Aufgabe 1 (Asymptotische Komplexität): (6 + 10 + 6 = 22 Punkte)

a) Geben Sie eine formale Definition dafür an, wann für zwei Funktionen $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ und $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ gilt, dass $f(n) \in \mathcal{O}(g(n))$.

b) Gegeben seien die folgenden 5 Funktionen f_1 bis f_5:

$$\begin{aligned} f_1(n) &= \log(n) + n^2 \\ f_2(n) &= \frac{n^5}{n^3 + n^2} \\ f_3(n) &= \log(n^2) \\ f_4(n) &= n \cdot \sqrt{n} \\ f_5(n) &= (\log(n))^2 \end{aligned}$$

Füllen Sie die folgende Liste aus, indem Sie Θ eintragen, wenn die Funktion links f ist, die Funktion rechts g und es gilt $f(n) \in \Theta(g(n))$. Tragen Sie o ein, falls in diesem Szenario $f(n) \in o(g(n)) = \mathcal{O}(g(n)) \backslash \Theta(g(n))$ gilt und ω, falls $f(n) \in \omega(g(n)) = \Omega(g(n)) \setminus \Theta(g(n))$ gilt.

$f_1(n) \in$ _____ $(f_2(n))$

$f_1(n) \in$ _____ $(f_3(n))$

$f_1(n) \in$ _____ $(f_4(n))$

$f_1(n) \in$ _____ $(f_5(n))$

$f_2(n) \in$ _____ $(f_3(n))$

$f_2(n) \in$ _____ $(f_4(n))$

$f_2(n) \in$ _____ $(f_5(n))$

$f_3(n) \in$ _____ $(f_4(n))$

$f_3(n) \in$ _____ $(f_5(n))$

$f_4(n) \in$ _____ $(f_5(n))$

c) Beweisen oder widerlegen Sie: $3^n \in \mathcal{O}(n!)$

Lösung: _____

a)

$$\exists c \in \mathbb{R}^{>0}, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N} \text{ mit } n \geq n_0 : f(n) \leq c \cdot g(n)$$

Alternativ:

$$\limsup_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

b)

$$
\begin{aligned}
f_1(n) &\in \Theta(f_2(n)) \\
f_1(n) &\in \omega(f_3(n)) \\
f_1(n) &\in \omega(f_4(n)) \\
f_1(n) &\in \omega(f_5(n)) \\
f_2(n) &\in \omega(f_3(n)) \\
f_2(n) &\in \omega(f_4(n)) \\
f_2(n) &\in \omega(f_5(n)) \\
f_3(n) &\in o(f_4(n)) \\
f_3(n) &\in o(f_5(n)) \\
f_4(n) &\in \omega(f_5(n))
\end{aligned}
$$

c) **Behauptung:** Die Aussage gilt.

Beweis:
Zu zeigen: $\exists c \in \mathbb{R}^{>0}, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}$ mit $n \geq n_0 : 3^n \leq c \cdot n!$.
Es gilt für $n \geq 2$:

$$
\begin{aligned}
3^n &= \frac{9}{2} \cdot 1 \cdot 2 \cdot 3^{n-2} \\
&= \frac{9}{2} \cdot 1 \cdot 2 \cdot \underbrace{3 \cdot 3 \cdot \dots \cdot 3}_{n-2} \\
&\leq \frac{9}{2} \cdot 1 \cdot 2 \cdot \underbrace{3 \cdot 4 \cdot \dots \cdot n}_{n-2} \\
&= \frac{9}{2} \cdot n!
\end{aligned}
$$

Wähle also $c = \frac{9}{2}$ und $n_0 = 2$. Damit ist die Aussage bewiesen. □

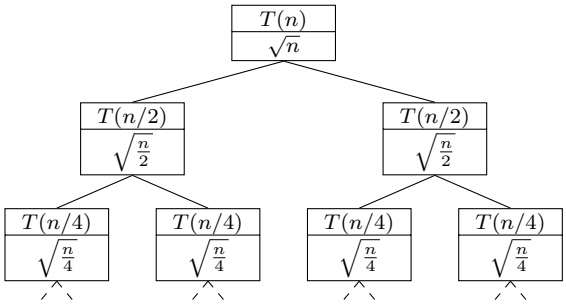
Aufgabe 2 (Rekursionsgleichungen): (6 + 7 + 10 = 23 Punkte)

a) Gegeben sei der folgende Algorithmus `int deepThought(int n, int k)`:

```
int deepThought(int n, int k) {
    if (n > 4) {
        int i = n - 4;
        while (i > 0) {
            i = i - 1;
        }
        if (k % 2 == 0) {
            return deepThought(n/3, k) + deepThought(n/3, k);
        } else {
            return 4 * deepThought(n/3, k);
        }
    }
    return 42;
}
```

Geben Sie die Rekursionsgleichung für die Worst-Case Laufzeit von `deepThought` an. Die Operationen `+`, `-`, `*`, `/` und `%` sind die elementaren Operationen und werden mit einer Laufzeit von 1 berücksichtigt. Vergleiche und Zuweisungen sind für die Berechnung der Laufzeit nicht relevant.

b) Gegeben sei folgender Rekursionsbaum:



$T(1)T(1)T(1)T(1)T(1) \quad \cdots \quad T(1)T(1)T(1)T(1)T(1)$

Ergänzen Sie die folgenden Aussagen (vereinfachen Sie die Terme so weit wie möglich):

- Die Tiefe des Rekursionsbaumes ist _____.
- Die Summe der Laufzeit aller Knoten auf Ebene i (die Wurzel ist Ebene 0) ist _____.
- Die Anzahl der Blätter im Rekursionsbaum ist _____.

c) Bestimmen Sie die Komplexitätsklasse der folgenden Rekursionsgleichung mithilfe des Mastertheorems oder begründen Sie, warum das Mastertheorem nicht anwendbar ist.

$$T(n) = \begin{cases} 2 \cdot T(\frac{n}{8}) + n^2 \cdot \log(n) & \text{für } n > 1 \\ 5 & \text{sonst} \end{cases}$$

Erinnerung Mastertheorem: $E = \log_b(a)$ (alternativ $b^E = a$) und

$f(n) \in \mathcal{O}(n^{E-\epsilon}), \epsilon > 0$ $\rightarrow T(n) \in \Theta(n^E)$

$f(n) \in \Theta(n^E)$ $\rightarrow T(n) \in \Theta(n^E \cdot \log(n))$

$f(n) \in \Omega(n^{E+\epsilon}), \epsilon > 0$ und $a \cdot f(n/b) \leq d \cdot f(n), d < 1$ $\rightarrow T(n) \in \Theta(f(n))$

Lösung:

a) $T_{\texttt{deepThought}}(n,k) = \begin{cases} 0 & \text{für } n \leq 4 \\ 2 \cdot T(n/3,k) + n + c_1 & \text{für } n > 4 \text{ und } k \text{ gerade} \\ T(n/3,k) + n + c_2 & \text{für } n > 4 \text{ und } k \text{ ungerade} \end{cases}$

b)

- Die Tiefe des Rekursionsbaumes ist $\log_2(n)$.
- Die Summe der Laufzeit aller Knoten auf Ebene i ist

$$\begin{aligned} 2^i \cdot \sqrt{\frac{n}{2^i}} &= 2^i \cdot \frac{1}{2^{i/2}} \cdot \sqrt{n} \\ &= 2^{i-i/2} \cdot \sqrt{n} \\ &= \sqrt{2^i} \cdot \sqrt{n} \\ &= \sqrt{2^i \cdot n} \end{aligned}$$

- Der Rekursionsbaum hat n Blätter:

$$2^{\log_2 n} = n$$

c) **Behauptung:** $T(n) \in \Theta(n^2 \cdot \log(n))$
Beweis:
Wir haben $a = 2, b = 8$ und $E = \frac{1}{3}$.
Wir zeigen $n^2 \cdot \log(n) \in \Omega(n^{\frac{1}{3}+\epsilon})$ für $\epsilon = 1$:

$$\lim_{n\to\infty} \frac{n^2 \cdot \log(n)}{n^{\frac{1}{3}+\epsilon}} = \lim_{n\to\infty} \underbrace{n^{\frac{5}{3}-\epsilon}}_{>0 \text{ für } n\geq 1} \cdot \underbrace{\log(n)}_{>0 \text{ für } n>1} = \infty$$

Außerdem:

$$\begin{aligned} 2 \cdot \left(\frac{n}{8}\right)^2 \cdot \log(\frac{n}{8}) &\leq d \cdot n^2 \cdot \log(n) \\ \frac{1}{32} \cdot n^2 \cdot \log(\frac{n}{8}) &\leq d \cdot n^2 \cdot \log(n) && \text{setze } d = \frac{1}{32} \\ \log(\frac{n}{8}) &\leq \log(n) && \text{für alle } n > 1 \end{aligned}$$

Somit ist die Behauptung gültig (3. Fall des Mastertheorems).

Aufgabe 3 (Abstrakte Datentypen): (14 + 8 = 22 Punkte)

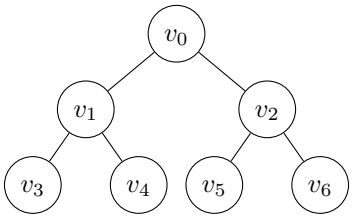
a) Implementieren Sie einen Datentypen `BinaryTree` für vollständige Binärbäume der festen Höhe h ($h \in \mathbb{N} \cup \{0\}$), welcher durch die nachfolgende Spezifikation definiert ist. Benutzen Sie bei Ihrer Implementierung genau eine Instanz eines Arrays der Größe $2^{h+1} - 1$ und beliebig viele der primitiven Datentypen `bool` und `int`.

Wertebereich:

Die vollständigen Binärbäume der Höhe h, wobei auch die letzte Ebene vollständig gefüllt ist. Die Knoten haben ganzzahlige Schlüssel vom Typ `int`. Wir definieren, dass die **Position** des j-ten Knotens von links auf der i-ten Ebene durch $2^i + j - 1$ gegeben ist, wobei wir immer ab 0 zählen.

Beispiel:

Gegeben ist ein vollständiger Binärbaum der Höhe 2.
Die Position von v_i ist dann i $(0 \leq i \leq 6)$.



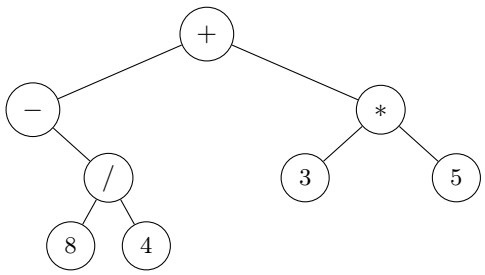
Operationen:

- Der Konstruktor `BinaryTree(int ht)` erstellt einen vollständigen Binärbaum der Höhe h = `ht`, sodass alle Knoten ihre Position als Schlüssel haben.
  Vorbedingung: `ht` ≥ 0.
- `int key(int pos)` gibt den Schlüssel des Knotens zurück, der sich an der gegebenen Position `pos` befindet.
  Vorbedingung: $0 \leq$ `pos` $< 2^{h+1} - 1$.
- `bool isLeaf(int pos)` gibt zurück, ob der Knoten an der gegebenen Position `pos` ein Blatt ist.
  Vorbedingung: $0 \leq$ `pos` $< 2^{h+1} - 1$.
- `int left(int pos)` gibt die Position des **linken** Kindes des Knotens an der gegebenen Position `pos` zurück.
  Vorbedingung: $0 \leq$ `pos` $< 2^h - 1$
- `int right(int pos)` gibt die Position des **rechten** Kindes des Knotens an der gegebenen Position `pos` zurück.
  Vorbedingung: $0 \leq$ `pos` $< 2^h - 1$
- `void replace(int pos, int key)` ersetzt den Schlüssel des Knotens, der sich an der gegebenen Position `pos` befindet, durch den gegebenen Schlüssel `key`.
  Vorbedingung: $0 \leq$ `pos` $< 2^{h+1} - 1$.

b) Implementieren Sie einen **iterativen** Suchalgorithmus `bool find(BinaryTree bt, int key)`, welcher `true` zurück gibt, falls `bt` einen Knoten mit Schlüssel `key` enthält, und sonst `false`.

Vorbedingung: Für die Inorder-Linearisierung $v_{e_0}, v_{e_1}, \ldots, v_{e_k}$ von `bt` gilt $v_{e_0} < v_{e_1} < \ldots < v_{e_k}$, wobei $k = 2^{h+1} - 2$ und h die Höhe von `bt` ist.

Erinnerung an ein Beispiel aus der Vorlesung:



Inorder-Linearisierung: $- \; 8 \; / \; 4 \; + \; 3 \; * \; 5$

Wichtig: Die **Worst-Case-Komplexität des Suchalgorithmus muss in** $\mathcal{O}(h)$ liegen (in der Anzahl der Vergleiche der Knotenschlüssel). Nutzen Sie hierbei für `bt` nur die Operationen aus Aufgabenteil a):

- `int key(int pos)`
- `bool isLeaf(int pos)`
- `int left(int pos)`
- `int right(int pos)`
- `void replace(int pos, int key)`

Lösung:

a)

```
class BinaryTree {
    // Attribute.
    int[] content;
    int height;

    // Konstruktor.
    BinaryTree(int ht) {
        content = int[pow(2, ht + 1) - 1]; // pow(a, b) = a^b
        height = ht;
        for (int i = 0; i < content.length(); i++) {
            content[i] = i;
        }
    }

    // Operationen.
    int key(int pos) {
        return content[pos];
    }

    bool isLeaf(int pos) {
        return pos >= pow(2, height) - 1;
    }

    int left(int pos) {
        return 2 * pos + 1;
    }

    int right(int pos) {
        return 2 * pos + 2;
    }

    void replace(int pos, int key) {
        content[pos] = key;
    }
};
```

b)
```
bool find(BinaryTree bt, int key) {
    // Starte mit der Wurzel
    int pos = 0;
    // Abbruchbedingung: Schluessel gefunden oder ein Blatt erreicht
    while (bt.key(pos) != key && !(bt.isLeaf(pos))) {
        if (bt.key(pos) < key) {
            // Durchsuche den rechten Teilbaum
            pos = bt.right(pos);
        } else { // Gleichheit wurde schon in der Schleifenbedingung ausgeschlossen
            // Durchsuche den linken Teilbaum
            pos = bt.left(pos);
        }
    }
    // Ueberpruefe ob die Schleife aufgrund der Gleichheit verlassen wurde
    return bt.key(pos) == key;
}
```

Aufgabe 4 (Sortierverfahren): (6 + 3 + 3 + 4 + 7 = 23 Punkte)

a) Geben Sie zu den folgenden Sortierverfahren aus der Vorlesung jeweils an, welche asymptotischen Best- und Worst-Case Laufzeiten (Θ) sie bzgl. der Anzahl an benötigten Vergleichen aufweisen und ob sie stabil sind.

Sortierverfahren	**Best-Case**	**Worst-Case**	**Stabil?**
Selectionsort			
Insertionsort			
Quicksort			
Mergesort			

b) Betrachten Sie die Klasse der Arrays mit beinahe vorsortierten Elementen. D. h. es gibt eine Anzahl n_{falsch} an Elementen, ohne die das Array sortiert wäre, und n_{falsch} kann als eine kleine Konstante angenommen werden (die Elemente an „falschen“ Positionen können sich aber beliebig weit von ihren „richtigen“ Positionen entfernt befinden). Der Wertebereich der Arrayelemente ist unendlich. Welches Sortierverfahren aus der Vorlesung sollte genutzt werden, um große Arrays aus dieser Klasse in möglichst kurzer Zeit zu sortieren? Begründen Sie Ihre Antwort kurz.

c) Betrachten Sie die Klasse der Arrays mit zufälligen ganzen Zahlen zwischen -2^7 und $2^7 - 1$. Welches Sortierverfahren aus der Vorlesung sollte genutzt werden, um große Arrays aus dieser Klasse in möglichst kurzer Zeit zu sortieren? Begründen Sie Ihre Antwort kurz.

d) Sortieren Sie das Array `[ 4,8,7,3,1 ]` durch Anwendung des Bubblesort-Algorithmus aus der Vorlesung. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

e) Sortieren Sie das Array `[ 8,7,4,3,9,1,2 ]` durch Anwendung des Heapsort-Algorithmus aus der Vorlesung. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

Lösung:

a)

Sortierverfahren	**Best-Case**	**Worst-Case**	**Stabil?**
Selectionsort	$\Theta(n^2)$	$\Theta(n^2)$	Nein
Insertionsort	$\Theta(n)$	$\Theta(n^2)$	Ja
Quicksort	$\Theta(n \cdot \log(n))$	$\Theta(n^2)$	Nein
Mergesort	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	Ja

b) Insertionsort hat für beinahe vorsortierte Arrays eine lineare Laufzeit und ist damit das schnellste Sortierverfahren aus der Vorlesung, das in diesem Szenario angewendet werden kann.

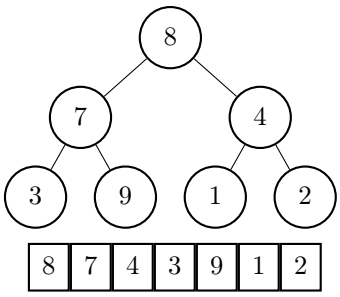
c) Countingsort hat eine lineare Laufzeit und kann bei einem endlichen Wertebereich angewendet werden. Damit ist es für das beschriebene Szenario das schnellste Verfahren aus der Vorlesung.

d)

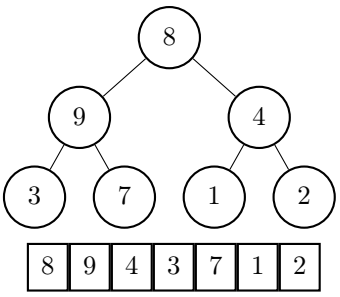
4	8	7	3	1
4	7	8	3	1
4	7	3	8	1
4	7	3	1	8
4	3	7	1	8
4	3	1	7	8
3	4	1	7	8
3	1	4	7	8
1	3	4	7	8

e)

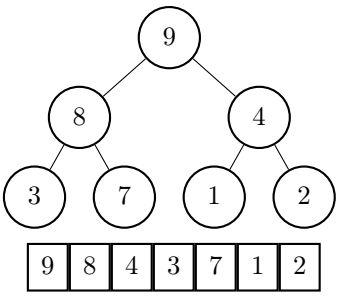
Schritt 0:



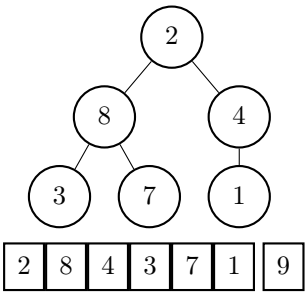
Schritt 1:



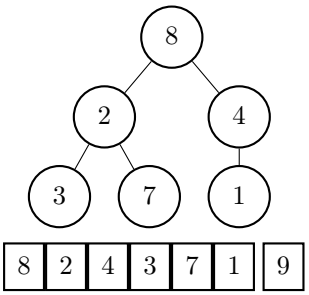
Schritt 2:



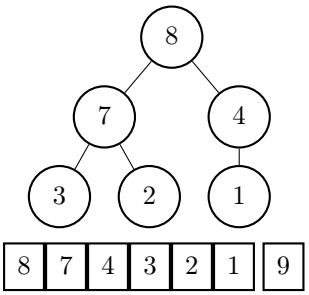
Schritt 3:



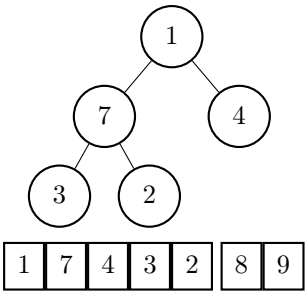
Schritt 4:



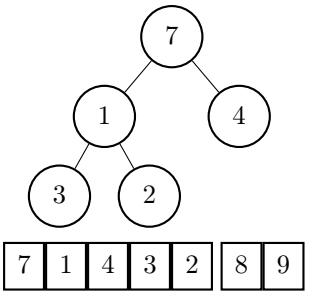
Schritt 5:



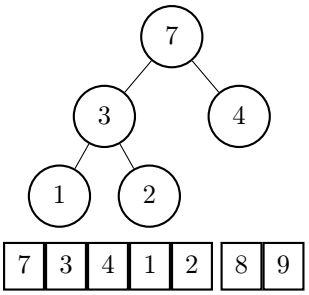
Schritt 6:



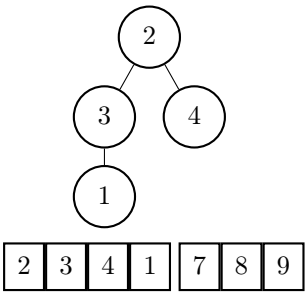
Schritt 7:



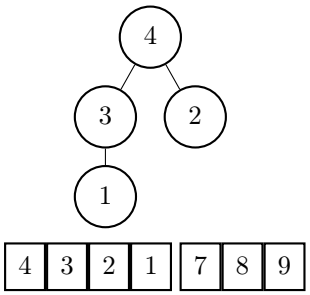
Schritt 8:



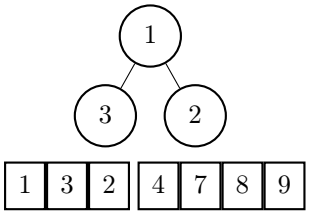
Schritt 9:



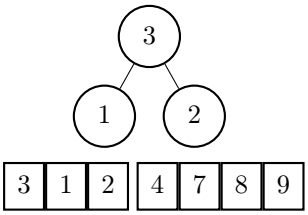
Schritt 10:



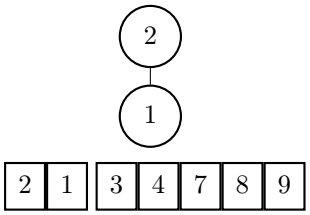
Schritt 11:



Schritt 12:



Schritt 13:



Schritt 14: