

# Algorithmen und Datenstrukturen - Hausübung 07

## Gruppenmitglieder

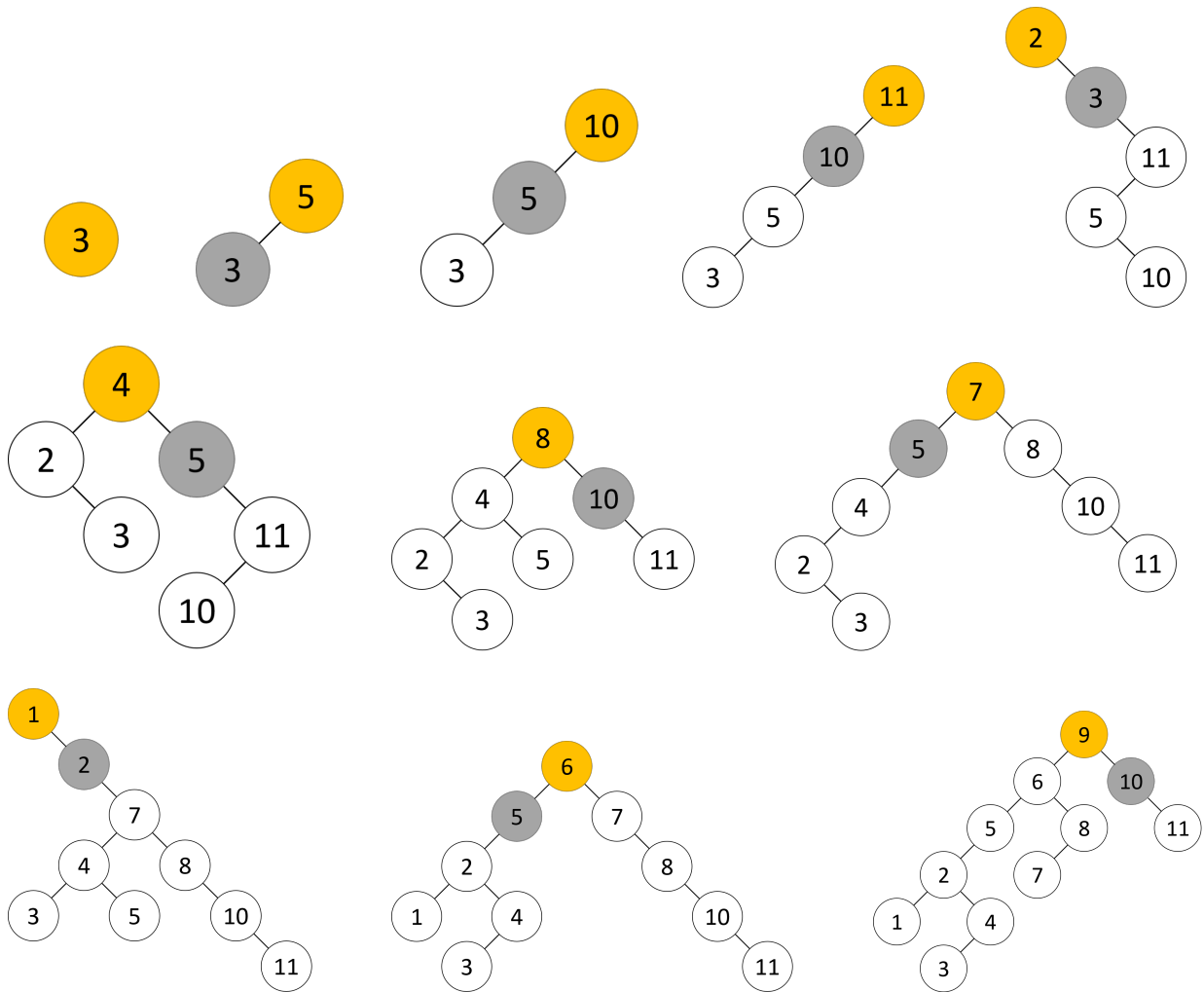
- Emre Berber (2957148)
- Christoph Berst (2743394)
- Jan Braun (2768531)

## Inhaltsverzeichnis

<b>H1</b>	<b>1</b>
a) . . . . .	1
b) . . . . .	1
<b>H2</b>	<b>2</b>
a) . . . . .	2
b) . . . . .	2
c) . . . . .	2
<b>H3</b>	<b>3</b>
a) . . . . .	3
b) . . . . .	3

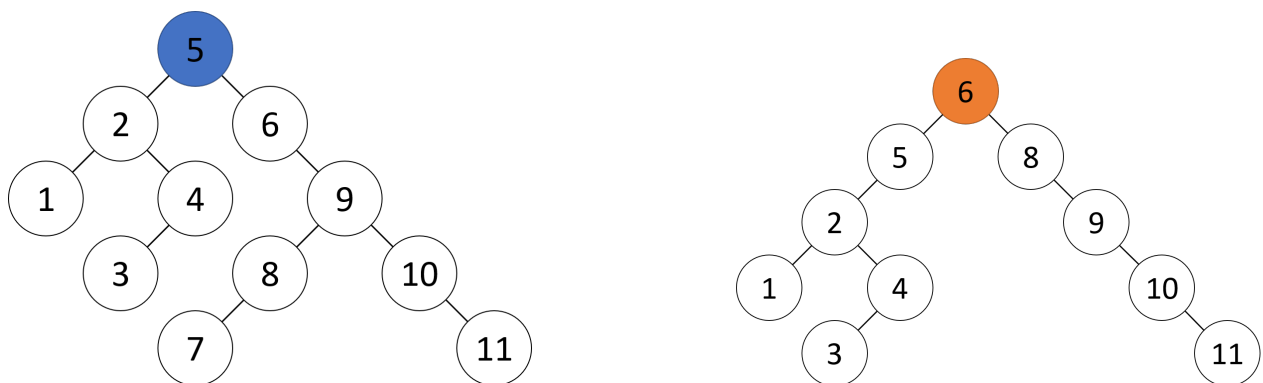
# H1

a)



grau: Elterknoten y

b)



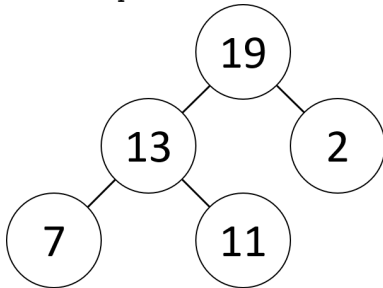
blau: gesuchter Knoten

orange: "größter" Knoten y aus L

## H2

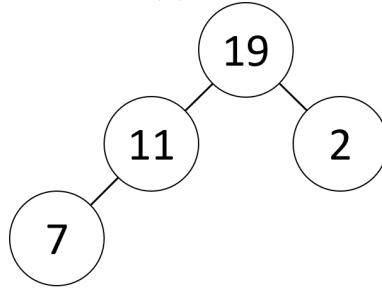
a)

BuildHeap(H.A)



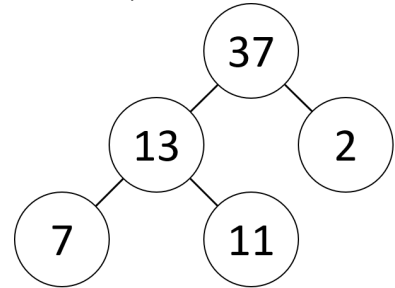
19	13	2	7	11
----	----	---	---	----

ExtractMax(H)



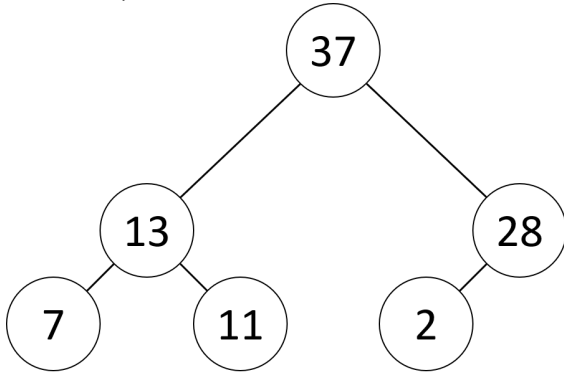
13	11	2	7
----	----	---	---

Insert(H,37)



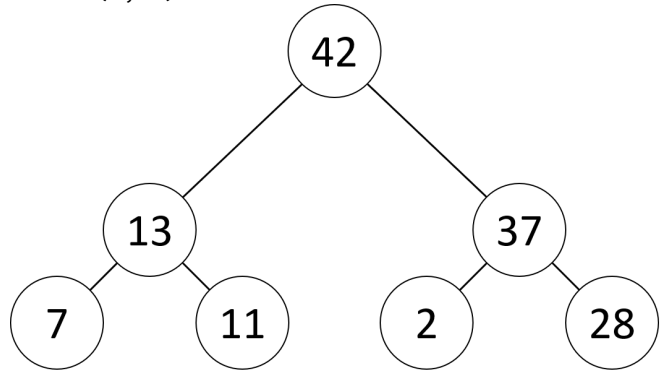
37	13	2	7	11
----	----	---	---	----

Insert(H,28)



37	13	28	7	11	2
----	----	----	---	----	---

Insert(H,42)



42	13	37	7	11	2	28
----	----	----	---	----	---	----

Nach HeapSort(H.A) ist der Baum leer und ein absteigend sortiertes Array ist entstanden.

b)

- BuildHeap(H.A) mit  $A = \{64, 16, 1, 4, 2\}$
- ExtractMax(H)
- Insert(H,32)
- ExtractMax(H)
- Insert(H,8)

c)

```

delete(H,i)
1  IF isEmpty(H) THEN
2    ERROR 'underflow'
3  IF i == 0 THEN
4    RETURN extractMax(H)
5  result = H.A[i]
6  IF i == H.size-1 THEN
7    H.size = H.size-1
8  ELSE
9    H.A = H.A[H.size-1]
10   H.size = H.size-1
11   heapify (H,i)
12  RETURN result
  
```

## H3

a)

```
minPalindrom(w)
1  count = 0
2  FOR i=0 TO w.length - 1
3      j = w.length - 1
4      WHILE true DO
5          WHILE(w[i] ≠ w[j]) DO
6              j--
7          IF i == j THEN
8              BREAK
9          IF checkPalindrom(w, i, j) THEN
10             i = j // next iteration of FOR should be i=j+1
11             BREAK
12     count++
13 RETURN count
```

```
checkPalindrom(w, i, j)
1  WHILE w[i] == w[j] DO
2      i++
3      j--
4  IF i < j THEN
5      RETURN false
6  ELSE
7      RETURN true
```

Mit  $i$  iterieren wir von links nach rechts über das Wort  $w$  und versuchen dabei jeweils mit Hilfe von  $j$  das größtmögliche Palindrom, das mit an  $i$  beginnt, zu finden. Wenn wir immer die größtmöglichen Palindrome finden, finden wir gleichzeitig auch die geringste Anzahl an Palindromen, da die Vorherigen den meisten Platz für die Folgenden wegnehmen.

b)

Korrektheit:

*Induktionsanfang:*

Betrachten für ein Wort mit **einem** Zeichen.

$i$  und  $j$  sind dann daraufhin 0. Da  $i$  und  $j$  gleich sind wird die erste WHILE-Schleife übersprungen und die TRUE-Schleife wird aufgrund der selben Aussage beendet.  $count$  wird auf 1 hochgezählt und die FOR-Schleife endet mit dem Wort. Ein Wort mit einem Zeichen hat ein Palindrom.

Betrachten wir ein Wort mit **zwei** Zeichen.

Sehen wir uns erst den Fall an in dem zweimal das selbe Zeichen verwendet wird. Dann wird wieder die WHILE-Schleife überprungen und `checkPalindrom` wird aufgerufen. Die WHILE-Schleife in `checkPalindrom` läuft einmal durch und da  $i$  nun größer als  $j$  wird "true" zurückgegeben.  $i$  wird an Ende des Palindroms gesetzt,  $count$  wird erhöht und die FOR-Schleife endet mit dem Wort. Wörter mit dem selben Zeichen bestehen aus einem Palidrom.

Seien die beiden Zeichen unterschiedlich so wird die WHILE-Schleife zum ersten Mal nicht übersprungen,  $j$  wird verringert und ist nun genauso groß wie  $i$ . Damit wird  $count$  und FOR-Schleife geht in die nächste Iteration. Nun kann man den zweiten Teil wie ein Wort aus einem Zeichen betrachten. Wörter mit komplett verschiedenen Buchstaben bestehen aus so vielen Palidromen wie sie Zeichen haben.

Gegenbeispiel: "aabba" Annahme: 2 Paliddrome a|abba; unser Algo: 3 Palindrome aa|bb|a

Komplexität:

Wenn wir von einem Wort ausgehen das vollständig aus unterschiedlichen Buchstaben besteht.

Die äußere FOR-Schleife läuft  $n$  mal. Die WHILE-Schleife läuft in jeder FOR-Schleifeniteration  $n$  mal. Damit kommen wir auf  $n^2$ .