1. (10 Punkte)

Es sei ein Array der Größe 5 gegeben. Sortieren Sie es mit Hilfe des Insertion-Sort-Algorithmus, so dass die kleinsten Zahlen ganz links stehen und die größten ganz rechts. Füllen Sie dazu die unten gegebene Tabelle aus, indem Sie die Änderungen pro Schritt in je eine Zeile eintragen. Prinzipiell ist dabei ein Schritt jede Veränderung, die im Array vor sich geht. Wenn aber klar ist, was dabei passiert, können Sie mehrere gleichgeartete Schritte auch zusammenfassen (z.B. Vertauschen zweier Elemente oder Verschieben mehrerer Elemente um eine Stelle nach rechts oder links, oder Umkehrung der Reihenfolge aller Elemente). Die erste Zeile der Tabelle enthält den Inhalt des unsortierten Arrays.

8	3	1	6	2
3	8	1	6	2
1	3	8	6	2
1	3	6	8	2
1	2	3	6	8

10/10

2. (12 Punkte)

(a) (6 Punkte)

Wofür steht $O(n)$? O(n) ist die O-Notation,
Für alle Funktionen f, die eine Laufzeit von n haben, also lineare Laufzeit.

Zu unscharf!

3/6

(b) (6 Punkte)

Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Was bedeutet $f = O(n \cdot \log n)$?

das ist genauer definiert!

f=O(n·log n) bedeutet, dass die Funktion f eine Laufzeit von n-maliger logarithmischer Dauer hat. Beispielsweise benötigt jeder vergleichsbasierte Sortieralgorithmus mit Eingabelänge n mindestens O(n·log n) Vergleiche, um eine Eingabe der Länge n zu sortieren.

3/6

3. (30 Punkte)

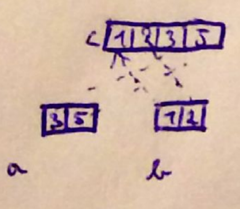
Geben Sie Mergesort in Pseudocode an. Benutzen Sie gerne mehrere Methoden/Funktionen, falls Ihnen das hilft. Gestalten Sie Ihren Pseudocode so, dass erkennbar ist, was im Algorithmus geschehen soll; Sie müssen aber (wie bei Pseudocode üblich) nicht alles bis ins allerkleinste Detail ausimplementieren. Falls Sie sich aber unsicher sind, wählen Sie lieber eine etwas genauere Darstellung als eine zu ungenaue.

```
// Array a der Eingabelänge n
merge(a) {
  // Ist die größe eines Array kleiner gleich 1 so gilt es als sortiert
  // Divide & Conquer
  if(a.length ≤ 1) return;
  // wenn dies nicht der Fall ist teile das Array bei der ungefähren Hälfte und rufe rekursiv merge auf.
  // zwei Teilarrays der größe a.length/2
  for(int i=0; i≤a.length/2; i++){
      fülle Teilarray mit der ersten Hälfte
  }
  // analog für das zweite
  merge(teilArr1);
  merge(teilArr2);
  sort(teilArr1, teilArr2, ~~ArrayInDasSortiertWird~~ a);
}

sort(a, b, c){
// 3 pointer werden erstellt die in auf die Beginne der Arrays zeigen
ap=0
bb=0
pc=0

// Zwei schon sortierte Teilarrays können nicht zusammen sortiert werden
solange cp < ausgangsArray a.length{
  Wenn [a[ap] < b[pp]) {
      füge a[ap] in c[cp] ein
      erhöhe ap
  }
  Wenn (b[pb] < a[ap]){
      füge b[pb] in [cp] ein
      erhöhe bp
  }
  erhöhe cp;
}
```

Die Teilarrays hätte man hier benennen können.

Die Variablennamen machen mitunter eine erstaunliche Metamorphose durch...

← ∀: Rest des einen Teilarrays hinzufügen



21/30

4. (20 Punkte)

Tragen Sie die asymptotischen Laufzeiten der folgenden Algorithmen jeweils im Worst-Case und im Average-Case in die untige Tabelle ein.

	Average-Case	Worst-Case
Bubblesort	$O(n^2)$ ✓	$O(n^2)$ ✓
Quicksort	$O(n \cdot \log n)$ ✓	$O(n^2)$ ✓
Mergesort	$O(n \cdot \log n)$ ✓	$O(n \cdot \log n)$ ✓
Heapsort	$O(n \cdot \log n)$ ✓	$O(n \cdot \log n)$ ✓
Insertion-Sort	$O(n^2)$ ✓	$O(n^2)$ ✓

20/20

5. (10 Punkte)

(a) (4 Punkte)

Was ist der Unterschied zwischen den Komplexitätsklassen P und NP?

NP enthält alle Entscheidungsprobleme, die von einer Nicht-deterministischen Turingmaschine in polynomialzeit gelöst werden können. Wo hingegen
P " ——— " ——— " einer deterministischen TM in polynomialzeit gelöst werden können. Und $P \subseteq NP$.

4/4

(b) (4 Punkte)

Wie ist NP-Vollständigkeit definiert?

Ein Problem ist ~~Ein~~ NP-Vollständig, wenn es in der Menge von NP ist nicht aber in der von P. D.h. es nicht auf P reduzierbar ist.

das wüssten wir gerne...

1/4

(c) (2 Punkte)

Benennen Sie ein NP-vollständiges Problem!

Bagpacking-Problem. / Rucksack Pack-Problem.

2/2

Scanned by CamScanner

6. (30 Punkte)

Es sei die folgende Adjazenzmatrix gegeben:

$$\begin{array}{c|c} & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

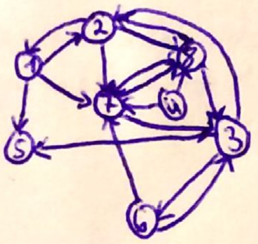
(a) (4 Punkte)

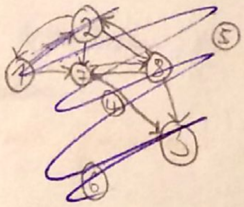
Handelt es sich bei dem durch die Matrix beschriebenen Graphen um einen gerichteten oder einen ungerichteten? Begründen Sie Ihre Antwort!

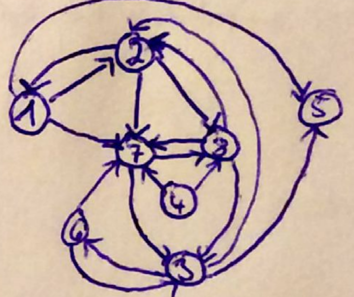
Um einen ~~un~~gerichteten Graphen. Bei einem ungerichteten Graphen wäre die Adjazenzmatrix an der Diagonalen gespiegelt, 4/4

(b) (18 Punkte)

Zeichnen Sie den durch die Matrix definierten Graphen auf! Benennen Sie die Knoten fortlaufend mit den natürlichen Zahlen $1, 2, \ldots, n$, wobei n die durch die Matrix vorgegebene Größe des Graphen ist.





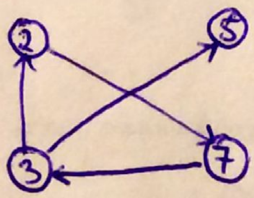


=> nochmal größer & schöner für bessere Lesbarkeit

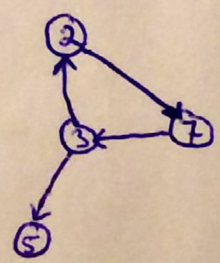
18/18

(c) (8 Punkte)

Mit dieser Knotenbenennung, geben Sie den durch die Knoten 2, 3, 5, 7 induzierten Untergraphen an!



=) "–"



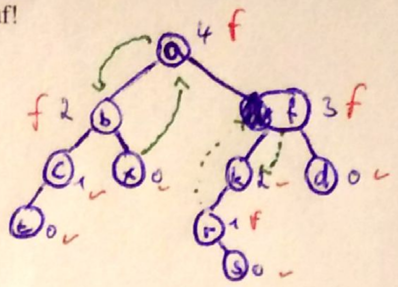
8/8

7. (40 Punkte)

Durch den Ausdruck $(a : (b : (c : t, \texttt{null}), x), (f : (k : (r : \texttt{null}, s), \texttt{null}), d))$ sei ein Binärbaum definiert.

(a) (10 Punkte)

Zeichnen Sie den Baum auf!



10/10

(b) (10 Punkte) 6/10

Fügen sie in Ihrem Diagramm zu jedem Knoten den Wert hinzu, auf den er von der avl-Funktion abgebildet wird!

(c) (4 Punkte)

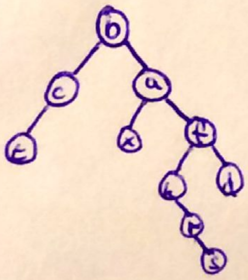
Handelt es sich bei dem Baum um einen AVL-Baum? Begründen Sie Ihre Aussage!

Ja es handelt sich um einen AVL-Baum, da die Tiefe der Unterbäume sich um höchstens 1 unterscheiden.

Stimmt das gar nicht! f 2/4

(d) (6 Punkte)

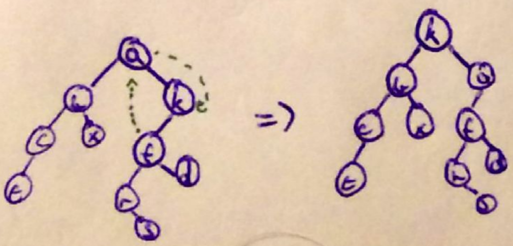
Welcher Baum entsteht, wenn um b Rechtsrotiert wird?



6/6

(e) (10 Punkte)

Welcher Baum entsteht, wenn um k doppelt Linksrotiert wird?



2/10