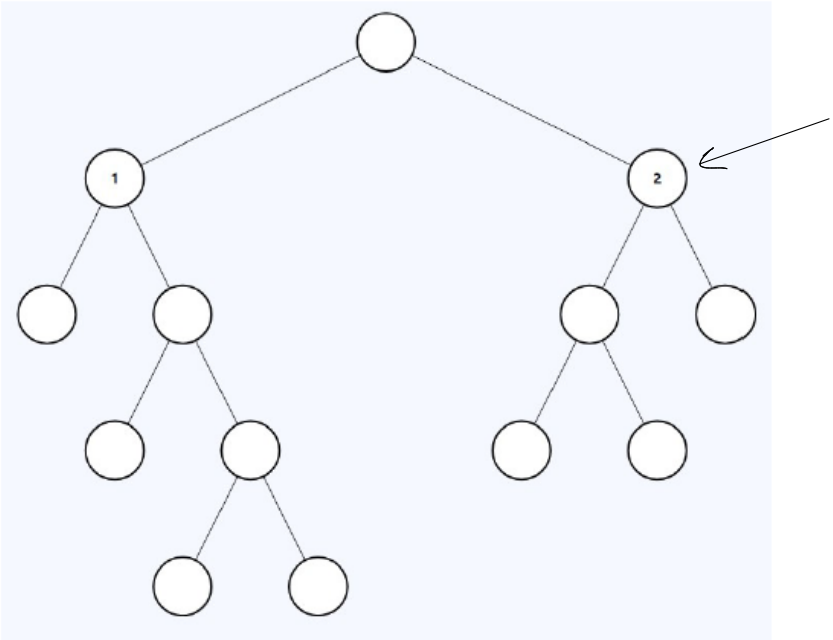
QUIZ-NACHBESPRECHUNG

True or false: In a *sorted* linked list, the search operation (as implemented in the lecture) has runtime $O(\log n)$.



In the binary tree above two nodes are labeled. Which of the labeled nodes satisfy the AVL condition?

2

True or false: An important downside of bottom-up programs compared to programs using top-down recursion is that their runtime is worse.

False

Consider the pseudocode snippet below, which implements a function $\mathrm{Fib}(n)$ which computes the n-th Fibonacci number.

```
Fib(n):
    if n ≤ 2:
        f ← 1
    else:
        f ← Fib(n − 1) + Fib(n − 2)
    return f
```

True or false: The runtime $T(n)$ of the function $\mathrm{Fib}(n)$ implemented above satisfies $T(n) \geq \Omega(n^{100})$.

True

Recall the *Jump Game* from the lecture: Given an array $A[1 \ldots n]$ of positive integers, we want to find the minimum number of *jumps* needed to reach position n starting from position 1. In each jump, we are allowed to move at most $A[i]$ steps forward, where i is our current position.

In the lecture, you saw multiple ways of solving this problem by defining a *subproblem* and a *recursive formula*.

Consider the *subproblem*: $S[i] :=$ Minimum number of jumps needed to reach position i.

Which of the following *recursive formulas* correctly computes $S[i]$?

Select one:

a. $S[i] = \max\{j + A[j] \mid 1 \leq j \leq S[i-1]\}$

b. $S[i] = \min\{1 + S[j] \mid 1 \leq j < i \text{ and } j + A[j] \geq i\}$

c. $S[i] = \min\{j + A[j] \mid S[i-2] \leq j \leq S[i-1]\}$

SERIE 4 - COMMON MISTAKES

4.3) · Immer an den Schlussatz (conclusion) bei Induktion denken!

4.4) · edge cases (n=2 nicht beachtet), z.B. `if A[m-1]<A[m] && A[m]>A[m+1]`

· finde den Fehler:

```
getPivot(A,n,l,r)
    ...
  m ← (l+r)/2
  if A[m] < A[n] then
    res ← getPivot(A,n,l,m-1)
  if A[m] > A[1] then
    res ← getPivot(A,n,m+1,r)
  return res
```

· 4.4b) war sehr kompakt lösbar

```
k ← FindPivot(A,1,n)
return binarySearch(A,l,1,k) || binarySearch(A,l,k+1,n)
```

4.5) · bei Code Snippets der Form

```
j ← 1
while ⌈∛j⌉ ≤ n do
  f()
  j ← j+1
```

kann für die Summe der Aufrufe die Schranke umgeformt werden

$$\lceil \sqrt[3]{j} \rceil \leq n \iff j \leq n^3$$

· $\sum_{i=0}^{n} r^i = \frac{(r^{n+1}-1)}{r-1} \neq \sum_{i=1}^{n} r^i$

$= \sum_{i=1}^{n} r^i + 1$

· um zu zeigen, dass etwas in $\Theta(n^n)$ liegt, muss der Grenzwert berechnet werden

· $\lim_{n\to\infty} \frac{n^{n+1}-1}{n-1} - 1 = \frac{n^{n+1}}{n} = n^n$ → falsch!

NACHBESPRECHUNG SERIE 5

Aufgabe 5.4b)

(b)* Prove that the runtime of executing Heapify(T) takes time $O(n)$.

You may use the fact that for any $k \in \mathbb{N}$

$$\sum_{i=1}^{k} \frac{i}{2^i} \leq 2$$

Hint: *Write the runtime as an outer sum over the various levels and an inner sum over all the nodes of that level.*

Algorithm 2 Heap Construction

function HEAPIFY(T)
  for $t = \text{height}(T) - 1, \ldots, 0$ **do**
    for nodes N at level t **do**
      for $\ell = t, \ldots, \text{height}(T) - 1$ **do**
        $C_1 \leftarrow$ the left child of N, if no such child exists assign it key $-\infty$.
        $C_2 \leftarrow$ the right child of N, if no such child exists assign it key $-\infty$.
        if $\text{key}(C_1) \geq \text{key}(C_2)$ and $\text{key}(C_1) > \text{key}(N)$ **then**
          Swap the keys of nodes N and C_1.
          $N \leftarrow C_1$
        else if $\text{key}(C_1) < \text{key}(C_2)$ and $\text{key}(C_2) > \text{key}(N)$ **then**
          Swap the keys of nodes N and C_2.
          $N \leftarrow C_2$
        else
          Exit inner for loop

$O(f(t))$

$f(t) : h(T) - t$

$$\sum_{t=0}^{h(T)-1} \sum_{\substack{N: \text{node} \\ \text{of level } t}} O(f(t)) \leq c \cdot \sum_{t=0}^{h(T)-1} \underbrace{\sum_{N:\ldots} f(t)}_{\leq 2^t \cdot f(t) = \underbrace{2^{t+f(t)}}_{2^{h(T)}} \cdot \frac{f(t)}{2^{f(t)}}}$$

$$\leq c \cdot \sum_{t=0}^{h(T)-1} 2^{h(T)} \cdot \frac{f(t)}{2^{f(t)}} = c \cdot 2^{h(T)} \cdot \sum_{t=0}^{h(T)-1} \frac{f(t)}{2^{f(t)}} = c \cdot 2^{h(T)} \cdot \sum_{t=1}^{h(T)} \frac{t}{2^t} \leq 2c \cdot 2^{h(T)}$$

$$= 2c \cdot n$$

$$\leq O(n)$$

$h(T) \leq \lfloor \log_2(n) \rfloor$

BINÄR- & AVL-BÄUME

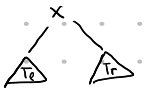
Binäre Suchbäume

Suchbedingung: für jeden Knoten x: alle Schlüssel im linken Teilbaum < x
alle Schlüssel im rechten Teilbaum > x

search und insert in O(h)

remove(x) in O(h)

1. Fall: x hat keine Kinder
2. Fall: x hat ein Kind
3. Fall: x hat zwei Kinder



x wird ersetzt durch min. Schlüssel v in T_r,
v in T_r löschen (nur 1. oder 2. Fall möglich)

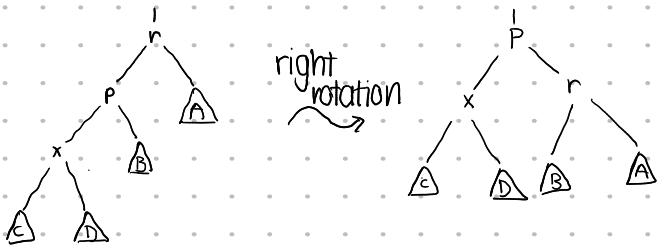
AVL-Bäume

Idee: balancierter Binärbaum mit $\underbrace{|h_\ell - h_r| \leq 1}_{\text{AVL-Bedingung}}$

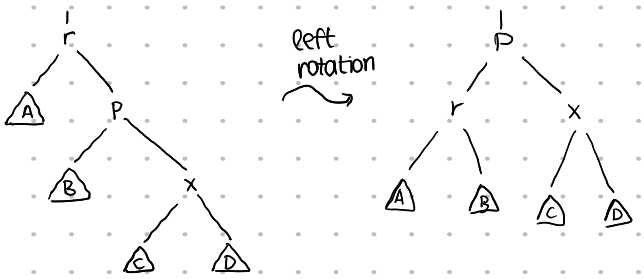
Satz: ein AVL-Baum der Höhe h hat $n \geq Fib(n+2) - 1$ Knoten

Rotation:

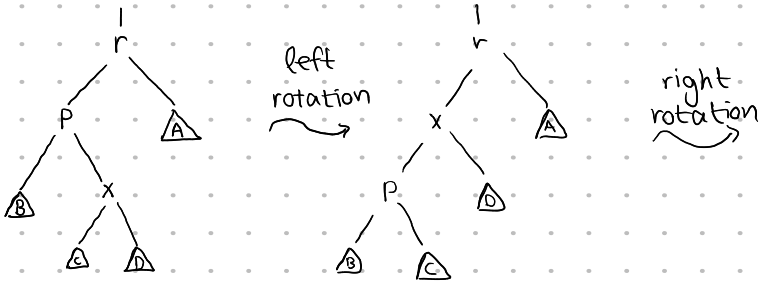
Case 1: left-left



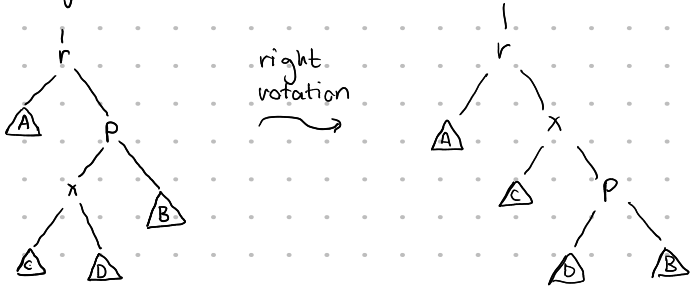
Case 2: right-right



Case 3: left-right



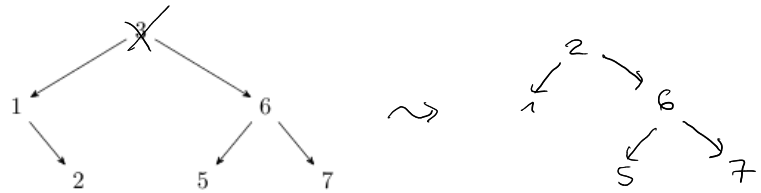
Case 4: right-left



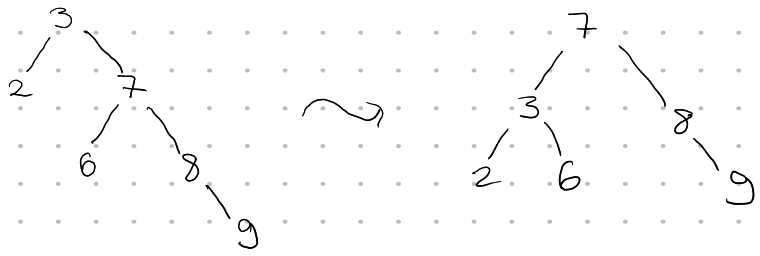
Prüfungsaufgabe (FS23/FS20) [3/60P]

~ ca. 6min Zeit

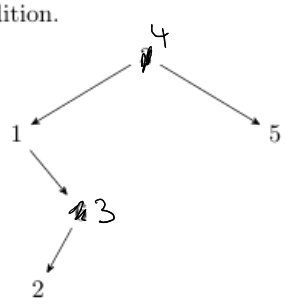
Draw the binary search tree obtained from the following tree by performing the operation DELETE(3).

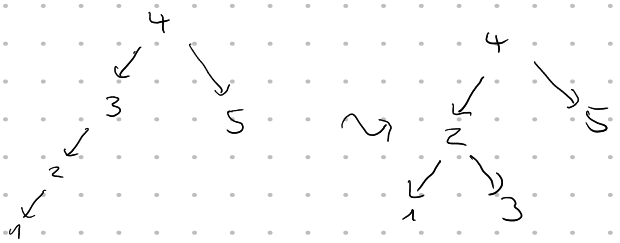


ii) Draw the **AVL tree** that is obtained when inserting the keys $3, 2, 7, 6, 8, 9$ in this order into an empty tree (it suffices to draw only the final tree).



iii) Draw the **AVL tree** that is obtained from the following tree by restoring the AVL-condition.





Dynamische Programmierung

Memoization: anstatt ein Zwischenresultat zu verwerfen, speichere es

Anwendungsbeispiel Fibonacci:

ohne Memoization: $\Omega(\Phi^n)$
mit Memoization: $O(n)$

Herangehensweise:
1. Dimensionen der DP-Tabelle finden
2. Teilprobleme / Bedeutung eines Tabelleneintrags finden
3. Rekursion / Base Cases bestimmen
4. Berechnungsreihenfolge festlegen
5. Finden der Lösung
6. Laufzeit bestimmen

Maximum Subarray Sum

$A[1...n]$

Teilproblem: $mss[i]$:= max. Summe, falls $A[i]$ das letzte Element in der Summe ist

Rekursion: $mss[i] = \max\{mss[i-1]+A[i], A[i]\}$ für $2 \le i \le n$ $\quad mss[1] = A[1]$

Auslesen der Lösung: $\max_{1 \le i \le n} \{mss[i]\}$

Laufzeit: $O(n)$

Jump Game

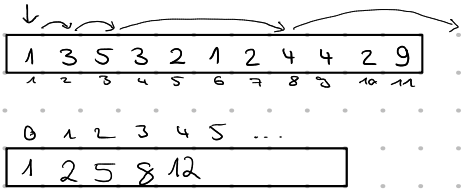
$A[1...n]$

Teilproblem: $m[k]$:= maximale Index der mit k Sprüngen erreichbar ist

Base Case: $m[0] = 1$

Rekursion: $m[k] = \max\{i + A[i] \mid m[k-2] < i \le m[k-1]\}$

Laufzeit: $O(n)$



Longest Common Subsequence

Input: 2 Arrays/Strings A[1...n] und B[1...m]

Teilproblem: L(i,j) = Länge des LCS von A[1...i] und B[1...j]

Base Case: L[i][0] = 0 für alle $1 \leq i \leq n$ L[0][j] = 0 für alle $1 \leq j \leq m$

Rekursion: $$L[i][j] = \begin{cases} 1 + L[i-1][j-1] & \text{if } A[i] = B[j] \\ \max\{L[i-1][j], L[i][j-1]\} & \text{else} \end{cases}$$

Auslesen der Lösung: L[n][m]

Laufzeit: O(n·m)

L(i,j)	-	K	A	T	Z	E
-	0	0	0	0	0	0
P	0	0	0	0	0	0
L	0	0	0	0	0	0
A	0	0	1	1	1	1
T	0	0	1	2	2	2
T	0	0	1	2	2	2
E	0	0	1	2	2	3

Edit Distance

Input: 2 Arrays/Strings A[1...n], B[1...m]

Output: minimale Anzahl an benötigten Operationen (Löschen, Einfügen, Ändern), so dass A=B

Teilproblem: ED[i][j] = minimale Anzahl an edits mit Strings A[1...i] und B[1...j] so dass A=B

Base Case: ED[0][j] = j ED[i][0] = i

Rekursion: $$ED[i][j] = \begin{cases} \min\{ED[i-1][j], ED[i][j-1], ED[i-1][j-1]\} + 1 & \text{if } A[i] \neq B[j] \\ \min\{ED[i-1][j]+1, ED[i][j-1]+1, ED[i-1][j-1]\} & \text{else} \end{cases}$$

Auslesen der Lösung: ED[n][m]

Laufzeit: O(n·m)