
Name (Blockschrift) Matrikelnummer

Hinweise:

Bearbeitungszeit: 180 Minuten

Hilfsmittel: Ein zweiseitig selbstgeschriebenes Hilfsblatt (DinA4), bitte deutlich mit Namen und Matrikelnummer beschriften und markieren.
Ein nichtprogrammierbarer Taschenrechner.

- Bearbeiten Sie die Aufgaben direkt und nur auf den Aufgabenblättern oder Leerbögen
- Lösungen ohne erkennbaren und leserlichen Lösungsweg sind wertlos.
- Klausur zusammenlassen!!!
- Deckblatt mit Namen und Matrikelnummer beschriften und unterschreiben!
- Die letzte Seite der Klausur beinhaltet eine STL-Hilfe.
- Das selbstverfasste Hilfsblatt ist mit der Klausur abzugeben!

Hiermit bestätige ich, dass ich die Hinweise zur Kenntnis genommen habe:

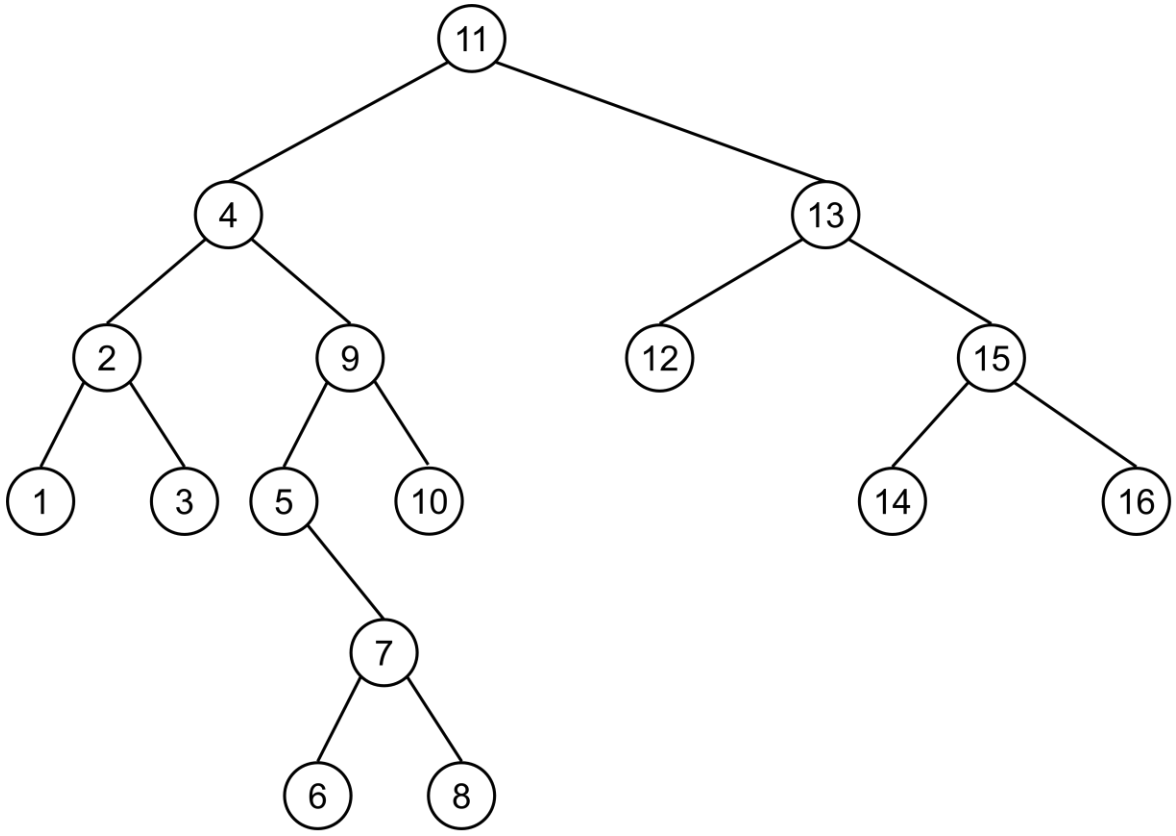
(Unterschrift)

Aufgabe	Max. Pkt.	Punkte	
1	**20**		
2	**20**		
3	**20**		
4	**9**		
5	**19**		
6	**11**		
7	**9**		
8	**12**		
Gesamt	**120**		**Note:**

Aufgabe 1 (20 Punkte)

(a) Gegeben sei eine Prioritätswarteschlange. Begründen oder widerlegen Sie:
Können beide Funktionen zum Einfügen und zur Suche nach der höchsten Priorität in konstanter Zeit implementiert werden? nein (Sonst kann man in O(n) Sortieren)

(b) Gegeben sei der folgende binäre Suchbaum:



Wie sieht die die Traversierungsreihenfolge der Elemente aus, wenn der Baum mit den folgenden Traversierungsalgorithmen durchlaufen wird?

WLR
- Preorder: 11 4 2 1 3 9 5 7 6 8 10 13 12 15 14 16

LWR
- Inorder: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

LRW
- Postorder: 1 3 2 6 8 7 5 10 9 4 12 14 16 15 13 11

- Levelorder 11 4 13 2 9 12 15 1 3 5 10 14 16 7 6 8

(c) Sortieren Sie die Komplexitätsklassen nach ansteigender Komplexität:
$O(3^n)$, $O(n)$, $O(\log n)$, $O(n^4)$, $O(n \log n)$, $O(n^2)$, $O(1)$ und ordnen Sie der Komplexität die richtigen Begriffe zu: linear, quadratisch, exponentiell, polynomial, logarithmisch, überlinear, konstant

$O(1)$ → $O(\log n)$ → $O(n)$ → $O(n \log n)$ → $O(n^2)$ → $O(n^4)$ → $O(3^n)$

Konstant, logarithm., linear, überlin., quadra., Polynomial, expon.

(d) Mit welchem Aufwand kann in einer doppelt verketteten Liste mit n Elementen ein Element eingefügt werden? Die Liste verfügt über einen Kopf- und Endezeiger (head und tail). Kreuzen Sie die richtigen Antworten an.

- [x] O(1) wenn es am Kopf eingefügt wird.
- [ ] O(n) beim Einfügen am Ende.
- [ ] O(log n) beim Einfügen in der Mitte.
- [x] O(n/2) beim Einfügen in der Mitte.
- [ ] O(n) beim Einfügen am Anfang.
- [x] O(1) beim Einfügen am Ende.

(e) Welcher Traversierungsalgorithmus durch einen binären Suchbaum liefert eine sortierte Reihenfolge der Elemente?

In Order LWR

(f) Welche Datenstruktur findet ein **beliebiges** Element in O(1)?

Hash table

(g) Beim HeapSort wird ein MaxHeap verwendet.

Welches Element befindet sich in der Baumdarstellung in der Wurzel?

- [ ] Minimum
- [ ] Median
- [x] Maximum

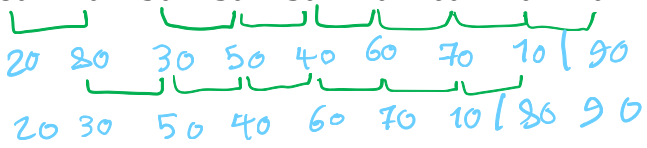
In welcher Reihenfolge ist das Array sortiert, wenn der HeapSort Algorithmus endet?

- [x] Aufsteigend
- [ ] Absteigend

(h) Führen Sie für die Folge 80, 20, 90, 30, 50, 40, 60, 70, 10 zwei Sortierdurchläufe mit Bubble Sort und jeweils einen Sortierdurchlauf mit dem Shell Sort mit Abstand 3 und Quick Sort mit Pivot-Element 50 durch. Markieren Sie die Vertauschungen durch Pfeile.

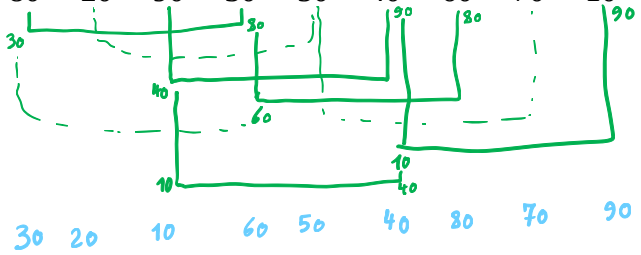
2 Sortier-Durchläufe mit Bubble Sort:

80 20 90 30 50 40 60 70 10



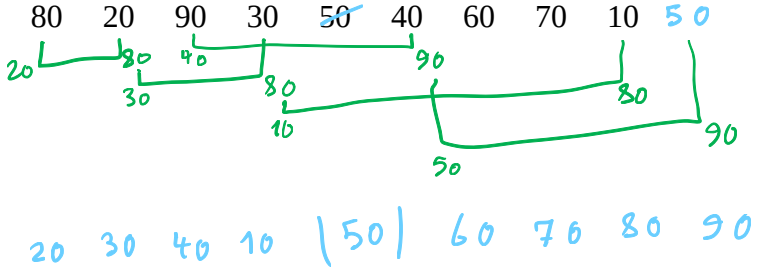
1 Sortier-Durchlauf mit Shell-Sort und Abstand = 3:

80 20 90 30 50 40 60 70 10



1 Sortier-Durchlauf mit Quick-Sort und Pivotelement 50:

80 20 90 30 ~~50~~ 40 60 70 10 50



Wie lauten die nächsten Quick-Sort-Aufrufe?

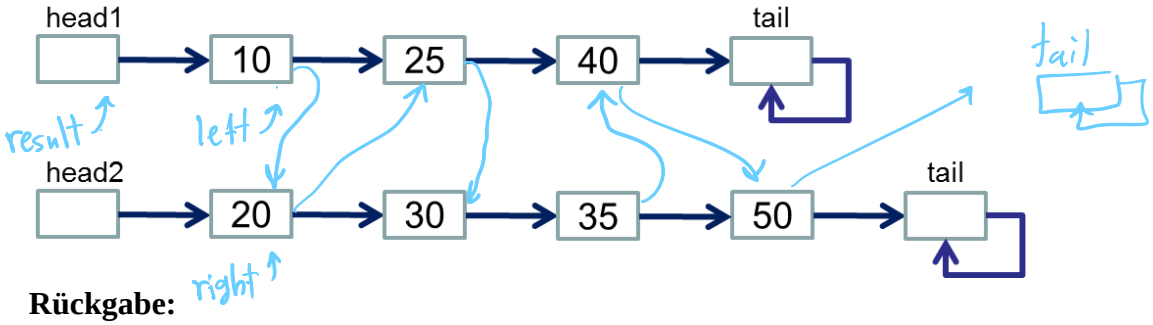
QuickSort(A, 0, 3);

QuickSort(A, 5, 8);

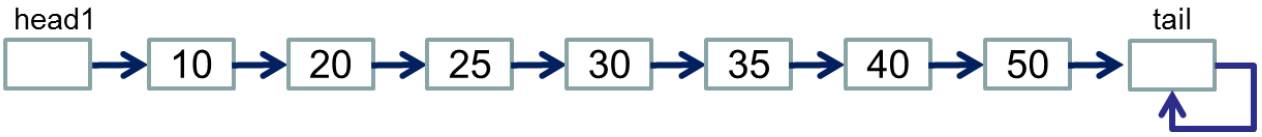
Aufgabe 2 (20 Punkte)
Mischen von 2 sortierten einfach verketteten Listen

Gegeben sind zwei einfach verkettete Listen, deren Knotenelemente aufsteigend sortiert sind. Implementieren Sie eine Funktion, die die zweite sortierte Liste in die erste sortierte Liste einsortiert. Dabei sollen die Knotenelemente der zweiten Liste per Referenz an die richtige Position der ersten Liste positioniert werden.

Beispiel:



Rückgabe:




```
void mergeLists(node* &head1, node* head2, node* tail)
{
// Rückgabewert: erfolgreiches oder erfolgloses Einfügen
// head1: Referenz auf das 1.te Liste, head1-next zeigt auf das 1.
// Listenelement (analog für head2)
// tail: ist eine Referenz auf den Nullpointer
```

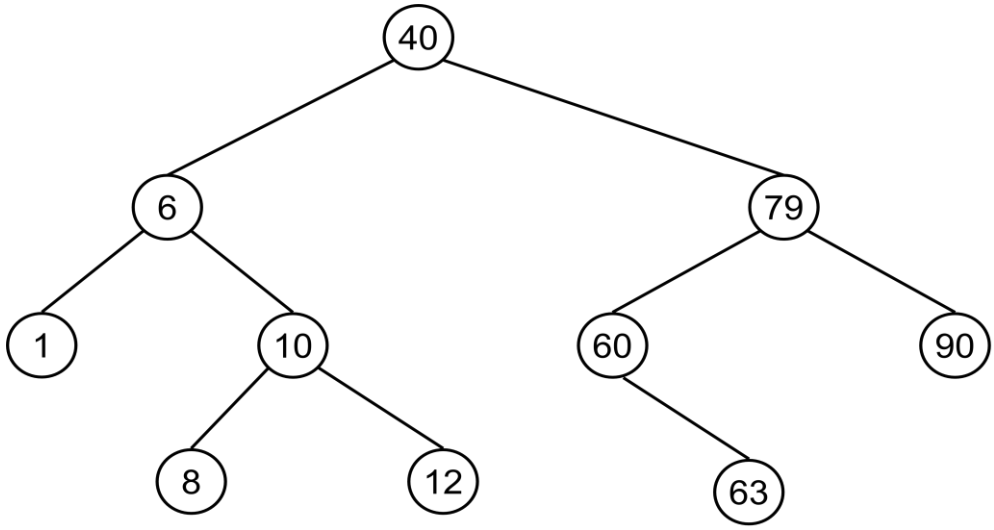
```
node *left = head1 -> next ; *right = head2 -> next , * result = head1;
while ( left != tail  && right != tail)
 {   if(left ->item  < right ->item) {
        result -> next = left;
        left = left -> next;
     } else {
        result -> next = right;
        right = right -> next;

     }
     result = result -> next;
}
while (left != tail) {
     result -> next = left ;
     left = left -> next;
}     result = result -> next;
```

Weiter auf Seite 6

```
while (right != tail)
    result->next = right;
    right = right->next;
    result = result->next;
}
result->next = tail;
}
```

Aufgabe 3 (8 + 12 = 20 Punkte)
AVL-Baum

Gegeben sei der folgende AVL-Baum:



(a) Gesucht ist eine mögliche Eingabereihenfolge der Knoten, so dass keine Rotation notwendig wurde. Geben Sie diese Reihenfolge an.

{40, 6, 79, 1, 10, 60, 90, 8, 12, 63}

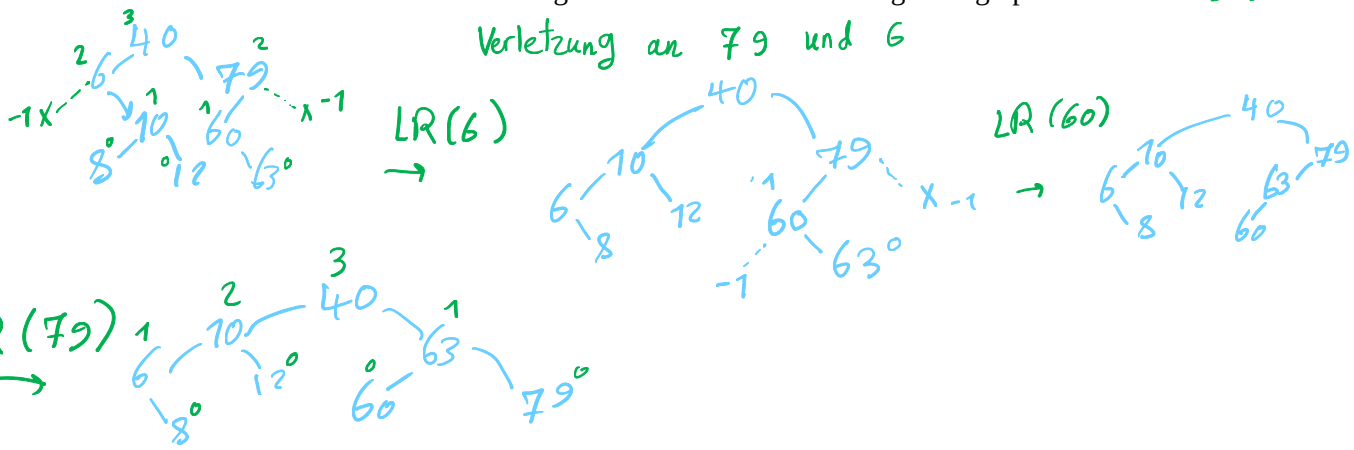
(b) Geben Sie eine mögliche Eingabereihenfolge der Knoten an, so dass genau eine Rechts- und eine Linksrotation notwendig ist und am Ende der oben abgebildete Baum entsteht.

{6, 40, 79, 1, 12, 10, 8, 60, 90, 63}
LR, 1 wichtig!, RR

(c) Löschen Sie nun nacheinander die Knoten 1 und 90.

- An welchen Knoten ist das Balance-Kriterium verletzt?
- Welche Rotationen sind nötig, um das AVL-Kriterium wieder herzustellen? Führen Sie diese durch und stellen Sie die Zwischenergebnisse wie auch das Endergebnis graphisch dar.

LR + LR-R



(d) Die Datenstruktur für einen Knoten im AVL-Baum sei:

```
class node {
   int item;
   int height;
   node* left;
   node* right;
};
```

Implementieren Sie eine Funktion, die in einem AVL-Baum für alle Knoten BottomUp die Höhe des Knotens setzt und alle Knoten ausgibt, die das Balance-Kriterium verletzen. Der Baum soll dabei nur **genau 1x traversiert** werden. (Tip: verwenden Sie zur Traversierung des Baumes einen rekursiven Postorder-Traversierungsalgorithmus, der den Baum BottomUp durchläuft). (10 Punkte)

Tip: Implementieren Sie zunächst die Starter-Funktion für den rekursiven Algorithmus und anschliessend die rekursive Methode.

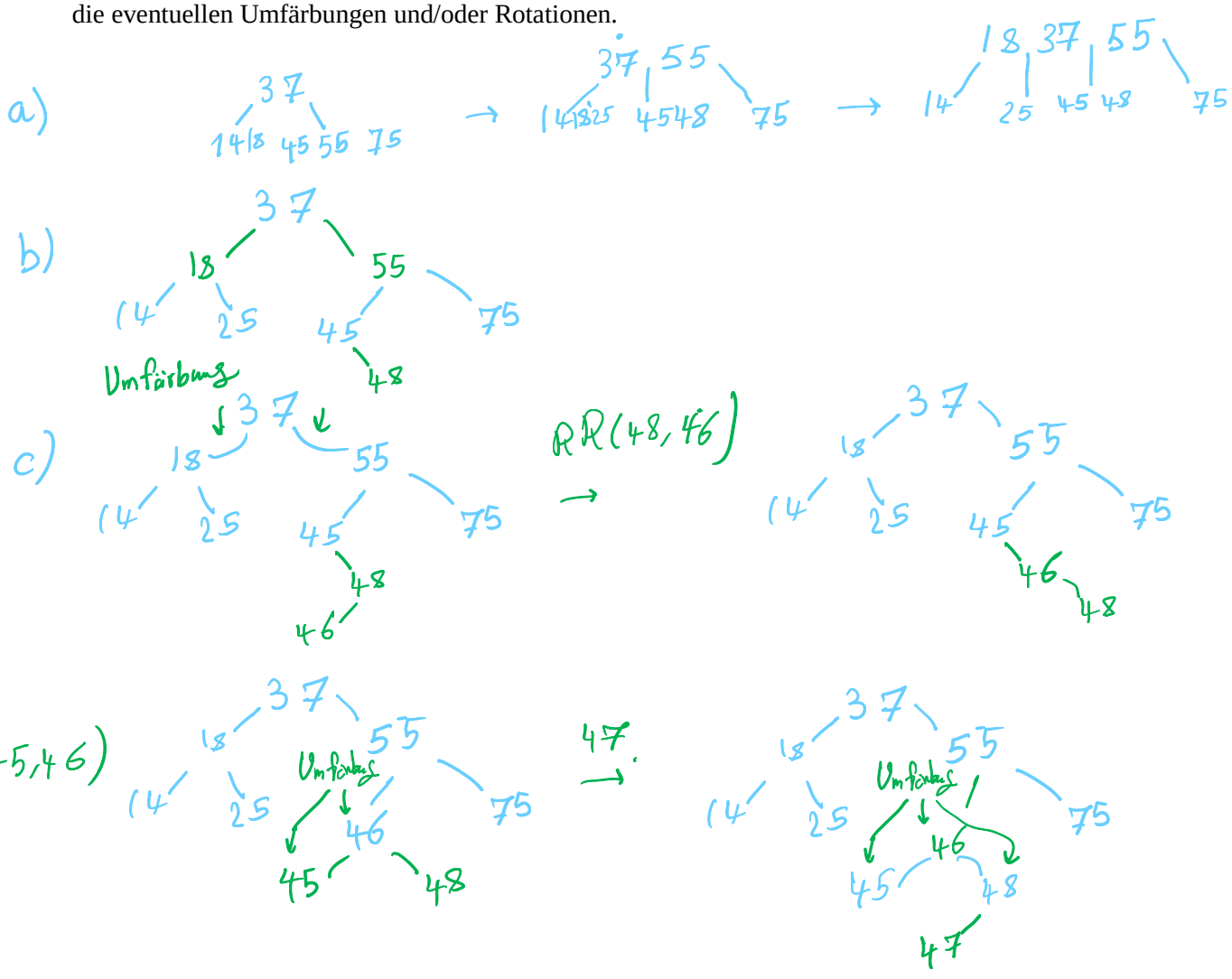
```
Void starter() {
        setHeight(anker);
}

int setHeight(node * k) {
   if(!k)
      return -1;
   int l = setHeight(k->left);
   int r = setHeight(k->right);
   k->height = (l<r) ? 1+r : 1+l;
   if(l-r >= 2 || l-r <= -2)
         cout << k->item << " :Balance(" << l-r << ")"
              << endl;

   return k->height;
}
```

Aufgabe 4 (9 Punkte)
234-Baum

(a) Fügen Sie die Folge 37, 14, 55, 45, 75, 18, 48, 25, 17 TopDown in einen anfangs leeren 234-Baum ein.

(b) Wie sieht der entsprechende Rot-Schwarz-Baum aus? Markieren Sie die roten Knoten bitte mit der Farbe grün (wenn möglich).

(c) Fügen Sie nun in den Rot-Schwarz-Baum die Elemente 46 und 47 TopDown ein. Beschreiben Sie die eventuellen Umfärbungen und/oder Rotationen.



Aufgabe 5 (9 + 10 = 19 Punkte)
Hashing

(a) Die Zahlen 64, 12, 34, 32, 74, 2 sollen in eine anfangs leere Hashtabelle der Größe M=8 nacheinander eingefügt werden. Zeigen Sie, mit welchem Index die Hashfunktionen kollisionsfrei sind und geben den entsprechenden Index an. Geben Sie für jeweiligen Verfahren die konkrete Hashfunktion an.

- Lineares Sondieren: $h_i(x)$ = $(x+i)\ \%\ 8$

i	0	1	2	3	4	5	6	7
ht[i]	64	32	34	74	12	2	-1	-1

- Quadratisches Sondieren: $h_i(x)$ = $(x+i^2)\ \%\ 8$

i	0	1	2	3	4	5	6	7
ht[i]	64	32	34	74	12	-1	2	-1

64 12 34 32 74 2

5 74 7-4 3 3 4 7 32 28 4

- Doppeltes Hashing: $h_i(x)$ = $(x+i(7-x\ \%\ 7))\ \%\ 8$

i	0	1	2	3	4	5	6	7
ht[i]	64	-1	34	32	12	74	-1	2

- Welchen Wert hat der Belegungsfaktor nach dem Einfügen der Folge? $\frac{6}{8} = \frac{3}{4} = 0.75$

(b) Implementieren Sie eine Funktion, die ein Element **`newitem`** mit dem Hashing einfügt. Bei Kollisionen soll das quadratische Sondieren verwendet werden. Wenn der Belegungsfaktor beim Einfügen >= **`factor`** ist, soll die Funktion ein false zurückgeben, ansonsten true. (Bitte programmieren Sie nicht das Rehashing selbst in der Funktion insertHashtable!). Falls die Anzahl der Kollisionen die Hashtabellengröße übersteigt, soll die Funktion abgebrochen werden und ebenfalls ein false zurückgegeben werden.

Wie groß ist der Aufwand in O-Notation zu ihrer Funktion (begründen Sie)? O(10) = O(1)
Die folgenden Anweisungen werden höchstens 10 mal durchgeführt.

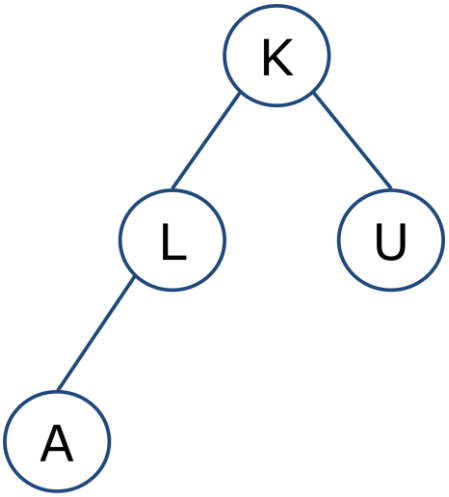
```
bool insertHashtable(vector<int> &a, int newItem, double factor, int
counter)
{
// a ist die Hashtabelle
// newItem ist das neu einzufügende Element
// factor ist der Wert des maximalen Belegungsfaktors
// counter ist die aktuelle Anzahl der belegten Elemente in der Hashtabelle
    if ( counter * 1.0 / a.size() >= factor)
        return false;

    int i=0;
    int hash;     int try = 10;

    while (try--)
    {   hash = (newitem + i*i) % a.size();

        if (a[hash] == -1) {

            a[hash] = newitem;
              break;
        } else
            i++;

    }

    if(try == -1)
      return false;

    return true;

}
```

Aufgabe 6 (11 Punkte)

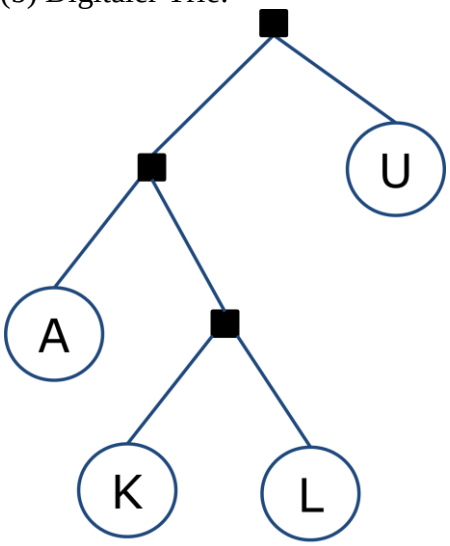
Gegeben sind digitale Bäume mit M=2, bei denen die Buchstaben KLAUS eingefügt wurden nach dem binären Code aus der gegebenen Tabelle. Fügen Sie nun die Buchstaben N, R und V ein und stellen die Ergebnisse der binären digitalen Bäume grafisch dar.

Buchstabe	Binärcode
K	01011
L	01100
A	00001
U	10101
N	01110
R	10001
V	10110

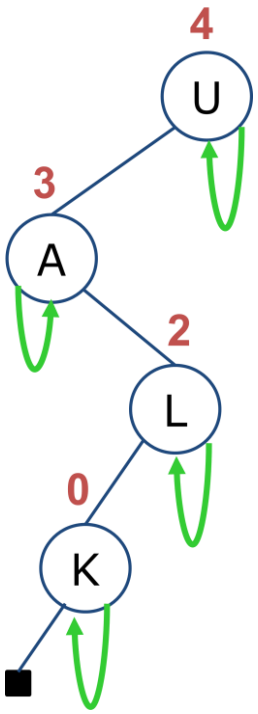
(a) Binärer Suchbaum:



(b) Digitaler Trie:



(c) Patricia Baum:



Aufgabe 7 (9 Punkte)

Sortieren Sie die folgende Zahlenfolge in **absteigender** Reihenfolge

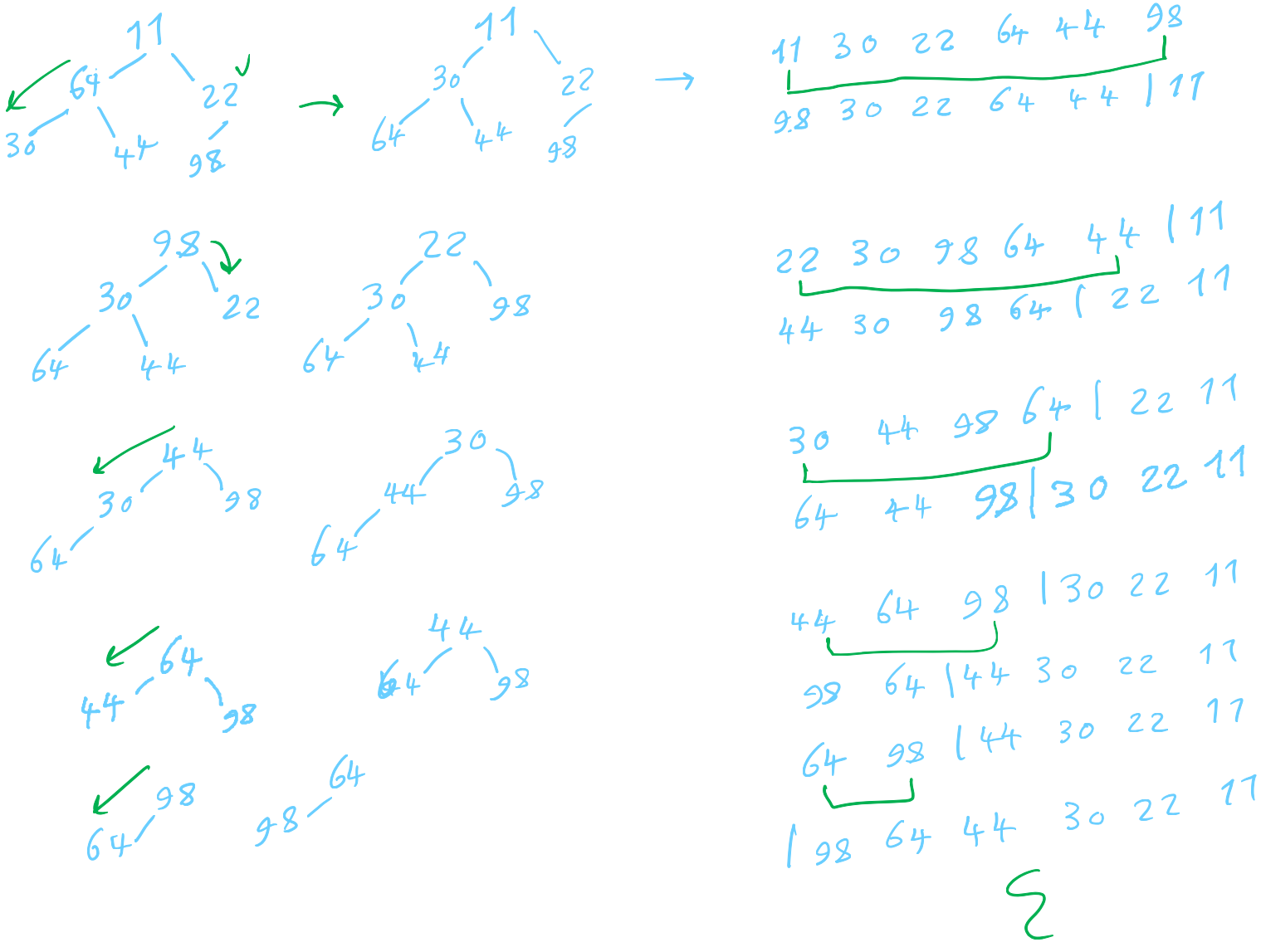
11 64 22 30 44 98

unter Verwendung von Heap Sort.

(a) Verwenden Sie dazu einen MinHeap oder einen MaxHeap?

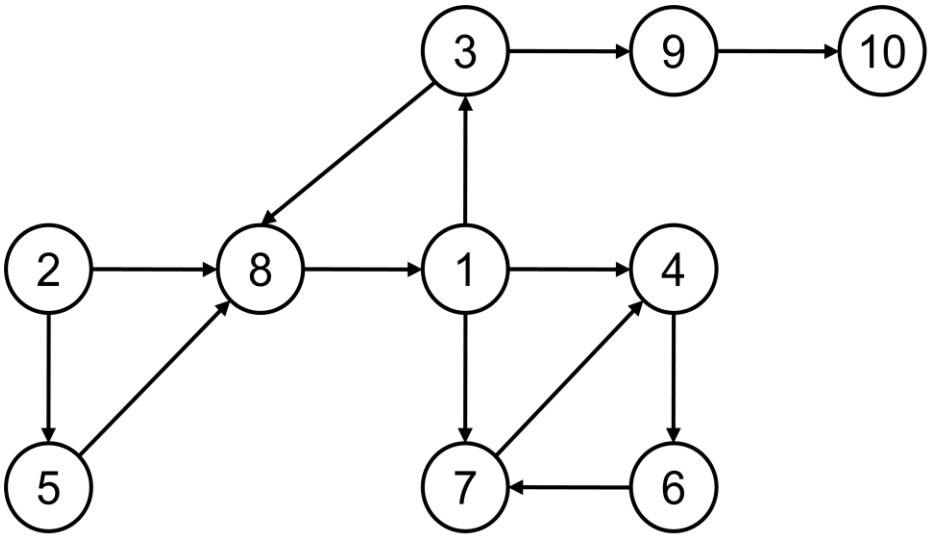
(b) Zeigen Sie den Aufbau des Heaps in Baumnotation und das Ergebnis des Heaps in Baum- und Arraynotation.

(c) Führen Sie die Sortierung mit dem Heap Sort Verfahren durch und zeigen Sie jeweils die notwendigen Zwischenschritte **in der Baum- und Arraydarstellung nach** jedem entfernten Element.



Aufgabe 8 (12 Punkte)
Graphen

Gegeben ist der folgende Graph G mit V={1,2,..,10} Knoten:



(a) Um welche Art eines Graphen handelt es sich hier?

ungewichtet und grichtet

(b) Welche Darstellungsformen von Graphen kennen Sie?

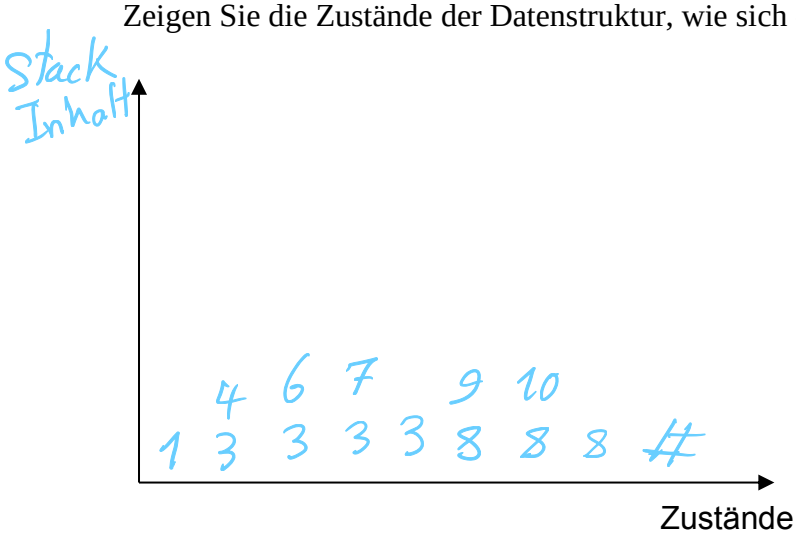
Siehe - Vorlesung -

(c) Geben Sie die Adjazenzliste zu dem Graphen an (bitte numerisch sortiert angeben).

1 → 3 4
2 → 5 8
3 → 8 9
4 → 6
5 → 8
6 → 7
7 → 4
8 → 1
9 → 10
10 →

(d) Führen Sie für den Graphen G eine **iterative Tiefensuche** beginnend vom Knoten 1 aus durch (Bearbeiten Sie die Nachfolgerknoten in der Reihenfolge der Adjazenzliste). Welche Datenstruktur verwenden Sie dazu (beschriften Sie die y-Achse entsprechend)? Zeigen Sie die Zustände der Datenstruktur, wie sich diese bei der Traversierung verändert.

Traversierungsreihenfolge:
1 4 6 7 3 9 10 8



STL-Hilfe:

Container, einige wichtige Funktionen und deren Verwendung:

vector<type> Feld

```
vector<int> numbers;
numbers.push_back(3);        // Einfügen am Ende des Feldes
numbers.pop_back();          // entfernt das letzte Element
int last = numbers.back();   // liefert das letzte Element
int first = numbers.front(); // liefert das erste Element
int item = numbers[i];       // liefert das i-te Element
int size = numbers.size();   // liefert die Größe des Feldes
bool empty = numbers.empty(); // true, wenn das Feld leer ist,
                              // andernfalls false
```

list<type> doppelt verkettete Liste (s.Funktionen wie bei vector)

```
list<int> numbers;
numbers.push_back(5); // fügt 5 ans Ende der Liste
numbers.remove(5);    // löscht das Element 5
```

queue<type> Queue

```
queue<int> q; // int – Warteschlange allokieren
q.push(5);    // Element 5 in die Warteschlange
bool leer = q.empty();   // Abfrage, ob Warteschlange leer ist
int element = q.front(); // liefert erstes Element
q.pop() ;     // löscht erstes Element
```

stack<type> Stapel

```
stack<int> nodes; // int - Stapel allokieren
bool ok = nodes.empty();   // überprüfe, ob Stapel leer ist
int element = nodes.top(); // hole Element vom Stapel
nodes.pop();   // entfernt Element vom Stapel
nodes.push(5); // 5 wird auf den Stapel gelegt
int i = (int) nodes.size(); // Anzahl Elemente im Stapel
```

map<keytype, valuetype> (Schlüssel, Referenz)-Container

```
map<int, vector<int> > adjlist;    // Adjazenzliste als Map
int size_adjlist = adjlist.size(); // Anzahl Knoten in der Adjazenzliste
vector<int> liste = adjlist[1]; // liefert den Vektor zum Schlüssel 1
adjlist[1].push_back(5); // Einfügen von Schlüssel 5 zur Liste von Schlüssel 1
```