Aufgabe 1: Verschiedenes

a) ***Nennen*** Sie 2 grundsätzlich verschiedene Ansätze um einen **Stack** zu implementieren!

b) Gegeben sei ein Integer-Array der Größe *n*. Sie möchten feststellen, ob ein gegebener Integer-Wert in diesem Array enthalten ist. Was ist die ***asymptotische Laufzeit*** (Landau-Notation) im ***Worst Case*** in Abhängigkeit der Größe *n* des Arrays?

- falls die Integer-Werte im Array ***nicht sortiert*** sind?
- falls die Integer-Werte im Array ***aufsteigend sortiert*** sind?

Geben Sie jeweils die ***Laufzeit*** sowie eine ***kurze Begründung*** an! Nehmen Sie jeweils an, dass Sie den ***optimalen*** Algorithmus wählen.

c) Der abstrakte Datentyp (ADT*)* ***Priority Queue*** speichert Key-Value Paare und definiert die folgenden Operationen:

- `insert(k, v)`: Fügt den Wert *v* mit dem Schlüssel *k* in die Priority Queue ein.
- `removeMin()`: Entfernt Eintrag mit kleinstem Schlüssel und gibt ihn zurück. („*return*“)
- `size()`: Liefert die Anzahl der Einträge in der Priority Queue.

2 einfache Implementierungen einer Priority Queue sind die ***unsortierte Liste***, die neue Einträge immer am Ende der Liste hinzufügt, und eine ***sortierte Liste***, die alle Einträge nach ihren Schlüsseln sortiert. Die 3. Implementierung aus der Vorlesung verwendet einen ***MinHeap***.

Tragen Sie in die folgende Tabelle die Worst-Case Laufzeiten (**$\mathbf{O}$-*Notation***) ein für die 3 Operationen `size, insert, removeMin` abhängig davon welche Implementierung verwendet wird![1]
(***ohne Begründung***)

	Unsortierte Liste	Sortierte Liste	MinHeap
`size`			
`insert`			
`removeMin`			

d) Stimmt die folgende Aussage? ***(knapp begründen)***
*„**Jedes** Integer-Array, das aufsteigend sortiert ist und das keine doppelten Werte enthält, ist ein MinHeap“*

}

[1] Gehen Sie davon aus, dass die Implementierung hinreichend vernünftig ist.

Aufgabe 2: Sortieren

a) Nennen Sie ***mindestens 2*** vergleichsbasierte Sortierverfahren deren asymptotische Laufzeit im Worst Case $\Theta(n \log n)$ ist!

b) Gegeben sei das Array $A =< 5, 2, 3, 4 >$. Dieses Array soll ***aufsteigend*** mit dem ***Heapsort***-Algorithmus der Vorlesung sortiert werden. Geben Sie die **Belegung** des Arrays nach ***JEDER EINZELNEN*** Änderung durch den Algorithmus in der folgenden ***Tabelle*** an!
Hinweis:
- Fertigen Sie ggfs. ähnlich wie in der Übung Zeichnungen an (Rückseiten verwenden). ***Vergessen Sie nicht*** das Ergebnis in die Tabelle zu übertragen.
- Als Gedächtnisstütze ist auszugsweise der Pseudocode der Methoden `HEAPSORT(.)` und `BUILD-MAX-HEAP(.)` aus der Vorlesung vorgegeben.

5 2 3 4 (Start)

```
HEAPSORT(A,n)
1  BUILD-MAX-HEAP(A,n)
2  for i = n-1 downto 1
3      exchange(. . .)
4      MAX-HEAPIFY(. . . )
```

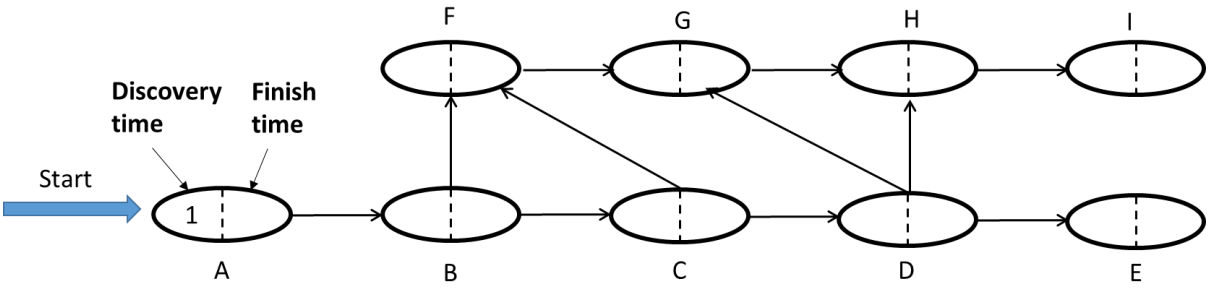
```
BUILD-MAX-HEAP(A,n)
1      for i = ⌊n/2⌋ downto 0
2          do MAX-HEAPIFY(. . . )
```

c) Angenommen, die Eingabearrays z.B. $A =< 1, 2, 3, 4, 5, 6 >$ seien bereits aufsteigend sortiert. Prof. Kluge behauptet, dass in einem solchen Fall die asymptotische Laufzeit des gegebenen Heapsort-Algorithmus ***konstant*** sei. Hat er Recht? Begründen Sie Ihre Antwort knapp in 1-2 Sätzen!

Aufgabe 3: Graphen

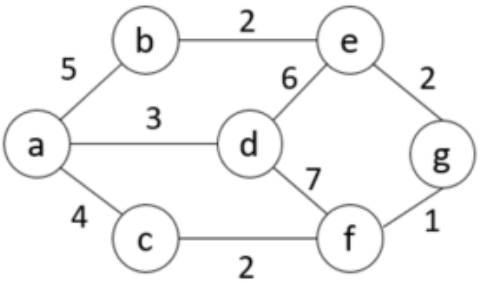
a) Führen Sie eine ***Tiefensuche*** auf dem abgebildeten gerichteten Graphen durch. Es genügt**,** wenn Sie als Ergebnis in die folgende Abbildung die ***Discovery*** und die ***Finish Times*** eintragen!

Verwenden Sie dazu den Algorithmus der Vorlesung. Gehen Sie davon aus, dass der Algorithmus Knoten stets in ***alphabetisch aufsteigender*** Reihenfolge besucht, die Adjazenzlisten jedes Knoten seien also ***alphabetisch sortiert***. Es wird mit dem ***Knoten A*** begonnen.



b) Was versteht man unter „topologischer Sortierung“?

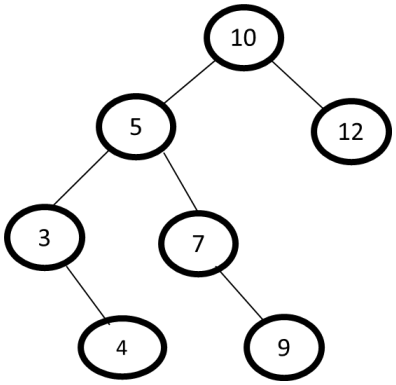
c) Betrachten Sie den rechten Graphen. Führen Sie den Dijkstra-Algorithmus mit dem ***Knoten a als Startknoten*** aus. Halten Sie sich an den Pseudocode der Vorlesung, der zur Erinnerung angegeben ist (dort ist der Parameter `s` der Startknoten). Geben Sie nur den Inhalt der Menge ***Q*** an, und zwar jeweils ***nach*** Ausführen der Zeile 8! (3P)




```
DIJKSTRA(G, w, s)
1    for each vertex v ∈ V
2        v.d = ∞
3        v.π = NIL
4    s.d = 0
5    S = ∅
6    Q = V
7    while Q ≠ ∅
8        u = EXTRACT-MIN(Q)
9        S = S ∪ {u}
10       for each vertex v ∈ G.Adj[u]
11           if v.d > u.d + w(u.v)
12               v.d = u.d + w(u,v)
13               v.π = u
```

Aufgabe 4: Binäre Suchbäume und Rot-Schwarz-Bäume

a) Gegeben sei der rechts abgebildete Baum.



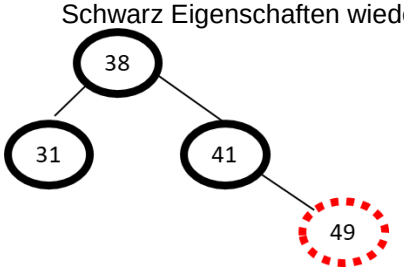
(1) Was muss erfüllt sein, damit ein Binärbaum ein binärer Suchbaum ist?

(2) Kann man den abgebildeten binären Suchbaum so einfärben, dass sich ein ***gültiger*** Rot-Schwarz-Baum ergibt?

- Falls nein, ***begründen*** Sie Ihre Antwort!
- Falls ja, ***färben*** Sie die „roten“ Knoten in der Abbildung entsprechend ein! (andere Farbe für „rot“ verwenden!)

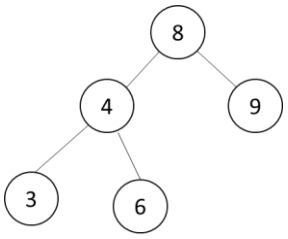
b) In den folgenden Rot-Schwarz-Baum wird ein Schlüssel mit dem Wert 43 eingefügt. Wenden Sie das in der Vorlesung besprochene Verfahren an!

- ***Zeichnen*** Sie den Rot-Schwarz-Baum ***nach*** dem Einfügen des Knotens 43, ***bevor*** die Rot-Schwarz Eigenschaften wieder hergestellt werden! (*andere Farbe für „rot“ verwenden*)
- ***Zeichnen*** Sie den Rot-Schwarz-Baum, ***nach jeder(!)*** Rotation und ***nachdem*** die Rot-Schwarz Eigenschaften wieder hergestellt wurden.



c) Als Schlüssel eines binären Suchbaumes dienen Integer. Die Knoten des binären Suchbaumes werden durch die folgende Datenstruktur abgebildet.

```
public class BTNode {     // Binary tree node
    public int item;      // data in this node
    public BTNode left;   // left subtree or null if none
    public BTNode right; // right subtree or null if none
}
```



Die Funktion **`BFS(BTNode r)` *startet beim übergegebenen Knoten `r`*** und durchläuft den binären Suchbaum ähnlich wie bei einer Breitensuche. Die Funktion soll die Schlüsselwerte in der sogenannten ***Level-Order*** mit `System.out.print(.)` ausgeben.
Beispiel: Die Ausgabe für den rechten Baum ist: 8 – 4 – 9 – 3 – 6.

Vervollständigen Sie den Code von `BFS(BTNode r)`! Nehmen Sie an, die Klasse ***Queue*** wäre bereits implementiert und verfügt über die folgenden Operationen:

- `void enqueue(int i)`: Fügt `i` der Queue hinzu.
- `int dequeue()`: Entfernt ein Element aus Queue und gibt den entfernten Schlüssel zurück.
- `boolean isEmpty()`: Ist noch ein Element in der Queue enthalten?

Halten Sie sich soweit als möglich an die Java-Syntax und verwenden Sie diese Klasse ***Queue***!

```
public void BFS(BTNode r) {
    Queue q = new Queue();

    // Assumption: null is never placed in the queue
    if (r != null) {
        q.enqueue(r);
    }
```

```
}
```

Aufgabe 5: Dynamische Programmierung

a) Die Tribonacci-Folge T_n sei wie folgt definiert:
$T_0 = 0$
$T_1 = 0$
$T_2 = 1$
$T_n = T_{n-1} + T_{n-2} + T_{n-3}$ für $n \geq 3$.

Schreiben Sie eine Java-Methode **`long tribonacci(long n),`** die für $n \geq 0$ die Tribonacci-Zahl T_n berechnet. Anforderung: Die asymptotische Laufzeit im Worst Case soll ***linear*** sein!

```
private static long tribonacci(int n) {
    // TODO




}
```

Gegeben sei für die ***folgenden Teilaufgaben*** der Pseudocode der Vorlesung zur Lösung des ROD-CUTTING Problems. Es gelten falls nicht anders erwähnt die ***Annahmen und Notationen der Vorlesung***. Das Array ***p*** enthalte wie in der Vorlesung die Verkaufspreise für verschiedene Stablängen, während ***n*** die Länge des zu unterteilenden Stabes bezeichnet. ***r[i*]** bezeichnet den maximalen erzielbaren Erlös für einen Stab der Länge *i*.

```
BOTTOM-UP-CUT-ROD(p,n)
1       let r[0..n] be a new array
2       r[0] = 0
3       for j = 1 to n
4             q = −∞
5             for i = 1 to j
6                   q = max(q, p[i] + r[j − i])
7             r[j] = q
8       return r[n]
```

b) Die Preistabelle ***p*** für Stäbe der Länge *i* sei in der folgenden Tabelle vorgegeben. Was ist der ***maximale Erlös*** für einen Stab der Länge $n = 5$? Was sind die ***Längen der Teilstücke***, um diesen maximalen Erlös zu erzielen?

Länge i	1	2	3	4	5
Preis p_i	5	11	15	19	24

c) In Vorlesung und Übung wurde für das ROD-CUTTING angenommen, dass eine Unterteilung bzw. ein Schnitt nichts kostet. ***Die neue Annahme*** ist nun, dass jeder ***einzelne Schnitt feste Kosten der Höhe c*** verursacht. Der Erlös einer bestimmten Unterteilung des Stabes ist nun die Summe der Preise der Teilstäbe minus die Kosten für die Anzahl der Schnitte. ***Wie muss der*** Pseudocode aus Aufgabe a) ***modifiziert werden?***
Hinweise:

- Es genügt wenn Sie die geänderten Codezeilen des obigen Pseudocodes angeben.
- Achten sie darauf, dass der Code auch funktioniert, wenn Sie gar nicht zuschneiden.