

Algorithmen und Datenstrukturen - Hausübung 03

Gruppenmitglieder

- Emre Berber (2957148)
- Christoph Berst (2743394)
- Jan Braun (2768531)

Inhaltsverzeichnis

H1	1
a)	1
b)	1
c)	1
H2	1
a)	1
b)	1
c)	2
H3	2
a)	2
b)	2
c)	2

H1

a)

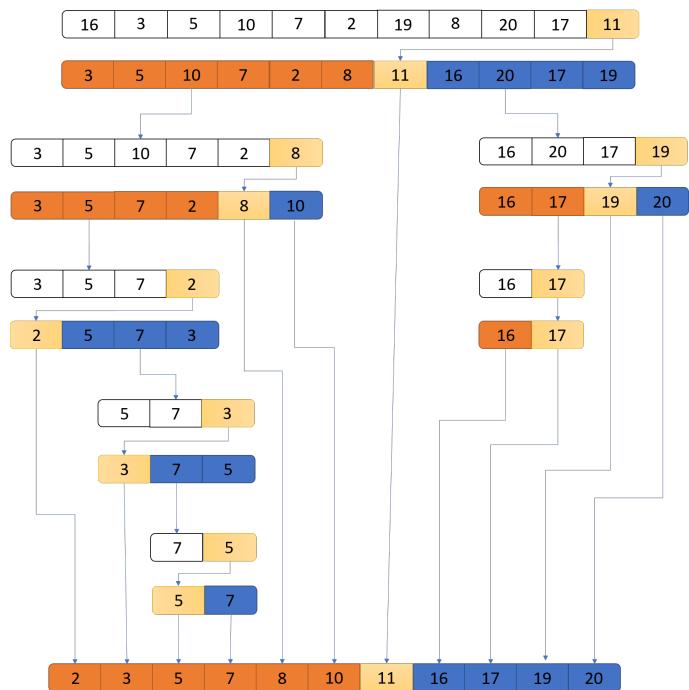
$$\frac{(n-1) \cdot (n+2)}{2} \in O(n^2)$$

Je nachdem, ob in Zeile 4 vom Partition-Algorithmus (s. Foliensatz 02 - Seite 220-284) $<$ oder \leq steht, durchläuft i das Teilarray gar nicht oder komplett. Auf jeden Fall steht i nach Partition an einer der Enden des Teilarrays, weswegen der nächste Rekursionsaufruf unbalanciert mit (Anzahl der Elemente im aktuellen Teilarray) -1 Elementen geschieht. Das ganze geschieht $(n-1)$ mal, da man den letzten Fall bei dem das Teilarray nur noch aus einem Element besteht nicht betrachten muss.

Wenn dieses Ablaufen der Elemente nun graphisch betrachten, haben wir nun ein Dreieck. Daher kommt auch die Multiplikation mit $(n+2)$ und $\frac{1}{2}$. Die $(n+2)$ entstehen durch das aneinander legen der Dreiecke. Zur Erinnerung der letzte Rekursionsaufruf hat 2 Elemente.

b)

Teilarrayzerlegung siehe Bild rechts



c)

Best-Case:

[2, 11, 7, 19, 3, 16, 8, 20, 5, 17, 10]

23 Vergleiche

Worst-Case:

[2, 3, 5, 7, 8, 10, 11, 16, 17, 19, 20]

55 Vergleiche

H2

a)

```
Find-Modal-M(A, l, r)
1  // l = 0
2  // r = A.length - 1
3  WHILE true DO
4      m = l + (r - l) / 2
5      IF A[m] > A[m+1] THEN
6          IF m == 1
7              return m
8          r = m
9      ELSE
10         IF m == 1
11             return m+1
12         left = m+1
```

Laufzeit bestimmt sich da dadurch, dass es sich durch die Zeilen 4, 8 und 12 das betrachtete Array immer halbiert wird. $O(\log n)$

Schleifeninvariante: Zwischen l und r liegt immer m.

Initialisierung: In jedem Modalenarray gibt es ein m.

Fortsetzung: Da m zwischen dem Auf- und dem Absteienteil liegt, muss m in der linken Hälfte liegen, wenn $A[m] > A[m+1]$ absteigen ist, und in der rechten Hälfte, wenn $A[m] \leq A[m+1]$ aufsteigend ist.

Terminierung: Sobald das betrachtete Array nur noch aus 2 Elementen besteht, wird das jeweils größere Element zurückgegeben.

b)

```
BIGGEST-X()
1  RETURN Find-Modal-M(V, 0, V.length - 1)
```

Die Laufzeit sollte identisch mit der Aufgabe a sein. Da das Polygon konvex ist, ist das Array V auch modal.

c)

```
BIGGEST-Y()
1  RETURN Find-Modal-M(W, 0, BIGGEST-X())
```

Da hier einfach nur 2 mal Algorithmen mit $\log n$ ausgeführt werden, ist es immer noch in $O(\log n)$.
Die Punkten werden im Uhrzeigersinn beginnend mit

dem Punkt mit dem kleinsten x-Wert aufgelistet. Die y-Werte in W könnten nun nach dem Hochpunkt m auch einen Tiefpunkt besitzen. Deshalb betrachten wir für den größten y-Wert nur den Halbkreis zwischen 0 und dem Punkt, mit dem größten x-Wert.

H3

a)

Reflexiv:

$$\begin{aligned} \text{Sei } P &= (x, y) \\ P \trianglelefteq P &\Leftrightarrow (x \leq x) \wedge (y \leq y) \end{aligned}$$

Antisymmetrie:

$$\begin{aligned} P \trianglelefteq Q \wedge Q \trianglelefteq P &\Leftrightarrow \\ ((x_1 \leq x_2) \wedge (y_1 \leq y_2)) \wedge ((x_1 \geq x_2) \wedge (y_1 \geq y_2)) &\Rightarrow (x_1 = x_2) \wedge (y_1 = y_2) \Rightarrow P = Q \end{aligned}$$

Transitivität:

$$\begin{aligned} \text{Sei } R \in \mathbb{Z}^2 \text{ mit } R &= (x_3, y_3) \\ P \trianglelefteq Q \wedge Q \trianglelefteq R &\Leftrightarrow \\ ((x_1 \leq x_2) \wedge (y_1 \leq y_2)) \wedge ((x_2 \leq x_3) \wedge (y_2 \leq y_3)) &\Rightarrow (x_1 \leq x_3) \wedge (y_1 \leq y_3) \Rightarrow P \trianglelefteq R \\ \Rightarrow \trianglelefteq \text{ ist eine Ordnungsraktion auf } \mathbb{Z}^2 \end{aligned}$$

Sei $P = (1, 2)$ und $Q = (2, 1)$, dann ist weder $P \trianglelefteq Q \Leftrightarrow (1 \leq 2) \wedge (2 \leq 1)$ noch $Q \trianglelefteq P \Leftrightarrow (2 \leq 1) \wedge (1 \leq 2)$
Also ist \trianglelefteq keine totale Ordnung auf \mathbb{Z}^2

b)

```
Fin-All-Max(M)
1  IF IsEmpty(M) THEN
2      RETURN M
3  IF Size(M) == 1 THEN
4      RETURN M
5  p = 0
6  q = 1
7  RETURN Compare(M, p, q)
```

c)

Laufzeit: $(n - 1) \cdot n \in O(n^2)$

```
Compare(M, p, q)
1  IF p+1 > M.length THEN
2      RETURN M
3  IF q+1 > M.length THEN
4      RETURN Compare(M, p+1, 0)
5  IF a == b THEN
6      RETURN Compare(M, p, q+1)
7  IF M[p] \trianglelefteq M[q] THEN
8      Remove(M, p)
9      RETURN Compare(M, q, q+1)
10 ELSE
11     RETURN Compare(M, p, q+1)
```

$M[p] \trianglelefteq M[q] := M[p].x \leq M[q].x \wedge M[p].y \leq M[q].y$

$\text{Remove}(M, p) := \text{removes an element at position } p \text{ in } M$