1. (3 + 3 + 3 Punkte)

   (a) (3 Punkte)
   Sei A aufsteigend sortiert.

	$O(\lg n)$	$O(n)$	$O(n \lg n)$	$O(n^2)$	keine Angabe möglich
Insertion-Sort	☐	x	x	x	☐
Merge-Sort	☐	☐	x	x	☐
Heap-Sort	☐	☐	x	x	☐
Quick-Sort	☐	☐	☐	x	☐
Bucket-Sort	☐	x	x	x	☐
Counting-Sort	☐	☐	☐	☐	x
Radix-Sort	☐	☐	☐	☐	x

   (b) (3 Punkte)
   Sei A absteigend sortiert.

	$O(\lg n)$	$O(n)$	$O(n \lg n)$	$O(n^2)$	keine Angabe möglich
Insertion-Sort	☐	☐	☐	x	☐
Merge-Sort	☐	☐	x	x	☐
Heap-Sort	☐	☐	x	x	☐
Quick-Sort	☐	☐	☐	x	☐
Bucket-Sort	☐	x	x	x	☐
Counting-Sort	☐	☐	☐	☐	x
Radix-Sort	☐	☐	☐	☐	x

   (c) (3 Punkte)
   Sei A unsortiert.

	$O(\lg n)$	$O(n)$	$O(n \lg n)$	$O(n^2)$	keine Angabe möglich
Insertion-Sort	☐	☐	☐	x	☐
Merge-Sort	☐	☐	x	x	☐
Heap-Sort	☐	☐	x	x	☐
Quick-Sort	☐	☐	x	x	☐
Bucket-Sort	☐	x	x	x	☐
Counting-Sort	☐	☐	☐	☐	x
Radix-Sort	☐	☐	☐	☐	x

2. (5 + 5 + 5 Punkte)

(a) $f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
Lösung:
Für $f(n)$ gilt:

$$\exists c_1, c_2 > 0, c_1 \leq c_2, n_0 > 0 \text{ so daß } \forall n > n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Für $g(n)$ gilt:

$$\exists c_3, c_4 > 0, c_3 \leq c_4, n_1 > 0 \text{ so daß } \forall n > n_1 : c_3 h(n) \leq g(n) \leq c_4 h(n)$$

Also gilt für $n_2 := max(n_0, n_1)$:

$$\forall n > n_2 : c_1 c_3 h(n) \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \leq c_2 c_4 h(n)$$

Daraus folgt $f(n) = \Theta(h(n))$

(b) $f(n) + \Theta(g(n)) = \Theta(g(n))$
Lösung:
Gegenbeispiel:
$f(n) = n^2, g(n) = n$

(c) $f(n) = \Theta(f(n/2))$
Lösung:
Gegenbeispiel:
$f(n) = 2^n$

---

3. (5 + 5 + 5 Punkte)

(a) $T(n) = 3T(n/2) + n$
Lösung:
Anwendung des Mastertheorems

$$a = 3, b = 2, f(n) = n, f(n) = O(n^{\log_b a - \epsilon})$$

Nach dem ersten Fall des Mastertheorems $T(n) = \Theta(n^{\log_b a})$

(b) $T(n) = 2T(n/4) + \sqrt{n}$
Lösung:
Anwendung des Mastertheorems

$$a = 2, b = 4, f(n) = \sqrt{n}, f(n) = \Theta(n^{\cdot 5})$$

Nach dem zweiten Fall des Mastertheorems $T(n) = \Theta(\sqrt{n} \lg n)$

(c) $T(n) = T(n-1) + 1/n$
Das Aufstellen der Summenformel ergibt 2 Punkte.
Iterationsmethode:

$$\begin{aligned} T(n) &= T(n-1) + \frac{1}{n} \\ &= T(n-2) + \frac{1}{n-1} + \frac{1}{n} \\ &\vdots \\ &= T(1) + \sum_{i=2}^{n} \frac{1}{i} \\ &= \Theta(1) + (\ln n + O(1) + 1) \\ &= \Theta(\lg n) \end{aligned}$$

---

4. (5 Punkte)

Jeder Knoten x wird um folgende Attribute angereichert:

- SIZE[x]: Anzahl der Knoten im Teilbaum, dessen Wurzel durch den Knoten x gebildet wird.
- SUM[x]: Summe der Schlüsselwert im Teilbaum, dessen Wurzel von x gebildet wird.

Offensichtlich lassen sich diese Werte für einen Knoten leicht berechnen:

- SIZE[x]: SIZE[LEFT[x]] + SIZE[RIGHT[x]]
- SUM[x]: SUM[LEFT[x]] + SUM[RIGHT[x]]

Nach dem Anreicherungstheorem für Rot-Schwarz-Bäume können die Attribute beim Einfügen und Löschen in $O(\lg n)$ aktualisiert werden.

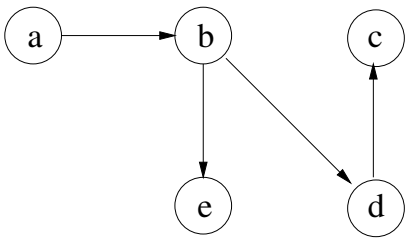
Der gesuchte Mittelwert eines Rot-Schwarz-Baums mit der Wurzel r läßt sich wie folgt berechnen.

SUM[r]/SIZE[r]

---

5. (4 + 2 Punkte)

(a) Ein möglicher Breitensuchbaum ist:



(b) Eine Topologische Sortierungen ist (es gibt nur diese).
f,a,b,e,d,c

---

6. (5 Punkte)

	Tiefensuche	Folge von Breitensuchen
1,2,3,4,5,6,7	☐	x
4,5,6,7,2,3,1	x	☐
7,6,5,4,3,2,1	x	☐
2,4,5,1,3,6,7	x	☐
4,5,2,4,5,6,7	☐	x

---

7. (5 Punkte)

```
DFS-BinTree(r)
  if r ≠ NIL then
    DFS-BinTree(left[r]);
    DFS-BinTree(right[r]);
    print r;
  endif
```

---

8. (10 Punkte)

Anwendung von *Radix-Sort*. Die zu sortierenden Zahlen werden als zweistellige n-äre Zahlen aufgefaßt. Zunächst von allen Werten 1 subtrahieren. Das Ergebnis sind Werte aus $0, \ldots, n^2 - 1$. Die Elemente aus $a \in A$ zunächst entsprechend dem Wert der Funktion $a \operatorname{div} n$ sortieren. Anschließend die Werte $a \in A$ entsprechend dem Wert $a \bmod n$ sortieren. Der Aufwand ergibt sich wie folgt. Die Anzahl der Stellen d ist gleich 2, pro Stelle existieren maximal n verschiedene Werte. Also ist der Gesamtaufwand:

$$\Theta(2n + 2n) = \Theta(n)$$

---

9. (10 Punkte)

   Zunächst wird ein Array mit $W|V|$ Einträgen angelegt. Hierbei ist $\{0, \ldots, W|V|\}$ der Wertebereich der Pfadlängen. Jedes Array-Element ist ein Zeiger auf eine doppelt verkettete Liste. Ein Knoten wird entsprechend seinem d-Wert in eine Liste eingefügt. Der Array wird mit leeren Listen initialisiert. Bis auf die erste Liste, in diese wird der Startknoten eingefügt.

   Bei der Suche nach dem Minimum wird ausgehend von dem letzten Minimum das Array durchsucht. Dies funktioniert deshalb, da durch Relaxation nie ein d-Wert gebildet werden kann der kleiner als der zuletzt selektierte d-Wert ist. Die Kosten für die Extraktion der Knoten aus dem Array betragen somit $O(WV)$.

   Die Decrease-Key-Operation kann in $O(1)$ durchgeführt werden. Der Gesamtaufwand beträgt somit $O(WV + E)$.