min, \max\}$.

Eine Lösung $y^* \in S_x$ heißt *optimal*, wenn
$m(x, y^*) = m^*(x) := \text{goal}\{\, m(x, y) \,;\, y \in S_x \,\}$.

Beispiel: MAX CLIQUE

Instanz:	Graph $G = (V, E)$
Lösung:	Clique $C \subseteq V$ mit $\{x, y\} \in E$ für alle $x, y \in C$
Maß:	Größe $\lvert C \rvert$

Ein Auswahlproblem:

Instanz: Menge $A = \{(s_1, e_1), \ldots, (s_n, e_n)\}$ mit $s_i \leq e_i$ für alle i

Lösung: $B \subseteq [n]$ mit $\forall i \neq j \in B \ : \ s_i \geq e_j$ oder $s_j \geq e_i$.

Ziel: maximiere $|B| = k$.

Annahme: S nach den e_i sortiert: $e_1 \leq e_2 \leq \cdots \leq e_n$.

Algorithmus G:

$B := \{1\}$
$j := 1$
for $i := 2$ **to** n **do**
  if $s_i \geq e_j$ **then**
    $B := B \cup \{i\}$
    $j := i$
return B

Satz: Der Algorithmus G findet stets eine optimale Lösung für das Auswahlproblem.

Entscheidende Eigenschaften:

- *Greedy Choice* Prinzip

  Eine optimale Lösung kann stets durch lokal optimale Auswahl erreicht werden.

- Optimale Teillösungen

  Optimale Lösung enthält optimale Teillösungen für Teilprobleme.

Das Rucksack-Problem:

Instanz: Menge $A = \{(w_1, g_1), \ldots, (w_n, g_n)\}$ mit $w_i, g_i \in \mathbb{N}$, $G \in \mathbb{N}$

Lösung: $a \in \{0,1\}^n$ mit $\sum_{i=1}^{n} a_i \cdot g_i \; < \; G$

Ziel: maximiere $\sum_{i=1}^{n} a_i \cdot w_i$.

Verallgemeinerung: das *gebrochene* Rucksackproblem:

Statt $a \in \{0,1\}^n$ ist $a \in [0,1]^n$ erlaubt.

Hier funktioniert die *greedy* Strategie.

Optimaler Präfixcode:

Instanz: Alphabet $A = \{a_1, \ldots, a_n\}$ mit Häufigkeiten $h(a_i) \in \mathbb{R}_+$

Lösung: Präfixcode $c : A \to \{0, 1\}^*$

Ziel: minimiere $\sum_{i=1}^{n} h(a_i) \cdot |c(a_i)|$.

Algorithmus Huffman:

```
procedure huffman(Q: queue) : tree
   for i := 1 to n − 1 do
      z := new-tree
      left(z) := x := extract-min(Q)
      right(z) := y := extract-min(Q)
      h(z) := h(x) + h(y)
      insert(Q, z)
   return extract-min(Q)
```

Definition: $(\mathsf{M}, \mathcal{U})$, wobei M eine endliche Menge ist und $\mathcal{U} \subseteq 2^{\mathsf{M}}$, ist ein *Matroid*, falls

1. $\emptyset \in \mathcal{U}$
2. Ist $\mathsf{A} \in \mathcal{U}$ und $\mathrm{B} \subseteq \mathsf{A}$, so auch $\mathrm{B} \in \mathcal{U}$
3. Sind $\mathsf{A}, \mathrm{B} \in \mathcal{U}$ mit $|\mathrm{B}| < |\mathsf{A}|$, dann gibt es $e \in \mathsf{A} \setminus \mathrm{B}$ mit $\mathrm{B} \cup \{e\} \in \mathcal{U}$.

$\mathcal{U}$ heißen die *unabhängigen* Teilmengen von M.

Ein gewichtetes Matroid $(\mathsf{M}, \mathcal{U})$ mit $w : \mathsf{M} \to \mathbb{R}_+$ definiert ein Optimierungsproblem:

Finde $\mathsf{T} \in \mathsf{U}$ mit $w(\mathsf{T}) := \sum_{e \in \mathsf{T}} w(e)$ maximal.

Annahme: $\mathsf{M} = \{e_1, e_2, \ldots, e_n\}$ nach Gewicht geordnet:

$$w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$$

Kanonischer *Greedy* Algorithmus:

```
T := ∅
for i := 1 to n do
   if T ∪ {e_i} ∈ 𝒰 then
      T := T ∪ {e_i}
return T
```

Satz: Ist $(\mathsf{M}, \mathcal{U})$ ein Matroid mit Gewicht $w : \mathsf{M} \to \mathbb{R}_+$, so liefert der kanonische Greedy-Algorithmus eine Teilmenge $\mathsf{T} \in \mathcal{U}$ maximalen Gewichts.

Algorithmus **LazySelect**:

1. Bilde $R \subset S$, indem $m = n^{\frac{3}{4}}$ Elemente aus S zufällig gewählt werden.
2. Sortiere R.
3. Setze $x := kn^{\frac{1}{4}}$, und

$$\ell := \max(\lfloor x - \sqrt{n} \rfloor, 1) \qquad h := \min(\lceil x + \sqrt{n} \rceil, n^{\frac{3}{4}})$$
$$a := R_{(\ell)} \qquad b := R_{(h)}$$

4. Bestimme $r_S(a)$ und $r_S(b)$.
5. In Abhängigkeit von k, bestimme die Menge P:

$$k < n^{\frac{1}{4}} : \qquad P := \{y \in S \,;\, y \leq b\}$$
$$k \in [n^{\frac{1}{4}}, n - n^{\frac{1}{4}}] : \qquad P := \{y \in S \,;\, a \leq y \leq b\}$$
$$k > n - n^{\frac{1}{4}} : \qquad P := \{y \in S \,;\, y \geq a\}$$

6. Falls $S_{(k)} \notin P$ oder $|P| > 4n^{\frac{3}{4}} + 2$, wiederhole Schritte 1. - 4.
7. Sonst sortiere P, und gib $P_{(k - r_S(a) + 1)}$ aus.