

[illegible]

Aufgabe 1 (3+3+4+3+2 = 15 Punkte)

Gegeben sei der folgende Algorithmus in Pseudo-Code:

```
# Eingabe
# p: Eine ganze Zahl

j = 0
result = 0
while j < p:
    result = result + (2*j+1)
    j = j + 1

return result
```

a) Geben Sie ein Struktogramm des Algorithmus an.

b) Terminiert der Algorithmus für alle Eingabewerte $p \in \mathbb{Z}$? Geben Sie eine Begründung.

- c) Berechnen Sie die Ausgabe des Algorithmus für die Eingabewerte von 0 bis 4. Gegeben Werte $p \geq 0$, welche Funktion f berechnet der Algorithmus? Geben Sie eine Vermutung ab, eine Begründung ist nicht erforderlich.

- d) Schreiben Sie den Algorithmus als rekursive Funktion `foo(p)` in Pseudo-Code.

```
function foo(p):
```

- e) Ergänzen Sie Ihre rekursive Funktion so, dass auch für negative Werte das korrekte Ergebnis $f(p)$ (siehe Aufgabe (c)) zurückgeliefert wird.

```
function foo(p):
```

Matrikelnummer: _____

Aufgabe 2 (3+3+3+3+3 = 15 Punkte)

Sind die folgenden Behauptungen korrekt? Kreuzen Sie an. Geben Sie (falls ja) eine knappe Begründung oder (falls nein) ein Gegenbeispiel an.

- a) Jeder Algorithmus mit deterministischem Ablauf ist auch terminierend. ☐ gilt ☐ gilt nicht

Begründung/Gegenbeispiel:

-
- b) Alle vergleichsbasierten Sortierverfahren besitzen einen Worst-Case-Aufwand von $\Theta(n^2)$. ☐ gilt ☐ gilt nicht

Begründung/Gegenbeispiel:

-
- c) Ein Algorithmus der Aufwandsklasse $\Theta(n)$ benötigt immer weniger Rechenschritte als ein Algorithmus der Aufwandsklasse $\Theta(n^2)$. ☐ gilt ☐ gilt nicht

Begründung/Gegenbeispiel:

-
- d) Backtracking-Verfahren finden immer die global optimale Lösung eines Problems. ☐ gilt ☐ gilt nicht

Begründung/Gegenbeispiel:

-
- e) Bei einfach verketteten Listen gehören sämtliche Standard-Operationen zur Aufwandssklasse $O(1)$. ☐ gilt ☐ gilt nicht

Begründung/Gegenbeispiel:

Matrikelnummer: _____

Aufgabe 3 (5+4+4 = 13 Punkte)

- a) Geben Sie rechts jeweils eine Funktion an, die beide Bedingungen auf der linken Seite erfüllt. Sollte keine solche Funktion existieren, markieren Sie dies durch einen Strich.

Hinweis: Es ist keine Herleitung erforderlich.

$$a(n) = \Theta(n^2) \text{ und } a(n) \in O\left(\frac{4n^6}{3n^3}\right) \quad \rightarrow a(n) = \underline{\hspace{2cm}}$$

$$b(n) = O(n \cdot \log_2(n)) \text{ und } b(n) \in \Omega(n^2) \quad \rightarrow b(n) = \underline{\hspace{2cm}}$$

$$\log_2(n) \in O(c(n)) \text{ und } c(n) \in O(2^n) \quad \rightarrow c(n) = \underline{\hspace{2cm}}$$

$$d(n) = \Theta(1) \text{ und } d(n) \in \Theta(\log(n)) \quad \rightarrow d(n) = \underline{\hspace{2cm}}$$

$$O(e(n)) \subseteq O(n^{10}) \text{ und } e(n) \in \Omega(\log(n)) \quad \rightarrow e(n) = \underline{\hspace{2cm}}$$

```

# Eingabe
# feld: Ein array
# n: Die Länge von feld
# start: Eine Position in feld

function sub(feld, n, start):

    pos = start
    max = feld[pos]

    while pos < start+10 and pos < n:
        if feld[pos] > max:
            max = feld[pos]
            pos += 1

    return max

```

```

# Eingabe
# feld: Ein array
# n: Die Länge von feld

function super(feld, n):

    s = 0
    pos = 0

    while pos < n:
        s += sub(feld, n, pos)
        pos += 10

    return s

```

- b) Gegeben seien zwei Methoden `sub()` und `super()` (welche `sub()` aufruft). Bestimmen Sie zunächst die Worst-Case-Laufzeit von `sub()`, d.h. zählen Sie die Anzahl der elementaren Operationen. Als elementare Operationen sollen gelten: Arithmetische Operationen, Vergleiche und Feldzugriffe, aber keine Zuweisungen.

- c) Bestimmen Sie die **Worst-Case-Aufwandsklasse** von `super()` in Abhängigkeit von n . Hier genügt eine Abschätzung gemäß den Rechenregeln der O-Notation. Es ist kein explizites Zählen von Einzelschritten erforderlich.

Aufgabe 4 (6+3+6 = 15 Punkte)

- a) Die erste Zeile der unteren Tabelle stellt ein Array mit 9 Elementen dar. Führen Sie eine absteigende Sortierung mittels InsertionSort durch. Tragen Sie in jeder neuen Zeile das Ergebnis nach einer weiteren Insertion ein.

[illegible]

- b) Diese Java-Implementierung realisiert einen absteigenden InsertionSort. Ist sie stabil? Geben Sie eine Begründung. Falls nein, wie müssten Sie die Implementierung ändern um Stabilität zu erreichen?

```
static void sort(int[] a) {  
    for(int i=0; i<a.length; ++i) {  
        int val = a[i];  
        int j = i;  
        while(j>0 && a[j-1]<=val) {  
            a[j] = a[j-1];  
            j--;  
        }  
        a[j] = val;  
    }  
}
```


- c) Die erste Zeile der unteren Tabelle stellt ein Array mit 8 Elementen dar. Führen Sie auf dem Array einen Mergesort durch. Sortieren Sie das Array erneut absteigend. Führen Sie in jeder Zeile eine Misch-Operation durch. Markieren Sie hierzu jeweils die beiden Bereiche die gemischt werden.

4	8	6	5	2	4	1	9
---	---	---	---	---	---	---	---

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

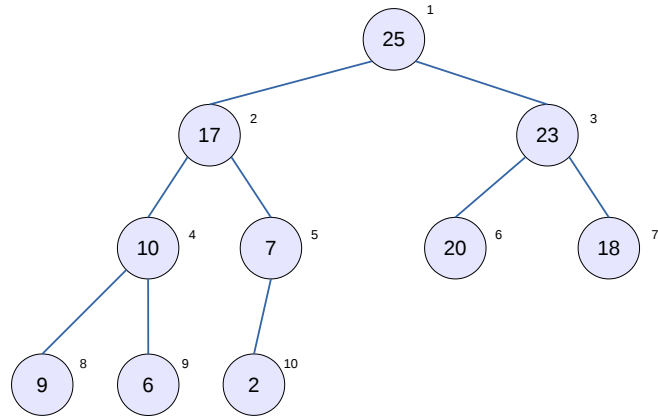
--	--	--	--	--	--	--	--

Aufgabe 5 (5+4+4 = 13 Punkte)

```
# Eingabe
# a: Array, das den Heap darstellt
# n: Anzahl der Elemente des Heaps

pos = 1
min = a[1]

while pos <= n:
    l = left(pos)
    r = right(pos)
    if r <= n and a[r] < a[l]:
        pos = r
        min = a[r]
    else if l <= n:
        pos = l
        min = a[l]
    else:
        pos = n+1
return min
```



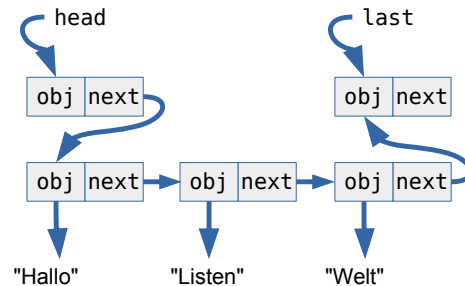
Gegeben sei ein Max-Heap wie in der Vorlesung definiert. Wir möchten das **Minimum** des Heaps finden. Hierzu schlägt Alice den obigen Algorithmus (in Pseudo-Code) vor.

Hinweise: Wie in der Vorlesung spezifiziert beginnt der Heap bei Element 1, und die Methoden `left()` und `right()` liefern die Positionen der Kindknoten.

- Welche Lösung gibt der Algorithmus für den Beispiel-Heap auf der rechten Seite zurück? Schildern Sie kurz den Verlauf des Algorithmus (welche Werte nimmt `pos` an?).
- Welchem der in der Vorlesung vorgestellten Algorithmenmuster entspricht der Algorithmus? Begründen Sie kurz.
- Findet der Algorithmus immer das Minimum des Heaps? Falls ja, begründen Sie. Falls nein, geben Sie ein Gegenbeispiel.

Aufgabe 6 (6+7 = 13 Punkte)

```
class LinkedList<T> {  
    private Node head;  
    private Node last;  
  
    private class Node {  
        T obj;  
        Node next;  
    }  
    ...  
}
```



Gegeben sei eine einfach verkettete Liste. Gemäß unserer Spezifikation aus der Vorlesung (oben links) besteht die Liste aus Knoten (Nodes) und besitzt für Anfang und Ende separate Knoten head und last.

- a) Implementieren Sie eine private Methode `get(int i)`, die den i -ten Knoten der Liste zurückgibt. Für $i = 1$ soll das 1. Element zurückgeliefert werden, für $i = 0$ der Knoten head, für $i = n + 1$ (für n -elementige Listen) der Knoten last. Für $i < 0$ und $i > n + 1$ soll eine `ListException` geworfen werden (diese brauchen Sie nicht extra zu definieren).

```
private Node get(i) throws ListException {
```

```
}
```

- b) Implementieren Sie eine Methode `swap(int j, int k)`, die die Listenelemente an den Positionen j und k vertauscht. Die Zähler j und k beginnen bei 1. Sie können außerdem davon ausgehen, dass $j < k$.

Hinweis: Verwenden Sie die Methode `get()` aus der vorherigen Aufgabe.

```
void swap(int j, int k) throws ListException
{
```

```
}
```