

1. Gegeben seien zwei Arrays $A[1..n]$ und $B[1..n]$ mit je n ganzzahligen Einträgen. Wir nehmen an daß in A und B die gleichen Zahlen stehen, aber in unterschiedlicher Reihenfolge. Wir betrachten Algorithmen bei denen Element in A mit Elementen mit B verglichen werden können, d.h. eine Operation der Form $A[i] < B[j]$ oder $A[i] = B[j]$ ist möglich, nicht aber eine Operation der Form $A[i] < A[j]$ oder $B[i] < B[j]$. Der Algorithmus soll nun ein Mapping von A nach B finden, d.h., das Ergebnis sollte ein Feld $I[1..n]$ sein sodaß $A[i] = B[I[i]]$ für $i = 1, \dots, n$, wobei I alle Zahlen in $\{1, \dots, n\}$ enthalten soll.

Geben Sie einen Algorithmus für dieses Problem an mit Laufzeit $O(n^2)$. Beschreiben Sie Ihren Algorithmus genau und begründen Sie die Laufzeit.

```
for i = 1 to NUM_ELEMENTS do  $\rightarrow O(n)$ 
  loop repeat j=j+1 until !(A[i]==A[j])  $\rightarrow O(n)$ 
    I[i] = j  $\rightarrow O(n)$ 
```

$$T(n) = O(n) \cdot O(n)$$

$$T(n) = O(n^2)$$

- Es muss jedes Element von A betrachtet werden = $O(n)$
- Im worst Case muss jedes Element von B mit dem aktuellen von A verglichen werden = $O(n)$

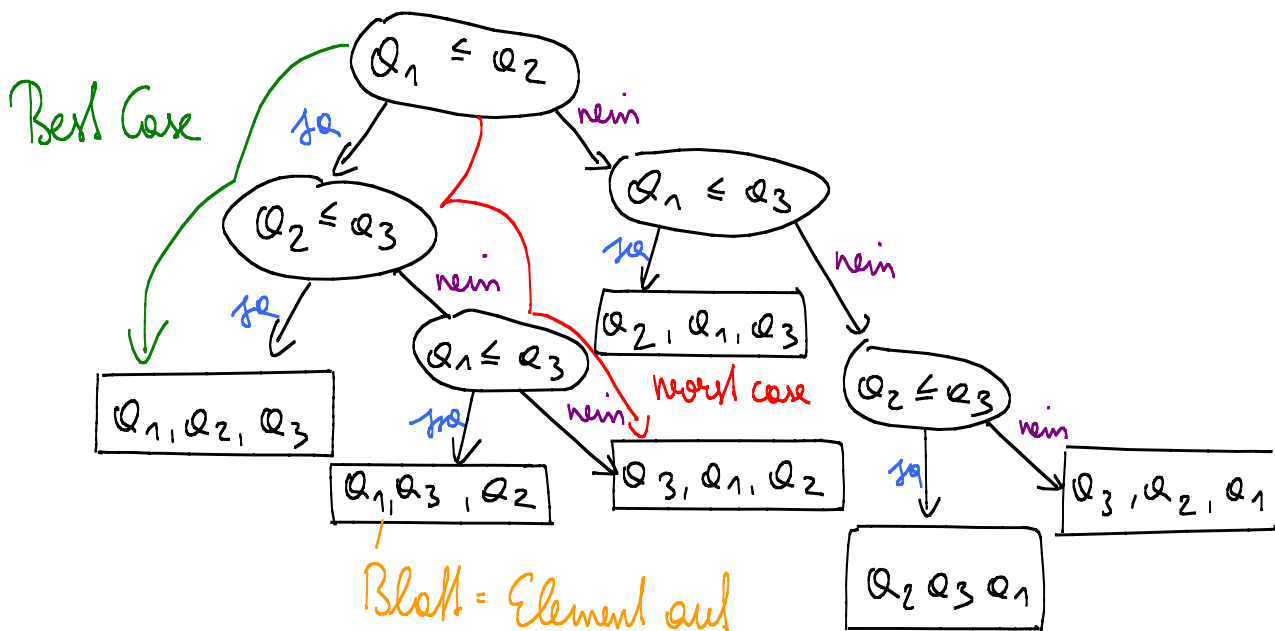
Erklärung siehe C-Code;

2. Beweisen Sie eine untere Schranke von $\Omega(n \log n)$ für die Anzahl der Vergleiche die jeder Algorithmus machen muß der dieses Problem löst.

Untere Schranke für vergleichende Sortieralgorithmen:

Entscheidungsbaum:

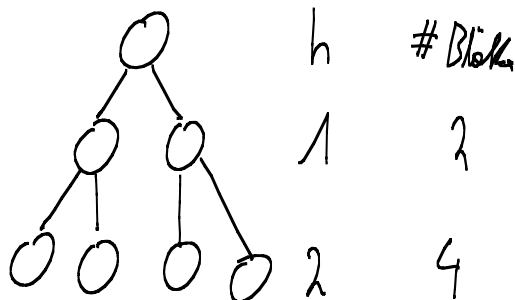
a_1, a_2, a_3



Blatt = Element auf unterster Ebene

optimal: Voller Binärbaum; d.h. jeder Weg von Root bis Blatt ist gleich

$$\Rightarrow h \geq \log(\# \text{Blätter})$$



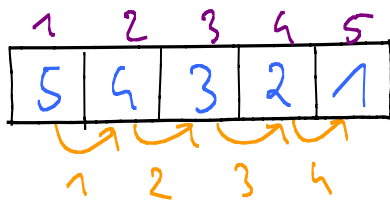
Anzahl der Sortiermöglichkeiten $\# = n!$

$$h \geq \log(n!) \quad \text{Stirling Abschätzung } n! > \left(\frac{n}{e}\right)^n$$

$$h \geq \log\left(\frac{n}{e}\right)^n = n \log(n) - \underbrace{n \log(e)}_{\text{const.}} = \Omega(n \log(n))$$

3. Bubblesort sortiert ein Feld A indem es benachbarte Elemente vertauscht und hat worst-case Laufzeit von $\Omega(n^2)$. Diese untere Schranke gilt für alle Sortierverfahren die nur benachbarte Elemente vertauschen. Zeigen Sie, dass jeder Sortieralgorithmus, der nur dadurch Feldelemente bewegt, dass er benachbarte Elemente vertauscht, eine worst-case Laufzeit von $\Omega(n^2)$ hat.

Argumentieren Sie mit Worten. Ihre Argumente müssen allerdings klar nachvollziehbar und stichhaltig sein.



\Rightarrow Ziel ist absteigend zu sortieren
= Worst Case

$$n-1 + n-2 + \dots + 1 \Rightarrow \sum_{i=1}^{n-1} i = \frac{n(n+1)}{2} - n$$

$$= \frac{n^2 + n}{2} - n$$

$$T(n) = \Omega\left(\frac{n^2}{2}\right) - \Omega\left(\frac{n}{2}\right)$$

$$T(n) = \Omega(n^2)$$

Da es sich um 2 Schleifen handelt, wobei:

die erste Schleife fix alle Elemente durchlaufen muss = $W(n)$

die zweite fuer das erste Element min $n-1$ Durchlaeufer hat = $W(n)$

4. Zeigen Sie, dass keine Datenstruktur S existieren kann, die gleichzeitig folgende Operationen unterstützt (dabei bezeichnet n jeweils die Anzahl der in S enthaltenen Elemente):

- Ein gegebenes Element in S einfügen in $O(1)$ Zeit
==> Einfügen in $O(1)$ ist möglich;
==> S ist jedoch danach nicht mehr sortiert bzw. erfüllt die Datenstrukturbedingung nicht mehr;
==> wodurch das Auffinden des Min in $O(1)$ nicht mehr möglich ist;
==> kann die Datenstruktur ein neues Element in $O(1)$ richtig einfügen, bedeutet dies eine Sortierzeit von $O(1)$, was der im 2. Bsp. gezeigten Schranke von $O(n \cdot \log(n))$ widerspricht
- Das Minimum in S finden in $O(1)$ Zeit
==> nur möglich wenn S irgendwie sortiert;
- Das Maximum in S finden in $O(n)$ Zeit
==> geht immer, da S einmal durchlaufen wird
- Das Minimum in S löschen in $O(\log(\log(n)))$ Zeit
==> geht ev; funktioniert aber nicht wenn in $O(1)$ neue Elemente eingefügt werden sollen;
==> was der im 2. Bsp. gezeigten Schranke von $O(n \cdot \log(n))$ widerspricht
- Das Maximum in S löschen in $O(n)$ Zeit
==> geht immer, da S einmal durchlaufen wird zum auffinden
+ $O(1)$ zum löschen + Datenstruktur wieder aufbauen, es gibt