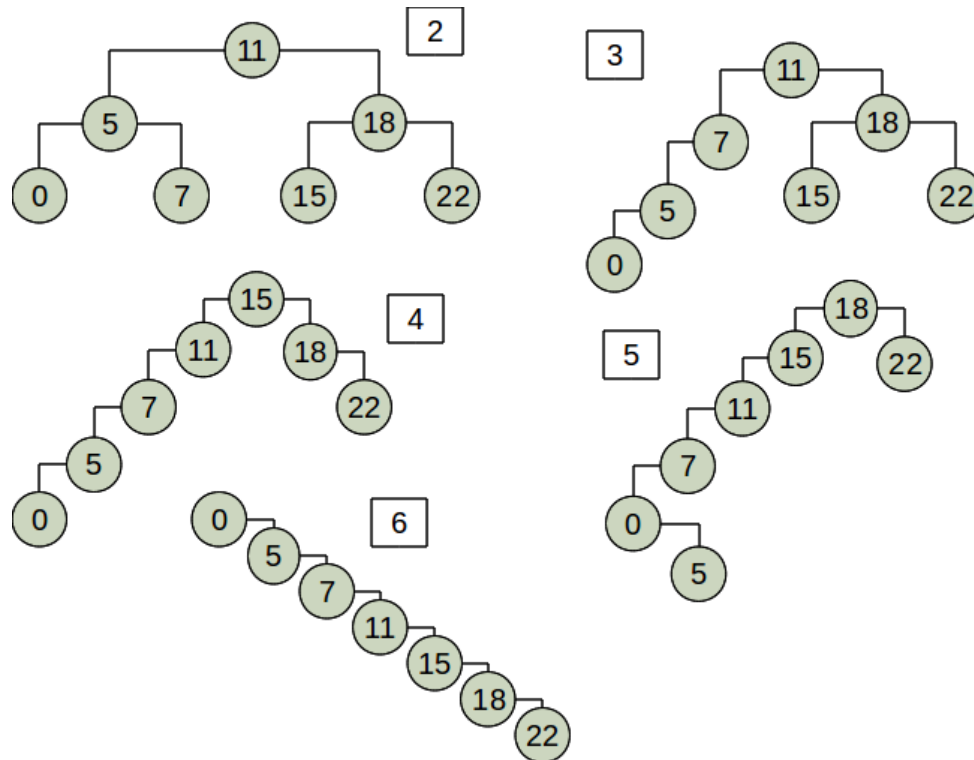


Theoretische Aufgaben: Binäre Suchbäume

Aufgabe 1




Aufgabe 2

Min-Heap-Bedingung: Die Schlüssel der Kinder eines Knotens sind stets grösser als der Schlüssel ihres Vaters.

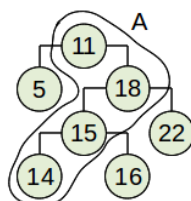
Beim Binärbaum sind im Unterschied dazu die Kinder links immer kleiner oder gleich, rechts immer grösser oder gleich.

Zur sortierten Ausgabe mit Min-Heap: Diese ist nicht möglich, da zu einem bestimmten Knoten nichts über die beiden Teilbäume ausgesagt werden kann (im Gegenteil zum binären Suchbaum).

Aufgabe 3

Möglichst kleiner Baum: 

Baum etwas vergrößert:



Im Beispiel sollte das Element mit Schlüssel 16 laut Behauptung grösser sein als z.B. Element mit Schlüssel 18 auf dem gegangenen Pfad. Dies ist aber nicht der Fall, deshalb kann die Behauptung nicht stimmen.

Aufgabe 4

Definition des Nachfolgers: Das im Sortierkriterium nächstfolgende Element. Definition von linkem Teilbaum: Element muss kleiner gleich dem Vater sein. Schluss: Der Nachfolger von x kann kein linkes Element haben, da dies zwischen den Elementen wären. Dies zerstört die Reihenfolge, der frühere Nachfolger von x wäre nicht mehr der Nachfolger.

Nimmt man an, es gäbe ein linkes Kind vom Nachfolger von x . dieses müsste in der Reihenfolge der Sortierung vor dem Nachfolger von x liegen. Da dies nicht x ist, und x schon als Vorgänger definiert ist, ergibt sich ein unlösbarer Widerspruch; Es kann kein linkes Kind geben. Analog gilt dies auch für den Vorgänger von x , der kein rechtes Kind haben kann.

Aufgabe 5

Listing 1: Aufgabe 5

```
1 treesort (array numbers)
2
3 for i = 0 to numbers.size()
4     tree-insert(numbers[i])
5
6 traverse(numbers)
```

Laufzeiten

- Best Case: Höhe des Baumes: $\Theta(\lg n)$, ausgeben $\Theta(n) \Rightarrow \Theta(n \cdot \lg n)$
- Worst Case: Höhe des Baumes: $\Theta(n)$, ausgeben: $\Theta(n) \Rightarrow \Theta(n^2)$

Theoretische Aufgaben: Repetition

Aufgabe 1

Listing 2: Aufgabe 1

```
7 concatenate (nil[a], nil[b])
8
9 //temporary variables
10 t1 = next[nil[b]]
11 t2 = last[nil[b]]
12
13 //if nil[b] doesn't point to itself,
14 //readress pointers
15 if t1 != nil[b]
16     next[last[nil[a]]] = t1 //point to beginning of b
17     last[t1] = last[nil[a]] //link to last element of a
18     next[t2] = nil[a]      //close circle
19     last[nil[a]] = t2
```

Aufgabe 2

(a) $\Theta(2^n)$, (b) $\Theta(n^5)$, (c) $\Theta(n)$, (d) $\Theta(n)$, (e) $\Theta(n)$, (f) $\Theta(n^4)$, (g) $\Theta(n \cdot \log n)$, (h) $\Theta(n \cdot \log n)$, (i) $\Theta(1)$

Reihenfolge: $\{i, e, c, d, g, h, f, b, a\}$

Aufgabe 3

Der Algorithmus hat eine Laufzeit von $\Theta(n)$. Bis auf den ersten Durchlauf wird in jedem folgenden Durchlauf zuerst die Variable j erhöht, die innere Schleife bricht ab, i wird erhöht, und die äussere Schleife beginnt von vorne.

Listing 3: Aufgabe 3

```
20 i, j = 1
21 while i <= n
22     while j <= i + 5
23         j=j+1
24     i=i+1
```

Aufgabe 4

-

Aufgabe 5

Listing 4: Aufgabe 5

```
25 minPrice(i, length, min)
26
27 if length < 0 return 0 //keine Kombination
28 if length > n return 0 //alle Platten durchsucht
29 if l = 0 return min    //Lösung gefunden
30
31 //Keine der bisherigen Fälle, schaue weiter
32 min1 = minPrice(i+1, length-length(i), min+price(i))
33 min2 = minPrice(i+1, length, min)
34
35 if min1 < min2 return min1
36 else return min2
```