13. Sortieren

Können Sortieren in

- $\Theta(n^2)$ Selection Sort, Insertion Sort
  Bubble Sort, Quicksort
- $O(n \cdot \log n)$ Heapsort, Mergesort
- $\Theta(n \cdot \log n)$ „in der Regel" Quicksort
  ((Heapsort, Mergesort oft nicht)

Können nur besser als $\Omega(n \cdot \log n)$
sortieren ↯ (Etwa $O(n \log(\log n))$,
oder auch $O(n \cdot \sqrt{\log n})$ oder
$O(n (\log n)^c)$ $c \leq 1$ ↯

Ziel: Klein, solange wir die
Elemente nur untereinander vergleichen
dürfen.

Es geht also hier um einen ganz neuen Aspekt, eine untere Schranke: Jeder Sortieralgorithmus braucht $\Omega(n \cdot \log n)$ (sofern nur Vergleiche).

Problem: Aussage über alle Algorithmen.

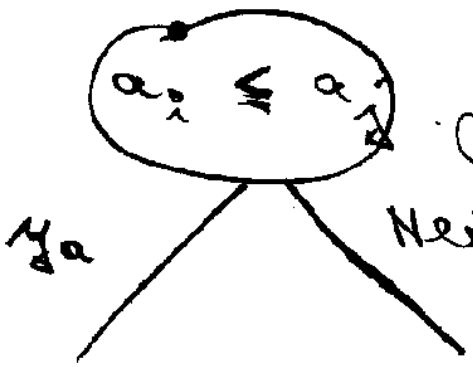
Haben uns also einen Sortieralgorithmus Φ: Wir geben Φ das array $A[1..N]$ of „geordneter Typ" ein. Sei

$$a_i := A[i],$$

wobei $A[1-N]$ Eingabearray ist.

Wir lassen p_1 laufen. Das macht p_2 p_3 p_4 schaut sich vielleicht das N an, rechnet etwas herum. Beachte p_1 kann nicht Teile des Arb A[i:j] = x ausführen. Nur Vergleiche des A.l A[i:j] ≤ A[j:j]. p_1 kann das A blind manipulieren. Hielte p_1 jetzt an, und gäbe eine Manipulation von A aus, wäre es nicht korrekt, da die Ausgabe unabhängig von den $a_1, \ldots, a_n$ ist.

Also irgendwann ein Vergleich



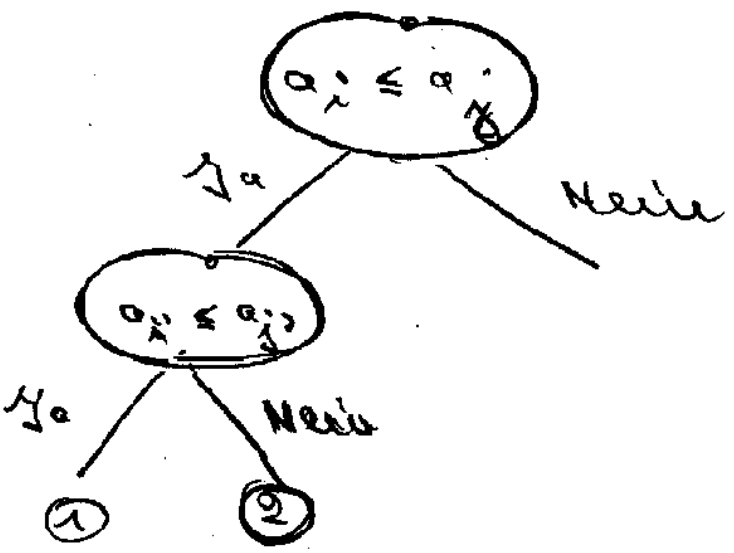
a_i, a_j bezogen auf das Eingabe-Array A!

(erneut ein $a_i \leq a_j$?)

P rechnet weiter, mindestens zwei

Für $N \geq 3$ kann P noch keine Ausgabe tätigen, da der dritte Index noch nicht angeschaut.

Also irgendwann



13.5

Sei jetzt $N = 3$ und

$a_{i'} \leq a_{j'}$ und $i' = j$ und $j' \neq i$,

$j' \neq j$, so haben wir

$$a_i \leq a_j \,,\quad \underset{\neq j}{a_{i'}} \leq a_{j'}.$$

und $a_i \leq a_j \leq a_{j'}$. Also ist

$(a_i, a_j, a_{j'})$ Teilfolge der

Eingabefolge A. Der Algorithmus

kann also bei (1) ausgeben,

wenn $i' = j$ ist. Ist

dagegen $a_{j'} < a_{i'}$ so kann auch

$a_{j'} < a_i$ sein. Der Algorithmus

ist bei (2) keinesfalls fertig,

sofern $i' = j$.

(13,6)

Beweis Blätter muß diese

Entscheidungsbaum haben:

$\geq N!$

bei Eingabe von array A[1..N].

Denn

Blatt $\leftrightarrow$ 1 Umsortierung von [1..N]

$(a_1, \ldots, a_N)$

Möglich sind N! Umsortierungen.

Selection Sort

Suche jeweils das größte

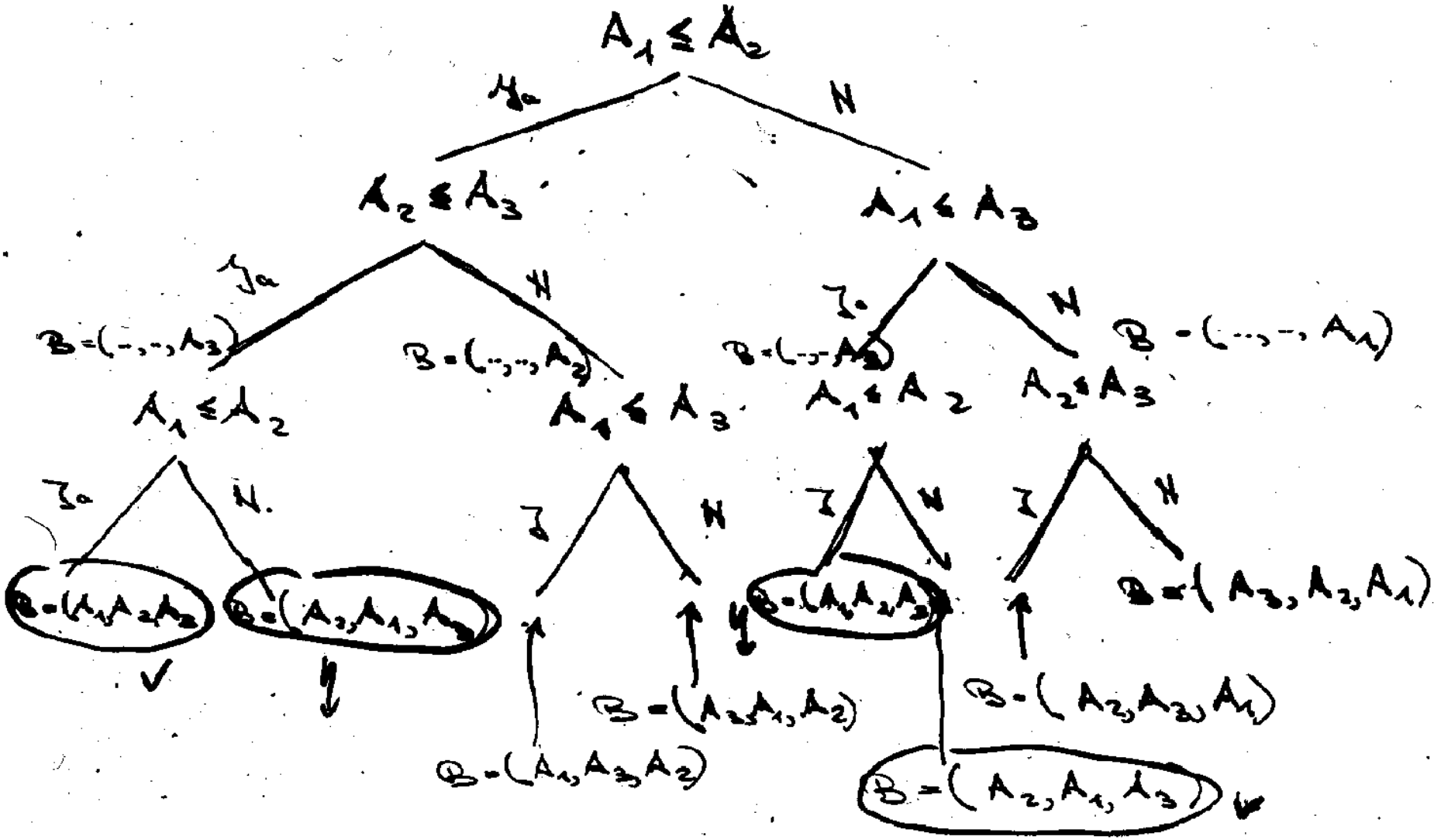
Element von A und lege es

vorläufig in das array B[1-N] ein.

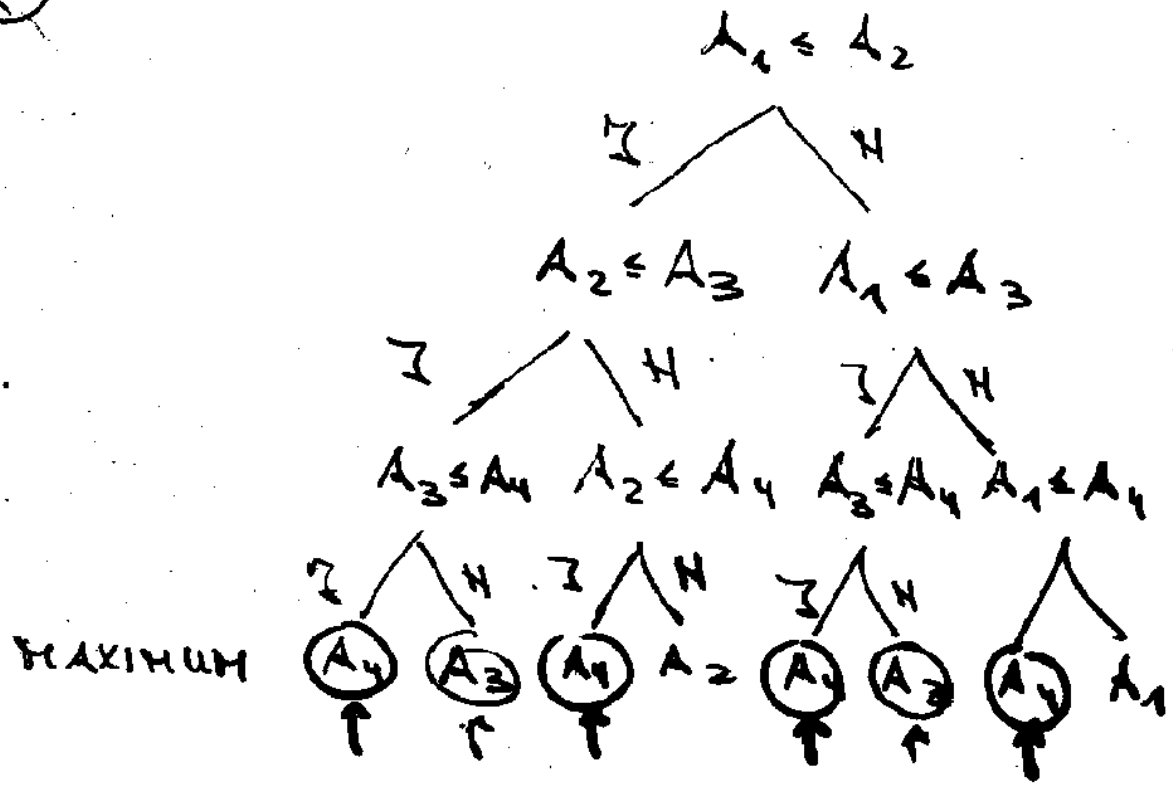
13.1

ENTSCHEIDUNGSBAUM SELECTION SORT

N = 3. EINGABE (A_1, A_2, A_3)



SELECTION SORT N=4



WORST CASE TIME $\in \Omega$ (TIETE)

8 SELECTION SORT'S WITH N = 3.

Beobachtung

(a) Jeder Sortieralgorithmus, der n Elemente nur über Vergleiche sortiert, führt zu einem Entscheidungsbaum T.

(b) Der Entscheidungsbaum hat $\geq n!$ viele Blätter.

(c) Laufzeit $\geq \Omega(\text{Tiefe}(T))$.

Ein Blatt pro Umordnung

Satz

$$\text{Tiefe}(T) \geq \Omega(n \cdot \log n)$$

□

(13.10)

Beweis:

Ein Binärbaum ist in jedem Fall ein bin. Baum, bei dem jeder innere Knoten genau 2 Kinder hat. Sei einmal T ein solcher Baum, dann gilt

$$\#\text{Blätter} \leq 2^{\text{Tiefe}(T)}$$

Tiefe(S) = 0 ~ o

= 1 ~ 

= 2 ~ 

Für unseren E-Baum T

für n Elemente gilt:

$$n! \leq \#\text{Blätter} \leq 2^{\text{Tiefe}(T)}$$

Logarithmieren liefert:

$$\text{Tiefe}(T) \geq \log_2(n!)$$

$$n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow \log(n!) \geq \frac{n}{2}\left(\log \frac{n}{2}\right)$$

$$\geq \log_2 n + \log_2(n-1) + \dots + \log_2 2 + \log_2 1$$

$$\geq \underbrace{\log_2 \frac{n}{2} + \log_2 \frac{n}{2} + \dots + \log_2 \frac{n}{2}}_{\lceil \frac{n}{2} \rceil \text{ - mal}}$$

$$= \lceil \tfrac{n}{2} \rceil \left(\log_2 \tfrac{n}{2}\right) = \lceil \tfrac{n}{2} \rceil \left(\left(\log_2 n\right) - 1\right)$$

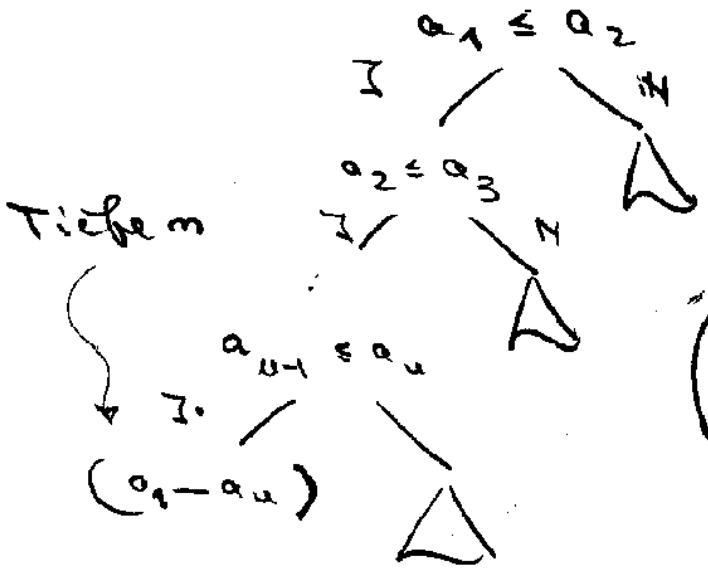
$$= \lceil \tfrac{n}{2} \rceil \cdot \left(\log_2 n\right) - \lceil \tfrac{n}{2} \rceil = \Omega(n \cdot \log n) \quad \square$$

13.12

Noch ein Sortieralgorithmus:

```
1. t := true
2. for i = 1 to n-2 {
3.     if A[i] > A[i+1]; { t := false;
                           break; }
4.  i = i + 1
5. if t = true { return A }
6. Sortiere A mit irgendeinem
                  Algorithmus.
```

E-Baum: Eingabe $(a_1, \dots, a_n)$.



Also: Mind. jedes Blatt in Tiefe $\Omega(n \log n)$

13.13

Erlauben wir mehr als bloße Vergleiche, so können wir schneller als die $O(n \cdot \log n)$ Algorithmen bekommen.

Wir nehmen einmal an, daß wir das array $A[1..n]$ of $\{1,\dots,k\}$ k bekannt und fest und denken, daß $1,\dots,k$ als Indices eines arrays verwendbar sind, sortieren wollen.

Dazu das Zählarray

array $C[1,\dots,k]$ of $\{1,\dots,n\}$

mit $C[i]$ = # Einträge i in A.

13.14

1. for i := 1 to k; C[i] := 0;
2. for j := 1 to m; {
   C[A[j]] := C[A[j]] + 1
   }

Damit ist C[x] in Linearzeit ermittelt. Ist etwa

$A = (1, 2, 3, 2, 6)$, $k = 6$

dann $C[1] = 1$, $C[2] = 2$, $C[3] = 3$
$C[4] = C[5] = 0$, $C[6] = 1$.

Dann berechne man

1. for i := 2 to k; C[i] := C[i] + C[i-1];

Das ergibt: C[i] = # Vorkommen von $j \leq i$ in A

13.15

Also

$C[1] = 1,\ C[2] = 3,\ C[3] = 4,$

$C[4] = C[5] = 4,\ C[6] = 5$

Jetzt A von rechts durchgehen und Element in B an der richtigen Stelle tragen:

$C[A[5]] = 5 \rightsquigarrow B[6] := \overbrace{A[5]}^{=6}$

$C[A[5]] := C[A[5]] - 1$

$C[A[4]] = 3 \rightsquigarrow B[3] := \overbrace{A[4]}^{\cdot 2}$

$C[A[4]] := C[A[4]] - 1$

$C[A[3]] = 4 \rightsquigarrow B[4] := \overbrace{A[3]}^{=3}$

$C[A[3]] := C[A[3]] - 1$

$C[A[2]] = 2 \rightsquigarrow B[2] := \overbrace{A[2]}^{\cdot 2}$

$C[A[2]] := C[A[2]] - 1$

13.16

$C[A[i]] = 1 \rightarrow B[1] = A[i]$

$C[A[i]] := C[A[i]] - 1$

1. for i = n to 1 {

2. $B[C[A[i]]] := A[i]$;

3. $C[A[i]] := C[A[i]] - 1$.

}

keine Vergleiche. Zeit $O(n+k)$.

$A = (1,2,1,2,1,2,1,2)$ $k = 2$

$C[1] = 4$, $C[2] = 4$

$C[1] = 4$, $C[2] = 8$

$B = (\_ \_ \_ \_ \_ \_ \_ ,2)$

$B[8]$

$C[2] = 7$

(13.17)

$B = (\ ,\ ,\ ,1,\ \ \ \ \ \ \ \ 2)$

$B[4]$

$q[1] = 4$

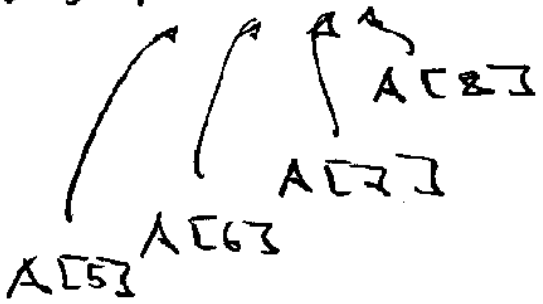
$B = (\ \ \ \ \ \ 1,\ \ \ \ \ 2,2)$

$B = (\ \ \ \ \ \ 1\ 1\ \ \ \ \ 2,2)$

$B = (\ ,1\ 1,1,\ \ 2,2,2)$

$\vdots$

$B = (1,1,1,1,2,2,2,2)$



Annahme gleiche Einträge
in A bleibt bestehen (stabil).

Wegen letzte Schleife

for $j = n$ to 1.