Teilklausur II in Informatik – TEL08GR1 + TELK08 (TEL08GR2) – Lewandowski

Es sind insgesamt 84 Punkte zu erreichen (80 Punkte werden als 100% gewertet)
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Binäre Suche, Interpolationssuche (8 Punkte) – durchführen und Wissensfrage
Aufgabe 2: Löschen in binären Suchbäumen (8 Punkte) – beschreiben und durchführen
Aufgabe 3: Einfügen in AVL-Bäumen (12 Punkte) – durchführen
Aufgabe 4: Insertionsort (11 Punkte) – durchführen und Fehlstand ermitteln
Aufgabe 5: Quicksort (8 Punkte) – Pivot-Element analysieren
Aufgabe 6: Dijkstra-Algorithmus (9 Punkte) – durchführen
Aufgabe 7: Ein Baumalgorithmus (28 Punkte) – Eigenschaften, C/C++, O-Notation

Bearbeiten Sie die Aufgaben bitte direkt auf der Aufgabenstellung! Bitte kennzeichnen Sie die Aufgabenstellung und jedes zusätzliche Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer!

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 8 Punkte

(3+3+2 = 8 Punkte) **Binäre Suche, Interpolationssuche:** Gegeben ist ein Array mit den Werten (5,6,8,10,25,28,29). Es soll der Wert 25 gesucht werden.

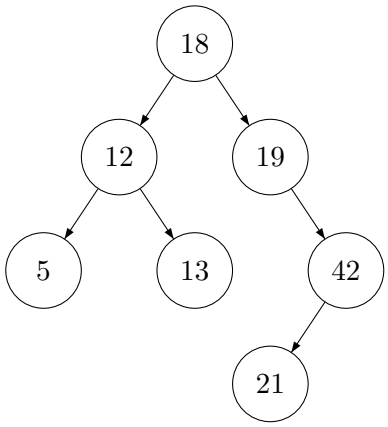
- (3 Punkte) Geben Sie an, mit welchen Elementen bei **binärer Suche** die 25 verglichen wird, bis sie gefunden wird.

- (3 Punkte) Geben Sie an, mit welchen Elementen bei **Interpolationssuche** die 25 verglichen wird, bis sie gefunden wird.

- (2 Punkte) Welche Eigenschaft muss ein Array mit Zahlen haben, damit man in diesem binäre Suche und Interpolationssuche anwenden kann? (Kurze Antwort mit einem Satz Begründung reicht)

Aufgabe 2 — 8 Punkte

(3+5 = 8 Punkte) **Löschen in binären Suchbäumen:** Gegeben ist folgender binäre Suchbaum:



- (3 Punkte) Geben Sie an, welcher Suchbaum entsteht, wenn die 18 gelöscht wird.
- (5 Punkte) Beschreiben Sie in 3 Sätzen, welche Fälle beim Löschen in binären Suchbäumen entstehen können und wie diese behandelt werden.

Aufgabe 3 — 12 Punkte

(12 Punkte) Fügen Sie in einen anfangs leeren **AVL-Baum** folgende Zahlen in dieser Reihenfolge ein: 9, 7, 2, 6, 3, 1, 4. Wenn eine Rotation notwendig ist, geben Sie den AVL-Baum jeweils vor und nach dieser Rotation an. Geben Sie in jedem Fall den AVL-Baum an, nachdem alle Elemente eingefügt wurden.

Aufgabe 4 — 11 Punkte

(9+2 Punkte) **Insertionsort:** Gegeben ist ein Array mit den Zahlen (9,7,2,6,3,1,4).

(9 Punkte) Führen Sie Insertionsort durch:

- Geben Sie den Inhalt des Arrays nach jedem kompletten Durchlauf der äußeren Schleife an.
- Tragen Sie jeweils die Anzahl der in diesem Durchlauf durchgeführten Vertauschungen ein.

Hinweis: die Anzahl der Zeilen in der Tabelle ist größer als die tatsächlich auszufüllende Anzahl der Zeilen.

Durchlauf								Vertauschungen
vor 1.	9	7	2	6	3	1	4	-
nach 1.								
nach 2.								
nach 3.								
nach 4.								
nach 5.								
nach 6.								
nach 7.								
nach 8.								
nach 9.								
nach 10.								
nach 11.								
nach 12.								
nach 13.								
nach 14.								
nach 15.								

(2 Punkte) Wie groß ist der Fehlstand (Inversionszahl) des obigen Feldes (9,7,2,6,3,1,4)?

Aufgabe 5 — 8 Punkte

(8 Punkte) Bei **Quicksort** wird als Pivot-Element das Element in der Mitte gewählt (Index `(l+r)/2`, hier Index 10). Folgendes Feld sei das Ergebnis der ersten Iteration bei Quicksort (also Wandern der Zeiger mit Vertauschungen bis die beiden Zeiger übereinander hinweg gelaufen sind):

10 22 15 14 16 6 25 9 31 56 91 75 99 88 94 55 79 98 73 77 59

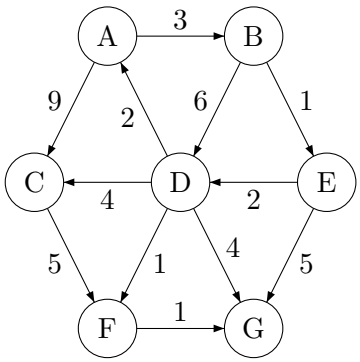
Geben Sie eine Möglichkeit an, welches Pivot-Element gewählt wurde, und begründen Sie Ihre Antwort kurz.

Zusatzaufgabe: Ist das von Ihnen gefundene Element das einzig mögliche?

Aufgabe 6 — 9 Punkte

(9 Punkte) **Kürzeste-Wege-Berechnung mit dem Algorithmus von Dijkstra:**

Führen Sie den Dijkstra-Algorithmus auf folgendem Graphen aus:
Zu Beginn ist lediglich die Entfernung zum Startknoten A mit d(A)=0 bekannt. Im Dijkstra-Algorithmus wird in jeder Iteration ein Knoten aus der Randmenge R entfernt. Geben Sie in jeder Zeile den Zustand der Menge R jeweils vor diesem Entfernen eines Knotens an, den durch delete_min gewählten Knoten und die berechneten Entfernungen d() am Ende des Durchlaufs für alle Knoten.



	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	$\emptyset$	„Initialisierung“	0	∞	∞	∞	∞	∞	∞
1.	$\{A\}$								
2.									
3.									
4.									
5.									
6.									
7.									
8.									

Aufgabe 7 — 28 Punkte

(3+3+2+15+2+3 = 28 Punkte) **Ein Baumalgorithmus:** In dieser Aufgabe werden einige grundlegende Eigenschaften von binären Suchbäumen untersucht und in ein Programm umgesetzt. (Hinweis: die Fragen weisen einen möglichen Weg, die C/C++-Funktion zu realisieren – sie können die Programmieraufgabe aber auch unabhängig von Ihren vorherigen Antworten lösen.)

Es reichen jeweils kurze Antworten mit je einem Satz Begründung.

- (3 Punkte) Wo befindet sich in einem binären Suchbaum das größte Element? Warum kann der Knoten mit dem größten Wert in einem binären Suchbaum kein rechtes Kind haben?
- (3 Punkte) Wo liegt das zweitgrößte Element, wenn das größte Element ein linkes Kind hat? Wo liegt das zweitgrößte Element, wenn das größte Element ein Blatt ist?
- (2 Punkte) Geben Sie eine C/C++-Datenstruktur für einen binären Suchbaum an. Der Datentyp für eine Variable, die auf die Wurzel eines binären Suchbaums zeigt, soll dabei `binSBaumPtr` heißen.
- (15 Punkte) Schreiben Sie eine C/C++-Funktion `void zweites(const binSBaumPtr root)`, die als Parameter den Zeiger auf die Wurzel eines binären Suchbaums bekommt. Die geforderte Ausgabe ist:
  - Ist der Baum leer, so soll die Meldung „`Baum ist leer.`“ ausgegeben werden.
  - Hat der Baum nur einen Knoten, so soll die Meldung „`Baum hat nur einen Knoten.`“ ausgegeben werden.
  - Hat der Baum mindestens zwei Knoten, so soll die Meldung „`Das zweitgrößte Element im Baum ist die` x“ ausgegeben werden, wobei x durch den entsprechenden Wert zu ersetzen ist.

  Zu dem Baum aus Aufgabe 2 soll die Ausgabe also „`Das zweitgrößte Element im Baum ist die 21`“ lauten.
- (2 Punkte) Geben Sie den Aufwand Ihres Algorithmus (für den schlechtesten Fall) in O-Notation an.
- (3 Punkte) Wieviel Schritte benötigt Ihr Algorithmus im besten Fall? Skizzieren Sie einen Baum mit 10 Knoten, der mit Ihrem Algorithmus möglichst schnell bearbeitet wird.

Beachten Sie: die Parameter sind an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe des Baumes kümmern.

Teilklausur II in Informatik – Lösungshinweise

Aufgabe 1 — Binäre Suche, Interpolationssuche

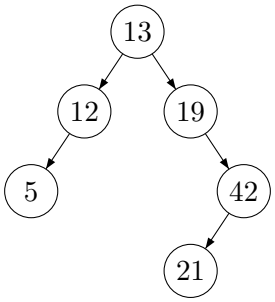
Gegeben war ein Array mit den Werten (5,6,8,10,25,28,29). Gesucht: der Wert $s = 25$.

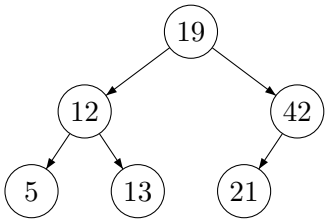
- Bei **binärer Suche** wird jeweils in der Mitte des verbleibenden Teilfeldes gesucht ⇒ Vergleiche mit 10, 28, 25 ⇒ gefunden
- Bei **Interpolationssuche** wird jeweils am Index $m = l + (s - a[l]) \cdot (r - l)/(a[r] - a[l])$ gesucht ⇒ Vergleiche mit $a[0 + (25 - 5) \cdot (6 - 0)/(29 - 5)] = a[5] = 28$ und $a[0 + (25 - 5) \cdot (4 - 0)/(25 - 5) = a[4] = 25$ ⇒ gefunden
- Das Feld muss sortiert sein, sonst kann nicht garantiert werden, dass ein kleinerer gesuchter Wert links vom getesteten Index und ein größerer Wert rechts davon liegt.

Aufgabe 2 — Löschen in binären Suchbäumen

Beim Löschen eines Blattes wird dieses einfach entfernt. Hat der zu löschende Knoten nur ein Kind, wird der Unterbaum durch den Unterbaum des Kindes ersetzt. Hat der zu löschende Knoten zwei Kinder, so wird er durch den Inordernachfolger (oder Inordervorgänger ersetzt) und die dort entstehende Lücke (der Knoten ist ein Blatt oder hat nur ein Kind) entsprechend behandelt.

Links der entstehende Suchbaum bei Ersetzen der 18 mit deren Inordervorgänger, rechts bei Verwendung des Inordernachfolgers.



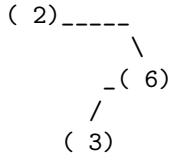


Aufgabe 3 — Einfügen in AVL-Bäume

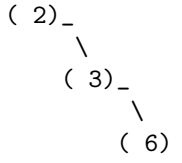
```
Werte: 9, 7, 2                          _( 9)
Balance an Knoten 9 verletzt           /
Unterbaum vor der Rotation:          _( 7)
                                    /
                                 ( 2)

Rechtsrotation                         _( 7)_
Baum nach der Rotation:               /      \
                                   ( 2)    ( 9)
```

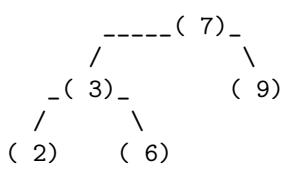
Nächste Werte: 6, 3
Balance an Knoten 2 verletzt
Unterbaum vor der Rotation:



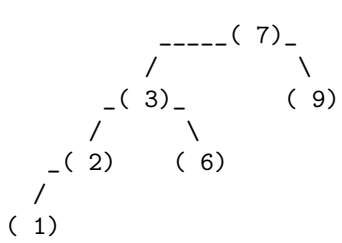
Rechtslinksrotation
Unterbaum nach der Rechtsrotation:



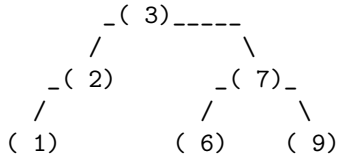
Baum nach der Linksrotation:



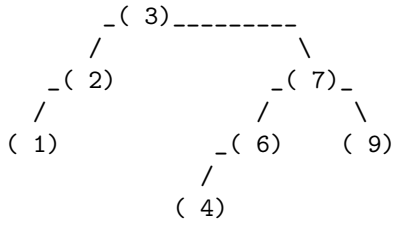
Nächster Wert: 1
Balance an Knoten 7 verletzt
Unterbaum vor der Rotation:



Rechtsrotation
Baum nach der Rotation:



Nächster Wert: 4

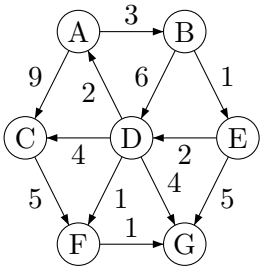


Aufgabe 4 — Insertionsort

Durchlauf								Vertauschungen
vor 1.	9	7	2	6	3	1	4	-
nach 1.	7	9	2	6	3	1	4	1
nach 2.	2	7	9	6	3	1	4	2
nach 3.	2	6	7	9	3	1	4	2
nach 4.	2	3	6	7	9	1	4	3
nach 5.	1	2	3	6	7	9	4	5
nach 6.	1	2	3	4	6	7	9	3

Der Fehlstand/Inversionszahl kann hier leicht als Summe der Vertauschungen ermittelt werden (zur Erinnerung: jeder Tausch eines benachbarten Pärchens verringert den Fehlstand um 1 – oder man zählt noch mal extra): das Ergebnis ist 16.

Aufgabe 5 — Dijkstra-Algorithmus



	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	$\emptyset$	„Initialisierung“	0	∞	∞	∞	∞	∞	∞
1.	{A}	A	0	3	9	∞	∞	∞	∞
2.	{B,C}	B	0	3	9	9	4	∞	∞
3.	{C,D,E}	E	0	3	9	6	4	∞	9
4.	{C,D,G}	D	0	3	9	6	4	7	9
5.	{C,F,G}	F	0	3	9	6	4	7	8
6.	{C,G}	G	0	3	9	6	4	7	8
7.	{C}	C	0	3	9	6	4	7	8
8.	$\emptyset$	„fertig“	0	3	9	6	4	7	8

Aufgabe 6 — Ein Baumalgorithmus

Die größeren Elemente befinden sich im rechten Unterbaum, das größte somit ganz rechts. Hätte der Knoten ein rechtes Kind, so wäre dessen Wert größer als der größte - ein Widerspruch.

Hat der größte Wert ein linkes Kind, so befinden sich in diesem Unterbaum alle Werte, die größer oder gleich diesem Kind sind und kleiner als der größte Wert. Der zweitgrößte Wert im Baum ist dann der größte in diesem Unterbaum.

Ist das größte Element ein Blatt, so ist dessen Vorgänger das zweitgrößte Element.

Hier sind 3 naheliegende Möglichkeiten aufgeführt – natürlich reichte eine Variante aus.

```
struct binSBaum;
type binSBaum * binSBaumPtr;
struct binSBaum {
  int value;
  binSBaumPtr left,right;
};

// Variante 1 mit Verwendung der Hinweise aus der Aufgabenstellung:

void zweites(const binSBaumPtr root) {
  binSBaumPtr node=root;
  if (node==NULL) { cout << "Baum ist leer." << endl; }
  else {
    binSBaumPtr elter=0;
    while (node->right) {
      elter=node;
      node=node->right;
    } // node ist jetzt größtes
    if (node->left) {
      node=node->left;
      while (node->right) {
        node=node->right;
      }
      cout << "Das zweitgrößte Element im Baum ist die " << node->value << endl;
    } else { // größtes hat keine Kinder
```

Im schlechtesten Fall muss jeder Knoten betrachtet werden (z.B. im Suchbaum, der durch Einfügen einer sortierten Zahlenfolge entstanden ist): Aufwand also $O(n)$ – Randbemerkung: in AVL-Bäumen könnte man sogar $O(\log n)$ garantieren.

```
      if (elter) {
        cout << "Das zweitgrößte Element im Baum ist die " << elter->value << endl;
      } else { // größtes war Wurzel und hat kein linkes Kind, also ...
        cout << "Baum hat nur einen Knoten." << endl;
      }
    }
  }
}

// Variante 2 erfordert 2 globale Variablen und benutzt einen Inorderdurchlauf

int akt, zweites;

void inorder(binSBaumPtr root) {
  if (root) {
    inorder(root->left);
    zweites=akt; akt=root->value;
    inorder(root->right);
  }
}

void zweites_reloaded(const binSBaumPtr root) {
  if (root==NULL) { cout << "Baum ist leer." << endl; }
  else if (root->left==NULL && root->right==NULL) {
    cout << "Baum hat nur einen Knoten." << endl;
  } else {
    inorder(root);
    cout << "Das zweitgrößte Element im Baum ist die "
         << zweites << endl;
  }
}

// Variante 3 analog Variante 2 aber ohne globale Variablen

void inorder(binSBaumPtr root, int & akt, int & zweites) {
  if (root) {
    inorder(root->left,akt,zweites);
    zweites=akt; akt=root->value;
    inorder(root->right,akt,zweites);
  }
}

void zweites_revolutions(const binSBaumPtr root) {
  if (root==NULL) { cout << "Baum ist leer." << endl; }
  else if (root->left==NULL && root->right==NULL) {
    cout << "Baum hat nur einen Knoten." << endl;
  } else {
    int akt,zweites;
    inorder(root,akt,zweites);
    cout << "Das zweitgrößte Element im Baum ist die "
         << zweites << endl;
  }
}
```

Varianten 2 und 3 benötigen immer Linearzeit durch den Inorderdurchlauf, bei Variante 1 könnten die 10 Wurzel sein, die 9 linkes Kind und die Elemente 1 bis 8 im linken Unterbaum der 9. Nach 1 Schritt ist die 10 als größtes Element gefunden, die 9 als linkes Kind hat kein rechtes Kind, wird somit sofort als zweitgrößtes Element ausgegeben (Aufwand im besten Fall also konstant)!

(Ebenfalls als Randbemerkung: bei AVL-Bäumen benötigt man durch die Ausgeglichenheit der Bäume auch im besten Fall logarithmisch viele Schritte.)

Nachtermin – Teilklausur II in Informatik – TELK08 (TEL08GR2) – Lewandowski

Es sind insgesamt 84 Punkte zu erreichen (80 Punkte werden als 100% gewertet)
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Binäre Suche, Interpolationssuche (10 Punkte) – durchführen und Wissensfrage
Aufgabe 2: Löschen in binären Suchbäumen (8 Punkte) – beschreiben und durchführen
Aufgabe 3: Einfügen in AVL-Bäumen (12 Punkte) – durchführen
Aufgabe 4: Insertionsort (11 Punkte) – durchführen und Fehlstand ermitteln
Aufgabe 5: Quicksort (10 Punkte) – durchführen
Aufgabe 6: Dijkstra-Algorithmus (9 Punkte) – durchführen
Aufgabe 7: Ein Baumalgorithmus (24 Punkte) – Eigenschaften, C/C++, O-Notation

Bearbeiten Sie die Aufgaben bitte direkt auf der Aufgabenstellung! Bitte kennzeichnen Sie die Aufgabenstellung und jedes zusätzliche Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer!

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 10 Punkte

(3+3+4 = 10 Punkte) **Binäre Suche, Interpolationssuche:** Gegeben ist ein Array mit den Werten (5,6,9,10,25,28,29). Es soll der Wert 9 gesucht werden.

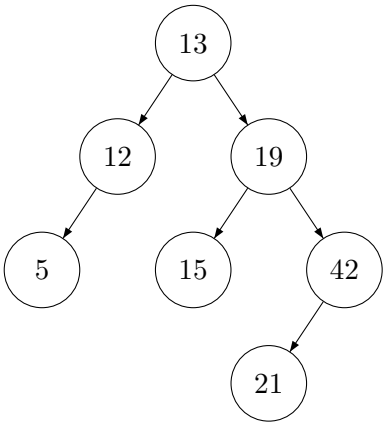
- (3 Punkte) Geben Sie an, mit welchen Elementen bei **binärer Suche** die 9 verglichen wird, bis sie gefunden wird.

- (3 Punkte) Geben Sie an, mit welchen Elementen bei **Interpolationssuche** die 9 verglichen wird, bis sie gefunden wird.

- (2+2 Punkte) Sind diese Aussagen richtig oder falsch? (korrekte Antwort je +1 Punkt, falsche Antwort je -1 Punkt; passendes Beispiel oder kurze Begründung je 1 Punkt)
  - Binäre Suche benötigt immer höchstens so viele Vergleiche wie Interpolationssuche.
  - Binäre Suche benötigt immer mindestens so viele Vergleiche wie Interpolationssuche.

Aufgabe 2 — 8 Punkte

(5+3 = 8 Punkte) **Löschen in binären Suchbäumen:** Gegeben ist folgender binäre Suchbaum:



- (5 Punkte) Werden in einem binären Suchbaum Knoten mit zwei Kindern gelöscht, so werden diese durch das nächstgrößere (Inordernachfolger) oder nächstkleinere Element (Inordervorgänger) ersetzt. Geben Sie die beiden Suchbäume an, die durch Löschen des Elements 13 entstehen können.

  Was muss man nach dem Ersetzen des zu löschenden Elements noch tun?

- (3 Punkte) Beschreiben Sie in 2 Sätzen, wie der Inordervorgänger oder Inordernachfolger gefunden werden kann.

Aufgabe 3 — 12 Punkte

(12 Punkte) Fügen Sie in einen anfangs leeren **AVL-Baum** folgende Zahlen in dieser Reihenfolge ein: 1, 3, 8, 4, 7, 9, 6. Wenn eine Rotation notwendig ist, geben Sie den AVL-Baum jeweils vor und nach dieser Rotation an. Geben Sie in jedem Fall den AVL-Baum an, nachdem alle Elemente eingefügt wurden.

Aufgabe 4 — 11 Punkte

(9+2 Punkte) **Insertionsort:** Gegeben ist ein Array mit den Zahlen (9,7,3,4,2,6,5).

(9 Punkte) Führen Sie Insertionsort durch:

- Geben Sie den Inhalt des Arrays nach jedem kompletten Durchlauf der äußeren Schleife an.
- Tragen Sie jeweils die Anzahl der in diesem Durchlauf durchgeführten Vertauschungen ein.

Hinweis: die Anzahl der Zeilen in der Tabelle ist größer als die tatsächlich auszufüllende Anzahl der Zeilen.

Durchlauf								Vertauschungen
vor 1.	9	7	3	4	2	6	5	-
nach 1.								
nach 2.								
nach 3.								
nach 4.								
nach 5.								
nach 6.								
nach 7.								
nach 8.								
nach 9.								
nach 10.								
nach 11.								
nach 12.								
nach 13.								
nach 14.								
nach 15.								

(2 Punkte) Wie groß ist der Fehlstand (Inversionszahl) des obigen Feldes (9,7,3,4,2,6,5)?

Aufgabe 5 — 10 Punkte

(10 Punkte) Bei **Quicksort** wird als Pivot-Element das Element in der Mitte gewählt (Index `(l+r)/2`). Folgendes Feld sei gegeben. Führen Sie die erste Iteration bei Quicksort durch (also Wandern der Zeiger mit Vertauschungen bis die beiden Zeiger übereinander hinweg gelaufen sind) – in Ihrer Bearbeitung muss ersichtlich sein, welche Elemente getauscht wurden und nach welchem Kriterium das Wandern der Zeiger beendet wird:

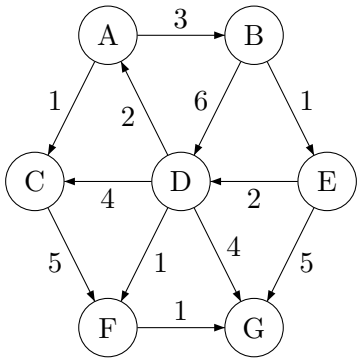
10 72 15 14 66 6 45 49 31 56 91 75 39 88 94 55 19 18 73

Geben Sie an, mit welchen Indexgrenzen die beiden resultierenden rekursiven Aufrufe von Quicksort erfolgen.

Aufgabe 6 — 9 Punkte

(9 Punkte) **Kürzeste-Wege-Berechnung mit dem Algorithmus von Dijkstra:**

Führen Sie den Dijkstra-Algorithmus auf folgendem Graphen aus:
Zu Beginn ist lediglich die Entfernung zum Startknoten B mit d(B)=0 bekannt. Im Dijkstra-Algorithmus wird in jeder Iteration ein Knoten aus der Randmenge R entfernt. Geben Sie in jeder Zeile den Zustand der Menge R jeweils vor diesem Entfernen eines Knotens an, den durch delete_min gewählten Knoten und die berechneten Entfernungen d() am Ende des Durchlaufs für alle Knoten.



	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	$\emptyset$	„Initialisierung“	∞	0	∞	∞	∞	∞	∞
1.	{B}								
2.									
3.									
4.									
5.									
6.									
7.									
8.									

Aufgabe 7 — 24 Punkte

(2+18+2+2 = 24 Punkte) **Ein Baumalgorithmus:** Bei Bäumen spielt oft das Level (Ebene) eines Knotens eine Rolle – der Wurzelknoten ist auf Level 1, dessen Kinder sind auf Level 2, usw. In dieser Aufgabe sollen die Werte in den Knoten jeweils mit deren Level gewichtet aufsummiert werden. Für den Baum aus Aufgabe 2 soll der Wert

$$1 \cdot 13 + 2 \cdot 12 + 3 \cdot 5 + 2 \cdot 19 + 3 \cdot 15 + 3 \cdot 42 + 4 \cdot 21 \quad = \quad 345$$

berechnet werden.

- (2 Punkte) Geben Sie eine C/C++-Datenstruktur für einen binären Baum an. Die Werte im Baum sollen vom Datentyp `int` sein. Der Datentyp für eine Variable, die auf die Wurzel eines binären Baums zeigt, soll dabei `binBaumPtr` heißen.
- (18 Punkte) Schreiben Sie eine C/C++-Funktion `int gesumme(const binBaumPtr root)`, die als Parameter den Zeiger auf die Wurzel eines binären Baums bekommt und als Ergebnis (mit dem Befehl `return`) die gewichtete Summe zurückgibt (eine Ausgabe per `printf` oder `cout` ist nicht gefordert). Ist der Baum leer, so ist der Rückgabewert 0.

  (Hinweis: wahrscheinlich werden Sie eine weitere Hilfsfunktion benötigen)
- (2 Punkte) Geben Sie den Aufwand für den schlechtesten Fall in O-Notation an.
- (2 Punkte) Wieviel Schritte benötigen Sie im besten Fall?

Beachten Sie: die Parameter sind an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe des Baumes kümmern.

Nachtermin Teilklausur II in Informatik – Lösungshinweise

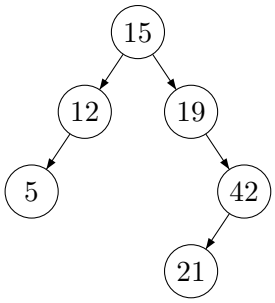
Aufgabe 1 — Binäre Suche, Interpolationssuche

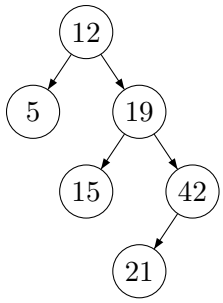
Gegeben war ein Array mit den Werten (5,6,9,10,25,28,29). Gesucht: der Wert $s = 9$.

- Bei **binärer Suche** wird jeweils in der Mitte des verbleibenden Teilfeldes gesucht ⇒ Vergleiche mit 10, 6, 9 ⇒ gefunden
- Bei **Interpolationssuche** wird jeweils am Index $m = l+(s-a[l])\cdot(r-l)/(a[r]-a[l])$ gesucht ⇒ Vergleiche mit $a[0+(9-5)\cdot(6-0)/(29-5)] = a[1] = 6$ und $a[2+(9-9)\cdot(6-2)/(29-9) = a[2] = 9$ ⇒ gefunden
- Bei diesem Beispiel benötigt binäre Suche mehr Vergleiche als Interpolationssuche. Hätte man die 10 gesucht, wäre es andersherum gewesen. Es ist also keines der beiden Verfahren garantiert schneller als das andere.

Aufgabe 2 — Löschen in binären Suchbäumen

Links der entstehende Suchbaum bei Ersetzen der 13 durch den Inordernachfolger, rechts bei Verwendung des Inordervorgängers. Die entstehende Lücke ist entweder ein Blatt (kann gelöscht werden) oder hat genau ein Kind (wird samt Unterbaum an die Lücke hochgezogen) – je einer der Fälle tritt hier auf.



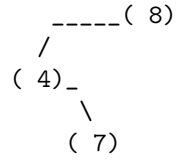


Aufgabe 3 — Einfügen in AVL-Bäume

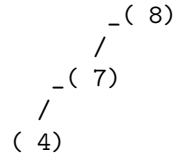
```
Werte: 1, 3, 8                      ( 1)_
Balance an Knoten 1 verletzt             \
Unterbaum vor der Rotation:             ( 3)_
                                             \
                                            ( 8)

Linksrotation                          _( 3)_
Baum nach der Rotation:              /       \
                                   ( 1)      ( 8)
```

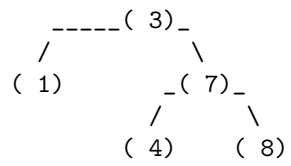
Nächste Werte: 4, 7
Balance an Knoten 8 verletzt
Unterbaum vor der Rotation:



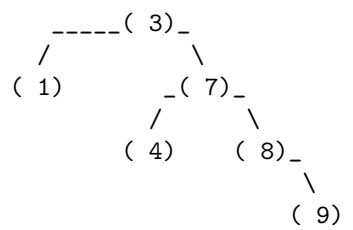
Linksrechtsrotation
Unterbaum nach der Linksrotation:



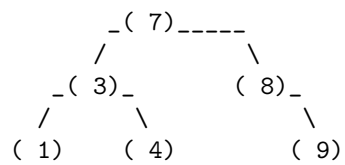
Baum nach der Rechtsrotation:



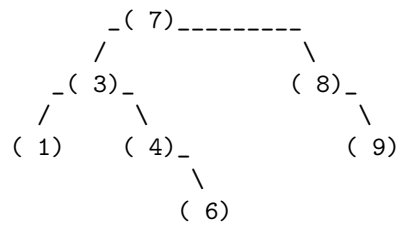
Nächster Wert: 9
Balance an Knoten 3 verletzt
Unterbaum vor der Rotation:



Linksrotation
Baum nach der Rotation:



Nächster Wert: 6



Aufgabe 4 — Insertionsort

Durchlauf								Vertauschungen
vor 1.	9	7	3	4	2	6	5	-
nach 1.	7	9	3	4	2	6	5	1
nach 2.	3	7	9	4	2	6	5	2
nach 3.	3	4	7	9	2	6	5	2
nach 4.	2	3	4	7	9	6	5	4
nach 5.	2	3	4	6	7	9	5	2
nach 6.	2	3	4	5	6	7	9	3

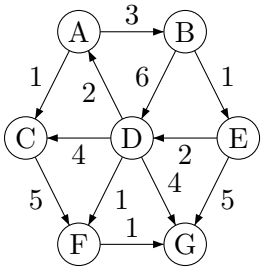
Der Fehlstand/Inversionszahl kann hier leicht als Summe der Vertauschungen ermittelt werden (zur Erinnerung: jeder Tausch eines benachbarten Pärchens verringert den Fehlstand um 1 – oder man zählt noch mal extra): das Ergebnis ist 14.

Aufgabe 5 — Quicksort

10 72 15 14 66 6 45 49 31 56 91 75 39 88 94 55 19 18 73

Pivot-Element ist die 56. Es werden zunächst die 72 und 18 vertauscht, dann die 66 und 19. Der linke Zeiger wandert dann bis zum Pivot-Element, der rechte zur 55 (Tausch 56 und 55). Ein weiterer Tausch der 91 mit 39, beide Zeiger stehen danach auf der 75. Die Schleife wird ein weiteres Mal betreten, der rechte Zeiger wandert noch ein Element nach links, es findet aber kein Tausch mehr statt, da die Zeiger bereits übereinander hinweggewandert sind. Die rekursiven Aufrufe haben die Parameter (0,10) und (11,18).

Aufgabe 6 — Dijkstra-Algorithmus



	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	$\emptyset$	„Initialisierung“	∞	0	∞	∞	∞	∞	∞
1.	{B}	B	∞	0	∞	6	1	∞	∞
2.	{D,E}	E	∞	0	∞	3	1	∞	6
3.	{D,G}	D	5	0	7	3	1	4	6
4.	{A,C,F,G}	F	5	0	7	3	1	4	5
5.	{A,C,G}	A	5	0	6	3	1	4	5
6.	{C,G}	G	5	0	6	3	1	4	5
7.	{C}	C	5	0	6	3	1	4	5
8.	$\emptyset$	„fertig“	5	0	6	3	1	4	5

Da die Knoten A und G beide die Entfernung 5 hatten, hätte man auch G vor A aus der Randmenge wählen können. Die anderen Zeilen ändern sich dadurch nicht.

Aufgabe 7 — Ein Baumalgorithmus

Man kann die Knoten des Baums durchlaufen und das Level als weiteren Parameter einer Hilfsfunktion übergeben. Dabei ist die Reihenfolge, in der die Werte in den Unterbäumen berechnet werden, unerheblich (im Code unten ist es Inorderreihenfolge).

```
struct binBaum;
type binBaum * binBaumPtr;
struct binBaum {
  int value;
  binBaumPtr left,right;
};

int hilfsumme(const binBaumPtr root, const int level) {
  if (root==NULL) { return 0; }
  else { return hilfsumme(root->left,level+1)
              +root->value*level
              +hilfsumme(root->right,level+1);
  }
```

```
}

int gesumme(const binBaumPtr root) {
  return hilfsumme(root,1);
}
```

Jeder Knoten wird genau 1 Mal betrachtet, der Gesamtaufwand ist linear, also $O(n)$ – im besten Fall wird derselbe Aufwand wie im schlechtesten Fall benötigt.