Aufgabe 1 (AVL-Bäume) (5 Punkte)

Fügen Sie die folgenden Zahlen nacheinander in einen *AVL-Baum* ein:

110 22 12 33 35 8

Zeichnen Sie den Baum vor und nach jeder durchgeführten Rotation. Geben Sie auch jeweils an, was für Rotationen Sie durchführen.

Aufgabe 2 (Heaps) **(5 Punkte)**

Fügen Sie die folgenden Zahlen nacheinander in einen *Min-Heap* ein:

70 12 30 8 10 1 45

Zeichnen Sie den Baum vor und nach jedem vollständigen Einfügen.

Aufgabe 3 (Sortieralgorithmen) **(5 Punkte)**

Sortieren Sie die folgende Liste von Zahlen aufsteigend mit dem Verfahren *Bubble Sort*:
Geben Sie die Liste nach jedem Durchlauf der inneren Schleife an.

Aufgabe 4 (Datentypen) **(10 Punkte)**

Erklären Sie Idee und Funktionsweise des folgenden Datentyps. Welche Vor- und Nachteile sehen Sie gegenüber einem binären Suchbaum?

```
4  type Table struct {
5    keys   []string
6    values []string
7  }
8
9  func targetIndex(key string) int {
10   firstchar := key[0]
11   return int((firstchar)-'a') * 10
12 }
13
14 func (h *Table) FirstEmptyIndexForKey(key string) int {
15   for i := 0; i < 260; i++ {
16     index := (targetIndex(key) + i) % 260
17     if h.keys[index] == "" {
18       return index
19     }
20   }
21   return -1
22 }
23
24 func (h *Table) FirstMatchingIndexForKey(key string) int {
25   for i := 0; i < 260; i++ {
26     index := (targetIndex(key) + i) % 260
27     if h.keys[index] == key {
28       return index
29     }
30   }
31   return -1
32 }
33
34 func (h *Table) Put(key string, value string) {
35   index := h.FirstEmptyIndexForKey(key)
36   if index == -1 {
37     panic("Table␣is␣full")
38   }
39   h.keys[index] = key
40   h.values[index] = value
41 }
```

Aufgabe 5 (Komplexität) **(10 Punkte)**

Bestimmen Sie die Komplexität des folgenden Algorithmus zur Bestimmung des Medians eine Liste der Länge n: Geben Sie auch einen Verbesserungsvorschlag an, wie die Komplexität reduziert werden kann.

```
6  func Median(list []int) int {
7    diffs := make([]int, len(list))
8
9    for i, el := range list {
10     d := largerCount(list, el) - smallerCount(list, el)
11     diffs[i] = abs(d)
12   }
13
14   return list[smallestIndex(diffs)]
15 }
```

Gehen Sie dabei davon aus, dass die Hilfsfunktionen tun, was ihre Namen vermuten lassen:

- largerCount und smallerCount zählen die Anzahl der Elemente, die größer bzw. kleiner als ein gegebenes Element sind.
- smallestIndex gibt den Index des kleinsten Elements zurück.
- abs berechnet den Betrag einer Zahl.

Gehen Sie weiter davon aus, dass diese Hilfsfunktionen jeweils optimal sind, d.h. dass sie die bestmögliche Laufzeit für ihre Aufgabe haben.

Aufgabe 6 (Sortieralgorithmen) **(10 Punkte)**

Erläutern Sie die Funktionsweise des folgenden Sortierverfahrens:

```
4  func FooSort(a []int) {
5    pos := 0
6    for pos < len(a) {
7      if pos == 0 {
8        pos++
9      } else {
10       if a[pos-1] > a[pos] {
11         a[pos-1], a[pos] = a[pos], a[pos-1]
12         pos--
13       } else {
14         pos++
15       }
16     }
17   }
18 }
```