

Datenstrukturen und Algorithmen

1. Gegeben sei ein Array $A[1 \dots n]$ mit n ganzen Zahlen und folgender Pseudocode:

```

Algo(A, i, j) -  $\Theta(n)$ 
1: IF i==j RETURN A[i] -  $\Theta(1)$ 
2: IF i==j-1 -  $\Theta(1)$ 
3:   IF A[i]<A[j] RETURN A[j] }  $\Theta(1)$ 
4:   ELSE   RETURN A[i] }
5: ELSE
6:   p=(i+j)/2 -  $\Theta(1)$ 
7:   r1 = Algo(A,i,p) -  $\Theta(\frac{n}{2})$ 
8:   r2 = Algo(A,p+1,j) -  $\Theta(\frac{n}{2})$ 
9:   IF r1<r2 RETURN r2 }  $\Theta(1)$ 
10:  ELSE   RETURN r1 }
    
```

Zeile 0:

Programmdefinition

Zeile 01:

Vergleich aufrufender Werte; Ausgabe eines Arrayelementes wenn i und j auf das selbe Element zeigen

Zeile 02 - 04:

Vergleich aufrufender Werte; sollte i auf genau das Element vor j zeigen, wird das groessere Element ausgegeben;

Zeile 05 - 08:

neue Aufrufende Werte definieren (p); --> is p ein int, d.h. nachkommastellen werden abgeschnitten;

Rekursiver Aufruf der Funktion;

Zeile 09 - 10:

Ausgabe des groesseren Wertes;

$A = \{ \overset{1}{1}, \overset{2}{2}, \overset{3}{3}, \overset{4}{4}, \overset{5}{5} \}$

$i=1$ $i=1$ $i=1$

$j=1$ $j=2$ $j=3$

$\rightarrow 1$ $\rightarrow 2$ $\rightarrow p=2$ $r_1 = (1,2) \rightarrow 2$ $r_2 = (3,3) \rightarrow 3$ } $\rightarrow 3$

$i=1$

$j=4$

$\rightarrow p=2$ $r_1 = (1,2) \rightarrow 2$

$r_2 = (3,4) \rightarrow p=3 \rightarrow r_1 = (1,3) \rightarrow 3$ $r_2 = (4,4) \rightarrow 4$ } $\rightarrow 4$

Da in jeder Rekursion im Prinzip nur die hoechsten Werte ausgegeben werden, liefert das Programm in endefekt den letzten Eintrag des Arrays, unter der Annahme $A\{1, \dots n\}$ liefert Algo n

$$T(n) = \Theta(n) + \Theta(n) + \Theta(n) + \Theta(n) + \Theta(n) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\Rightarrow 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \Theta(n)$$

$$T(n) = 2(2(T\left(\frac{n}{4}\right) + \Theta(n)) + \Theta(n))$$

$$= 4T\left(\frac{n}{4}\right) + 2\Theta(n) + \Theta(n)$$

$$\Rightarrow 2^k T\left(\frac{n}{2^k}\right) + (2^k - 1)\Theta(n)$$

$$\rightarrow \frac{n}{2^k} = 1 \Rightarrow n = 2^k$$

$$\ln(n) = k \ln(2)$$

$$k = \frac{\ln(n)}{\ln(2)} = \log_2(n)$$

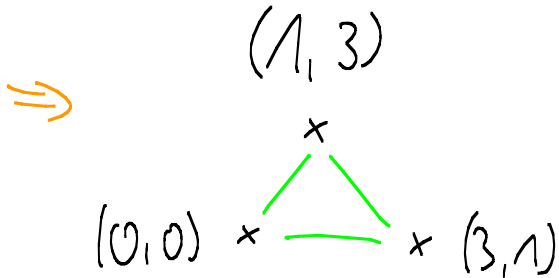
$$2^{\frac{\ln(n)}{\ln(2)}} \cdot T(1) + \left(2^{\frac{\ln(n)}{\ln(2)}} - 1\right)\Theta(n)$$

$$2^{\log_2(n)} \cdot T(1) + (2^{\log_2(n)} - 1)\Theta(n)$$

$$n T(1) + (n - 1)\Theta(n)$$

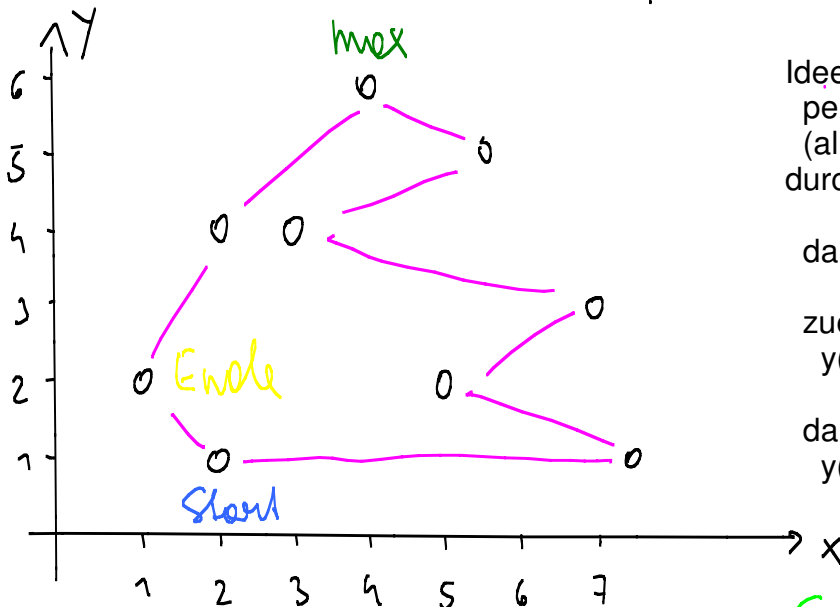
$$\Rightarrow \Theta(n)$$

3. Gegeben seien n Punkte in der Ebene mit ihren (x, y) -Koordinaten. Entwerfen Sie einen möglichst schnellen Algorithmus der alle Punkte durch einen geschlossenen Streckenzug (d.h. Anfangspunkt = Endpunkt) verbindet. Der Streckenzug muß jeden Punkt genau einmal besuchen und darf sich nicht selbst kreuzen. Eine Prozedur die zwei Punkte durch eine Strecke verbindet können Sie als gegeben voraussetzen. Außerdem können Sie annehmen daß es keine kollinearen Punkte gibt (d.h. keine drei verschiedene Punkte liegen auf einer Geraden).



$X(1, \dots, n)$
 $Y(1, \dots, n)$

Bsp.: $(1, 2), (2, 1), (3, 4), (4, 6), (5, 2), (6, 5), (7, 3)$



Idee:

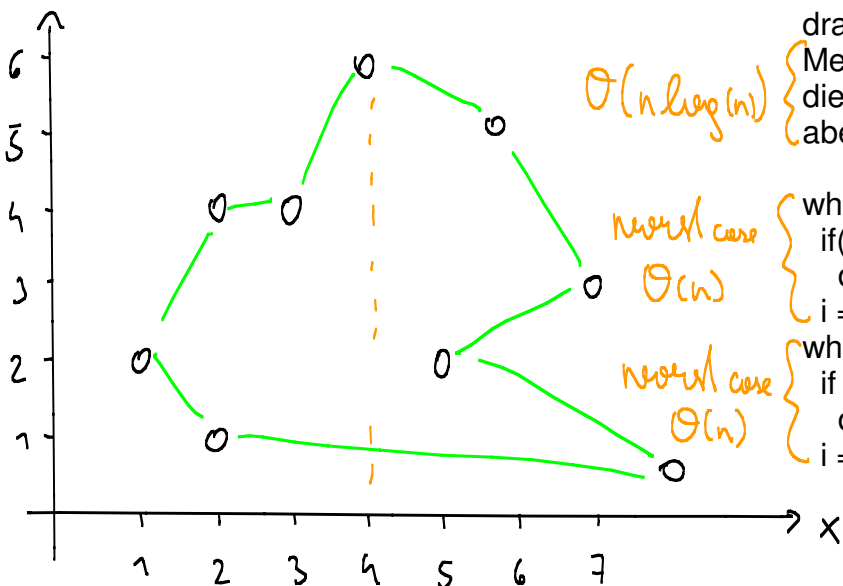
per Mergesort das Y Array sortieren
 (alle Verschiebungen parallel in X
 durchfuehren);

danach $y(1), x(1)$ und $y(n), x(n)$ ermitteln;

zuerst Punkte verbinden die erfullen:
 $y(i) > y(i-1) \ \&\& \ x(i) > x(n); \ i=i+1$ bis $i = n$

danach Punkte verbinden die erfullen:
 $y(i) < y(i+1) \ \&\& \ x(i) < x(n); \ i=i-1$ bis $i = 1$

Code



drawGrid(X,Y,n)

Mergesort(1,j,X,Y); //modifizierter Mergesort, der
 die beiden Arrays nach X sortiert,
 aber Y parallell mit verschiebt

while $i \leq n$ do
 if $(y(i) > y(i-1) \ \&\& \ x(i) > x(n))$
 draw($x(i), y(i)$);
 $i = i + 1$;

while $i \geq 1$ do
 if $(y(i) < y(i-1) \ \&\& \ x(i) < x(n))$
 draw($x(i), y(i)$);
 $i = i - 1$;

$$T(n) = \Theta(n \log(n)) + \Theta(n) + \Theta(n) \rightarrow \bar{T}(n) = \Theta(n \cdot \log(n))$$

4. In einem doppelt indizierten Feld $A[1 \dots n, 1 \dots n]$ sind ganze Zahlen so gespeichert, daß

$$A[i, j] \leq A[i, j+1] \text{ für alle } i = 1, \dots, n, j = 1, \dots, n-1$$

und

$$A[i, j] \leq A[i+1, j] \text{ für alle } i = 1, \dots, n-1, j = 1, \dots, n.$$

Geben Sie einen möglichst effizienten Algorithmus an (Pseudo-Kode), der überprüft ob eine Zahl x in dem Feld A vorkommt.

$$\begin{array}{ll} A(1,1) = 3 & A(1,1) \leq A(1,2) \\ A(1,2) = 4 & \\ \\ A(2,3) = 6 & A(2,3) \leq A(3,3) \\ A(3,3) = 9 & \end{array}$$

Idee:

while ($A(i, n) \leq x$) do $i = i+1$
 while ($A(i, j) \leq x$) do $j = j-1$

⇒

	1	...	n
1	3	4	6
⋮	4	5	6
n	7	8	9

(Note: In the original image, the value 6 in the cell (2,3) is circled in green, with arrows pointing from the 5 in (2,2) and the 6 in (1,3) towards it.)

Wool:

```

finX(A,x)
While (A(i,n) <= x) do
  i = i+1;
While (j <= n && j > 0) do
  if(A(i,j) == x) then
    return i,j;
  else
    if(A(i,j) <= x) then
      j = (n-j)/2 + 1;
    else
      j = (n+j)/2 - 1;
    
```

Complexity annotations:
 } $\mathcal{O}(n)$
 } $\mathcal{O}(n)$

Total complexity:
 } $T(n) = \mathcal{O}(n)$