Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 14. Mai um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!
- Am Freitag den 11. Mai, sowie am Dienstag den 15.Mai findet keine Vorlesung statt.

Aufgabe 1 (Eigenschaften von Heaps): (1 + 5 Punkte)

Beweisen Sie die folgenden, aus der Vorlesung bekannten, Lemmata:

a) Ein n-elementiger Heap hat die Höhe $\lfloor \log_2 n \rfloor$.

b) Ein Heap hat maximal $\lceil n/2^{h+1} \rceil$ Knoten mit der Höhe h.

Hinweis: Die Höhe h des Knotens entspricht der Länge des längsten Pfades zu einem Blatt des Heaps.

Aufgabe 2 (Heapsort): (2 + 2 + 4 Punkte)

a) In der Vorlesung wurden so genannte Max-Heaps vorgestellt. D.h. jedes Element ist größer/gleich seiner Kinder. Analog hierzu ist ein *Min-Heap* ein Heap, bei dem jedes Element kleiner/gleich seiner Kinder ist. Bestimmen sie, ob die folgenden Arrays Max-Heaps oder Min-Heaps sind. Für den Fall, dass es sich weder um den einen noch den anderen handelt, geben sie an, wo die Heapeigenschaft verletzt ist.

1.)

37	21	32	17	10	24	22	1	9	15	14

2.)

24	21	32	17	10	37	22	1	9	15	14

3.)

10	29	15	20	30	21	16	40	31	41	35

4.)

10	20	15	29	31	21	16	40	30	41	35

b) Stellen Sie das folgende Array als Baum dar. Skizzieren Sie (durch nummerierte Pfeile) die, zur Herstellung der Max-Heap Eigenschaft nötigen Versickerungen. Geben Sie den resultierenden Heap sowohl als Baum als auch als Array an.

9	2	1	4	3	2	1	5	7	1

c) Sortieren Sie das in Aufgabenteil **b)** gegebene Array mit Hilfe von Heapsort. Geben Sie für jeden Sortierschritt den neuentstandenen Heap als Array sowie als Baum an und skizzieren Sie nötige Versickerungen in der Baumdarstellung. Geben Sie auch die Reihenfolge der Versickerungen an.

Aufgabe 3 (Rekursiver Sortieralgorithmus): (2 + 3 Punkte)

```
sort(int [ ] A, int l, int r) {
  if (A[l] > A[r]){
    exchange(A[l], A[r]);
  }
  if (l < r-1){
    k:= (r-l+1) div 3;
    sort(A, l, r-k);
    sort(A, l+k, r);
    sort(A, l, r-k);
  }
}
```

a) Bestimmen Sie für den gegebenen Sortieralgorithmus die Komplexitätsklasse Θ im Best-, Worst- und Average-Case für den Aufruf `sort(A,0,n-1)` in Abhängigkeit von `n`, der Anzahl der zu sortierenden Elemente aus dem Array `A`.

b) Der vorgestellte Algorithmus ist nicht stabil. Ändern Sie den Algorithmus so ab, dass er stabil wird.

Aufgabe 4 (Rekursionsgleichungen): (2 + 2 + 4 + 3 Punkte)

Geben Sie für die folgenden Rekursionsgleichungen die Komplexitätsklasse (Θ) an. Begründen Sie Ihre Antwort.

a) $T(n) = 8T(\frac{n}{4}) + 2n \cdot \sqrt{n} + 10n$

b) $T(n) = 5T(\frac{n}{4}) + n \cdot \log_2 n$

c) $T(n) = 4T(\sqrt[5]{n}) + (\log_2 n)^7$

d) $T(n) = T(\frac{n}{2}) + 2T(\frac{n}{4}) + n$

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 21. Mai um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Sortieralgorithmus): (5 + 4 + 1 + 3 Punkte)

In dieser Aufgabe werden sie einen neuen Sortieralgorithmus implementieren. Dieser verfolgt, ebenso wie Quicksort, den Divide-and-Conquer Ansatz. Die Aufteilung in Teilprobleme erfolgt in diesem Fall durch einer Abwandlung des Dutch-National-Flag Algorithmus aus Vorlesung sieben.
Der Sortieralgorithmus ist in Java zu implementieren. Im L2P finden Sie ein Klassengerüst, das zu benutzen ist. Bitte kommentieren Sie Ihre Lösung und schicken Sie den Java Code an Ihren Tutor.

a) Implementieren Sie, basierend auf dem Dutch-National-Flag Algorithmus, einen Partitionierung Algorithmus `IntTuple partion(int[] E, int left, int right, int red, int blue)`, der für zwei gegebene Pivotelemente `red`, `blue` mit `red` $\leq$ `blue`, das gegebene Array `E` in dem Bereich zwischen `left` und `right` in drei Teile partitioniert. Die Partionierung soll so realisiert werden, dass der linke Teil die Elementen enthält, die kleiner sind als die beiden Pivotelemente, sowie der rechte Teil die Elementen die größer sind als die beiden Pivotelemente. Der mittlere Teil bleibt den restlichen Elementen vorbehalten. Die Grenzen der einzelnen Bereiche werden als `IntTuple` zurückgegeben.

b) Implementieren Sie nun die Methode `void sort(int[] E)`, die das gegebenes Array `E` sortiert. Wählen Sie hierzu zwei Pivotelemente und teilen Sie das Array, entsprechend dem Algorithmus aus **a)** in drei Partionen. Sortieren Sie dann rekursiv jede der drei Partionen. Achten Sie darauf, dass der Algorithmus stets terminiert.

c) Ist der von Ihnen implementierte Algorithmus stabil? Begründen Sie Ihre Antwort.

d) Geben Sie die Worst-Case sowie Best-Case Laufzeit Ihres Algorithmus an. Begründen Sie Ihre Antwort.

Aufgabe 2 (Höhergradige Heaps): (3+2+4+3 Punkte)

In Übung 5 haben Sie Heapsort, basierend auf *binären Heaps*, zum Sortieren genutzt. In binären Heaps hat jedes Element bis zu zwei Kinderelemente. Das Konzept der binären Heaps lässt sich auf *d-Heaps* erweitern. Ein *d*-Heap ist ein Baum mit Verzweigungsgrad *d* (d.h. jedes Element besitzt bis zu *d* Kinder), der die *max-Heapeigenschaft* (alle Kinderelemente haben *niedrigere* Werte) erfüllt.

a) Geben Sie die Formel an mit der, für die Arraydarstellung eines *d*-Heaps, die Position der Kinder des Elementes an Position *i* bestimmt werden kann.

In welchem Bereich des Arrays ist die Heapeigenschaft immer erfüllt?

b) Stellen Sie den im folgenden Array repräsentierten *3-Heap* als Baum dar.

40	32	24	36	28	30	5	12	21

c) Geben Sie eine modifizierte Version des in der Vorlesung vorgestellten Algorithmus `sink` an, der Elemente in einem *d*-Heap versickern lässt. Der Grad *d* wird hierbei als Parameter übergeben.

d) Sortieren Sie das in **b)** gegebene Array entsprechend der *Heapsortmethode für 3-Heaps*.

Aufgabe 3 (Komplexitätsanalyse): (3 + 3 Punkte)

a) Bestimmen Sie für den folgenden Code die asymptotische Laufzeit Θ für den Aufruf `berechne(n,m)` in Abhängigkeit von den Parametern n und m (für $n, m \geq 0$). Begründen Sie Ihre Antwort und geben Sie Zwischenschritte an.

```
int berechne(int n, int m){
        int k = m;
        while(n > 0){
            k = k * m;
            n--;
        }
        for(int i = k; i > n; i--){
            k = k - m;
            n++;
        }

        return k;
}
```

b) Bestimmen Sie für den folgenden Code die asymptotische Laufzeit Θ für den Aufruf `berechne(n, k)` in Abhängigkeit von den nicht negativen Parametern n und k (für $n, k \geq 0$). Begründen Sie Ihre Antwort und geben Sie Zwischenschritte an.

```
int berechne(int n, int k){
    if(n == 0)
        return k;

    int value = berechne(n/3, k);

    if(n < 42){

        for(int i = 0; i < k*k; i++)
            value += i * n;

        value += berechne(n/3, k) + 2;

    }else{

        for(int i = 0; i < n; i++)
            value += i * n;

        value += 3 * berechne(n/3, k) + 3;
    }

    return value;

}
```

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

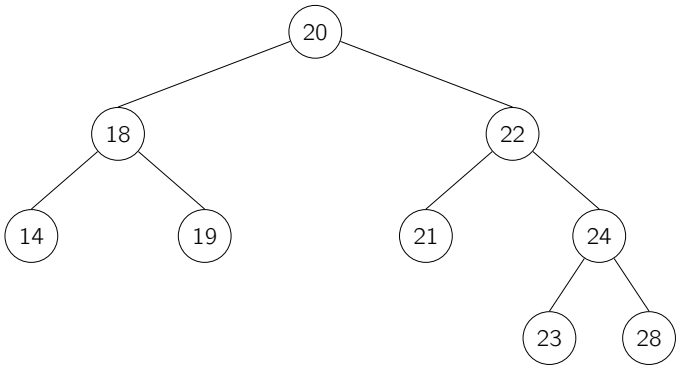
- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 4. Juni um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!
- Zur Bearbeitung der Aufgaben 3 wird Stoff aus der Vorlesung am Dienstag den 22. Mai benötigt.
- Das 8. Übungsblatt kommt am Freitag, den 25. Mai und ist bis Montag den 11. Juni zu bearbeiten.

Aufgabe 1 (Binäre Suchbäume): (3 + 4 + 3 Punkte)

a) Geben Sie den binären Suchbaum an der entsteht, wenn die folgenden Werte in gegebener Reihenfolge in einen *leeren* Suchbaum eingefügt werden:

8, 5, 3, 4, 7, 11, 10

b) Gegeben sei der folgende binäre Suchbaum:

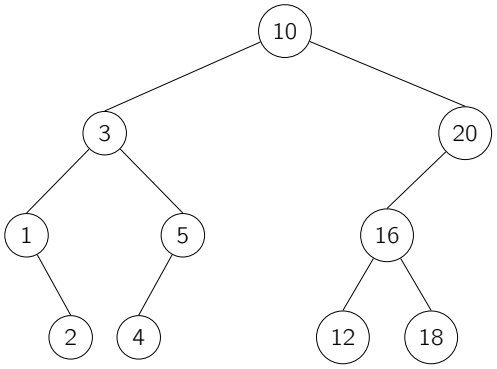


Geben Sie jeweils den binären Suchbaum an, der entsteht wenn, entsprechend dem in der Vorlesung vorgestellten Verfahre, nacheinander die Zahlen **14, 20, 22** aus dem gegeben Baum gelöscht werden.

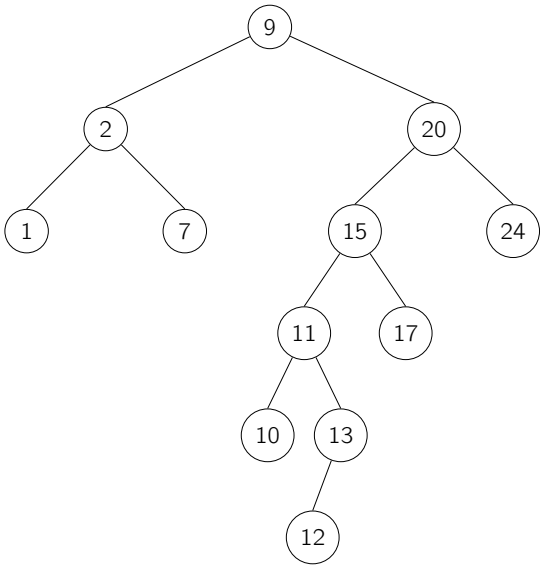
c) In der Vorlesung wurde ein Verfahren zur *Bestimmung des Nachfolgers* eines Elementes im binären Suchbaum vorgestellt. Hierfür wird der Pfad zwischen den beiden Elementen zurückgelegt. Wie lang ist dieser Pfad maximal für einen Baum mit n Elementen? Beweisen Sie Ihre Aussage.

Aufgabe 2 (Rotationen): (3 + 5 Punkte)

a) Geben Sie für den folgenden binären Suchbaum die Binärbäume an, die entstehen, wenn eine **Rechtsrotation** auf den Knoten mit dem Wert **20** ausgeführt wird, dann eine **Linksrotation** auf den Knoten mit Wert **3** und zuletzt eine **Linksrotation auf die Wurzel** des entstandenen Baumes:



b) Geben Sie **maximal vier Rotationen** an, die den folgenden Baum der **Höhe fünf** in einen Baum der **Höhe drei** transformieren. Geben Sie auch den Zustand des Baumes nach jeder Rotation an.



Aufgabe 3 (Rot-Schwarz Bäume): (5 + 6 Punkte)

a) Bestimmen Sie die Anzahl verschiedener Rot-Schwarz-Bäume mit Schwarzhöhe *zwei*, wenn die Schlüsselwerte ignoriert werden. D.h. lediglich die Farbe eines Knotens und seine Position im Baum sind entscheidend. Begründen Sie Ihre Antwort.

b) Zeigen Sie, dass zu jeder Höhe h ein Rot-Schwarz-Baum $B(h)$ existiert mit der folgenden Anzahl roter Knoten $r(B(h))$:

$$r(B(h)) = \sum_{i=2}^{h} \left((1 + (-1)^{i-h}) \cdot 2^{i-2} \right)$$

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 11. Juni um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Countingsort): (4 + 2 Punkte)

a) Das folgendes Array ist mit Countingsort zu sortieren. Geben Sie das Histogramm- und das Positionsarray vor dem ersten Einfügen ins Ausgabearray an, sowie das Positions- und Ausgabearray nach jedem Einfügeschritt.

4	3	0	1	4	2	3	7	3

b) Der in der Vorlesung vorgestellte Algorithmus Countingsort fügt die Elemente des Eingabearrays von hinten nach vorne in das Ausgabearray ein. Welche Nachteile ergäben sich, wenn man das Eingabearray stattdessen von vorne nach hinten durchlaufen würde?

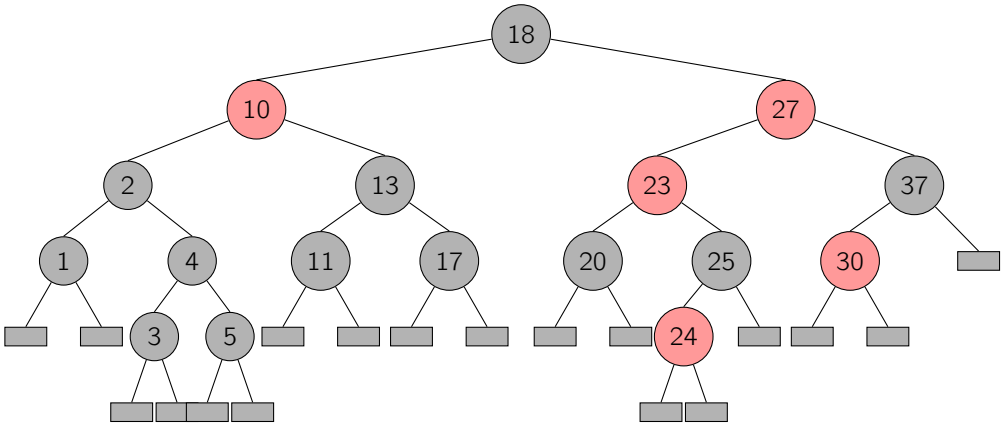
Aufgabe 2 (Rot-Schwarz Bäume): (6 + 4 Punkte)

a) Fügen Sie die folgenden Werte in der gegebenen Reihenfolge in einen leeren Rot-Schwarz-Baum ein:

$$12, 7, 19, 5, 6, 23, 25, 21, 15, 20$$

Zeichnen Sie den Baum jeweils nach dem Einfügen und nach jeder erfolgten Rotation und Umfärbung. Machen Sie außerdem kenntlich, welche Transformation in welchem Schritt passiert.

b) Gegeben sei der folgende Baum:



Handelt es sich um einen Rot-Schwarz-Baum? Ist dies nicht der Fall, stellen Sie die Rot-Schwarz-Eigenschaft durch **Umfärben** von **maximal zwei** Knoten her.

Löschen Sie nun die folgenden Werte in der gegebenen Reihenfolge aus ihrem (ggf. "reparierten") Rot-Schwarz-Baum: 2, 18. Zeichnen Sie den Baum jeweils nach dem Löschen und nach jeder erfolgten Rotation und Umfärbung. Machen Sie außerdem kenntlich, welche Transformation in welchem Schritt passiert.

Aufgabe 3 (Hashing): (3 + 3 + 2 Punkte)

a) Betrachten Sie die folgenden Funktionen, die ein Wort $s = a_1 \dots a_n$ auf einen Hash-Wert zwischen 0 und m abbilden. Dabei sind die a_i als ASCII-Werte gegeben, also $0 \leq a_i \leq 127$ für alle i.

- $h_1(s) = n \mod m$
- $h_2(s) = (\sum_{k=1}^{n} a_k) \mod m$
- $h_3(s) = (\sum_{k=1}^{n} k \cdot a_k) \mod m$
- $h_4(s) = ((h_{good}(s)^{p-1} \mod p) \mod m)$

wobei $h_{good}(s)$ eine Hash-Funktion ist, die alle wichtigen an eine Hash-Funktion gestellten Eigenschaften erfüllt und p eine Primzahl ist. Diskutieren Sie, inwiefern die Funktionen h_i $(1 \leq i \leq 4)$ als Hash-Funktionen geeignet sind.

b) Gegeben sei eine Hash-Table der Größe m und eine beliebige Hash-Funktion $h\colon U \to \{0, \dots, m-1\}$. Die Menge U habe nun mindestens $n \cdot m$ Elemente, also $|U| \geq n \cdot m$. Zeigen Sie, dass U eine Teilmenge U_0 der Größe n besitzt ($|U_0| = n$), so dass

$$h(x_1) = h(x_2) \quad \text{für alle} \quad x_1, x_2 \in U_0$$

Was haben Sie damit für die Worst-Case-Laufzeit der Suche mittels Hashing mit Verkettung bewiesen?

c) Gegeben sei nun eine Hash-Table mit einer initialen Größe von 1000 und eine Hash-Funktion, die ein gleichverteiltes Hashing gewährleistet. Nach wie vielen Einfügungen müssen sie a-priori mit einer Kollisionswahrscheinlichkeit von mehr als 80% rechnen?

Um die Anzahl von Kollisionen beim Hashing gering zu halten, kann man die Größe der Hash-Table nach einer gewissen Anzahl von Einfügungen erhöhen. Nach welcher Anzahl k von Einfügeoperationen muss die Tabelle das erste Mal vergrößert werden, wenn bei den vorhergehenden $k-1$ Einfügeoperationen keine Kollision auftrat und die Wahrscheinlichkeit für eine Kollision bei der k-ten Einfügung weniger als 20% betragen soll? Begründen Sie ihre Antwort.

Aufgabe 4 (Hashing): (3 + 2 + 1 Punkte)

Gegeben sei eine Hash-Table der Größe 23 und die folgenden beiden Hash-Funktionen über der Universalmenge $U = \{0, \dots, 499\}$:

- $h_1(x) =$ Quersumme von x
- $h_2(x) = x \mod 23$

a) Fügen Sie die Werte 12, 99, 111, 76, 23, 30 sowohl mit h_1 als auch h_2 mittels

(i) Hashing mit Verkettung

(ii) Hashing mit linearem Sondieren

in jeweils eine Tabelle ein (es sind also 4 Tabellen zu erstellen). Geben Sie die nicht-leeren Teile der Tabellen nach jedem Einfügeschritt an.

b) Löschen Sie nacheinander die Werte 111, 12 und 76 aus allen Tabellen aus Aufgabe **a)**. Geben Sie die nicht-leeren Teile der Tabellen nach jedem Löschschritt an. Erläutern Sie, welches Vorgehen dafür jeweils nötig ist.

c) Suchen Sie in den Tabellen, die aus Teilaufgabe **b)** resultierten, nach dem Wert 30. Erläutern Sie, welches Vorgehen dafür jeweils nötig ist.

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 18. Juni um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Duplikaterkennung): (4 + 4 Punkte)

Da Ihr Chef kürzlich einen Artikel über die Vorzüge von Hashing gelesen hat, wettet er mit seinem alten Schulfreund, dass er Folgen von Zahlen, welche der Schulfreund nennt, schneller als dieser auf Duplikate testen kann. Er bittet Sie, die Umsetzung per Hashing zu übernehmen. Sie implementieren (natürlich unter Protest!) ein Verfahren, welches Hashing mit Verkettung als Kollisionsauflösung benutzt, wie folgt.

- Ihre Methode übernimmt ein Array der Länge n von natürlichen Zahlen als Parameter.
- Sie fügen die Elemente nacheinander in die Hashtabelle ein.
- Gibt es eine Kollision der Hashwerte (sind also schon Elemente in der Liste der entsprechenden Zelle), so prüfen sie nacheinander alle Elemente der Liste, ob der neu eingefügte Wert sich bereits darunter befindet.
- Finden Sie das entsprechende Element dabei, so geben Sie "ja" zurück, um anzuzeigen, dass ein Duplikat gefunden wurde.
- Finden Sie das Element jedoch nicht, so fügen Sie das neue Element wie bisher am Anfang der Liste an und fahren mit dem nächsten einzufügenden Element fort.

Die Hashfunktion, die sie dabei benutzen sollen, hat ihr Chef in einer Übungsaufgabe zur Vorlesung "Datenstrukturen und Algorithmen" gefunden. Sie lautet:

$$h(k) = (k + (k \mod 23)^2 + z) \mod 3001$$

wobei 3001 die Größe der Hashtabelle und z eine Zufallszahl ist, die zur Laufzeit einmal gewählt wird und dann beibehalten wird. Sie testen Ihre Implementierung mit ein paar zufälligen Testeingaben und ermitteln eine Laufzeit von weniger als einer halben Sekunde für Listen von 500000 Elementen und sind damit einigermaßen zufrieden.

a) Stolz zeigt ihr Chef seinem Freund Ihren Algorithmus. Dieser schaut eine Zeit lang darauf, verschwindet für eine Minute und gibt ihm daraufhin eine Eingabesequenz der Länge 500000. Verdutzt stellt ihr Chef fest, dass die Duplikatenbestimmung erstaunlich lange läuft, bevor sie terminiert (nämlich über 10 Minuten). Aufgrund dieser Schmach fragt Sie ihr Chef später wie "in Gottes Namen das passieren konnte!" Erklären Sie ihm das. Dazu gehört eine Worst-Case-Analyse und eine Hypothese, wie der Freund es schaffen konnte, so schnell mit so einer Zahlenfolge aufzuwarten.

b) Zähneknirschend nimmt ihr Chef Ihre Erklärungen zur Kenntnis, bittet Sie seine Ehre zu retten und stellt Ihnen einen großen Bonus in Aussicht. Schlagen Sie ihm ein Verfahren vor, welches eine asymptotisch bessere Worst-Case-Komplexität als die aus **a)** besitzt und sich auf Mittel stützt, die sie bereits in der Vorlesung gelernt haben. Dabei reicht eine Beschreibung des Verfahrens in Worten. Vergessen Sie aber nicht, zu erklären, wieso die Worst-Case-Komplexität Ihres Verfahrens besser als diejenige aus **a)** ist.

Aufgabe 2 (Hashing): (8 + 2 + 2 + 2 Punkte)

In einem kleinen Studentenkino mit $m = 23$ Plätzen wird ein Film gezeigt. Um dem Ansturm Herr zu werden, sollen die Plätze mit einem offenen Hashverfahren (Hashing mit Sondieren als Kollisionsauflösung) auf die Wartenden verteilt werden. Als Schlüssel werden dabei nur die beiden letzten Ziffern der Matrikelnummer verwendet. Betrachten Sie die folgenden beiden Hashfunktionen:

- $h_1(x)$:= Quersumme von x
- $h_2(x)$:= x mod 23 (Division-Rest-Methode)

a) Welche Platznummern erhalten die Besucher, wenn sie in der gegebenen Reihenfolge das Kino betreten?

$$6,\ 16,\ 61,\ 87,\ 69,\ 90,\ 4,\ 43,\ 57,\ 4,\ 12,\ 80,\ 46$$

Verwenden Sie jeweils folgende Hashfunktionen:

(i) lineares Sondieren mit h_1 und mit h_2 als Hashfunktion

(ii) quadratisches Sondieren mit h_1 und mit h_2 als Hashfunktion und mit $c_1 = 2$ und $c_2 = 3$ als Konstanten,

(iii) Doppelhashing mit h_1 als erster und h_2 als zweiter Hashfunktion. Inwieweit muss h_2 geändert werden, um Probleme zu vermeiden?

b) Warum ist ein geschlossenes Hashing (Hashing mit Verkettung als Kollisionsauflösung), in dem geschildertem Szenario, nicht praktikabel bzw. sinnvoll?

c) Angenommen, wir haben eine perfekte Hashfunktion und fügen die obigen Elemente in einen Hash der Größe 23 ein. Wie viele Sondierungen sind dann durchschnittlich bei erfolgreicher Suche nötig?

d) Angenommen, wir haben eine perfekte Hashfunktion und fügen die obigen Elemente in einen Hash der Größe 23 ein. Wie viele Sondierungen sind dann durchschnittlich bei nicht erfolgreicher Suche nötig?

Aufgabe 3 (Sondieren): (6 + 2 Punkte)

a) Es sollen zwei gegebene Hashfunktionen h_1 und h_2 für doppeltes Hashing im Rahmen des Hashing mit offener Adressierung benutzt werden. Das heißt, es wird die Hashfunktion

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m$$

benutzt, wobei m die Größe der Hashtabelle ist. Es sei nun für einen Schlüssel k bekannt, dass der größte gemeinsame Teiler von $h_2(k)$ und m ein Wert $d \geq 1$ ist. Beweisen Sie, dass die Suche nach einem freien Einfügeplatz für k genau $(1/d)$-tel der Tabellenplätze überprüft, bevor sie schließlich wieder zum Ausgangspunkt (also $h_1(k)$) zurückkehrt. Gehen Sie dabei davon aus, dass alle überprüften Plätze belegt sind und daher die jeweils nächste Sondierung notwendig ist.

b) Was bedeutet das Ergebnis von **a)**, falls $d = 1$ ist? Welche Folgerung ergibt sich damit für die kluge Wahl von m?

Prof. Dr. Ir. Joost-Pieter Katoen

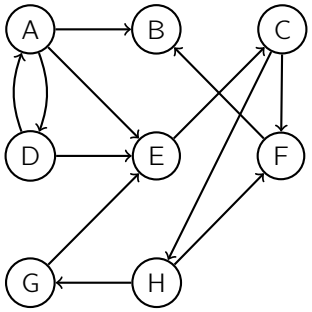
Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 25. Juni um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Elementare Graphalgorithmen I): (3 + 1 + 6 = 10 Punkte)

Betrachten Sie den folgenden gerichteten Graphen G_1:



a) Geben Sie den Kondensationsgraphen $G_1 \downarrow$ an. Beschriften Sie die Knoten im Kondensationsgraphen mit den Namen aller Knoten, die zur jeweiligen starken Zusammenhangskomponente gehören. Bilden beispielsweise die Knoten 1 und 3 eine starke Zusammenhangskomponente, so sieht der zugehörige Knoten im Kondensationsgraphen wie folgt aus:

b) Geben Sie den transponierten Graphen G_1^T an.

c) Zeigen Sie, dass für jeden gerichteten Graphen G gilt, dass die Transposition des Kondensationsgraphen von G gleich dem Kondensationsgraphen der Transposition von G ist:

$$(G\downarrow)^T \;=\; (G^T)\downarrow$$

Aufgabe 2 (Elementare Graphalgorithmen II): (5 + 4 = 9 Punkte)

a) Betrachten Sie die folgenden wöchentlichen Arbeitsschritte im Rahmen einer Lehrveranstaltung.

- Es muss eine **Vorlesung** gehalten werden. Dazu müssen zunächst **Folien** erstellt worden sein.
- Um sinnvolle **Folien** vorzubereiten, muss zunächst der **Lehrstoff** ausgewählt werden.
- Außerdem muss eine **Globalübung** gehalten werden. Dazu müssen natürlich vorher **Übungsaufgaben** und zugehörige **Musterlösungen** erstellt worden sein.
- **Musterlösungen** können selbstverständlich erst erstellt werden, wenn die **Übungsaufgaben** ausgewählt worden sind.
- Die **Übungsaufgaben** müssen ebenfalls auf dem ausgewählten **Lehrstoff** beruhen.
- Darüber hinaus ist ein **Übungsblatt** auszugeben. Um dieses zu erstellen, müssen die **Übungsaufgaben** bereits ausgewählt worden sein.

- Schließlich muss eine **Tutorenbesprechung** stattfinden, um diese auf die Korrektur der Übungen vorzubereiten. Dazu sollten natürlich die **Übungsaufgaben** und **Musterlösungen** vorab erstellt worden sein.

Die verschiedenen Arbeitsschritte sind in folgender Tabelle jeweils mit ihrer Dauer in Tagen aufgeführt.

Arbeitsschritt	Abkürzung	Dauer
Vorlesung	V	1
Folien	F	4
Lehrstoff	L	1
Globalübung	G	1
Übungsaufgaben	A	3
Musterlösungen	M	2
Übungsblatt	B	1
Tutorenbesprechung	T	1

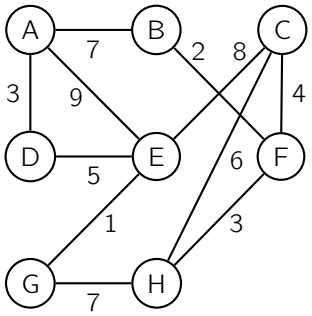
Geben Sie eine topologische Sortierung für den oben beschriebenen Sachverhalt als gerichteten Graphen an. Verwenden Sie für jeden der Arbeitsschritte genau einen Knoten. Sie können die Abkürzungen aus der Tabelle verwenden.

Markieren Sie außerdem den kritischen Pfad in Ihrem Graphen und geben Sie an, nach wievielen Tagen alle Aufgaben frühestens abgeschlossen sein können.

b) Beweisen Sie, dass für alle gerichteten Graphen G gilt, dass das Gewicht eines kritischen Pfades in G genauso groß ist wie das Gewicht eines kritischen Pfades im transponierten Graphen G^T.

Aufgabe 3 (Minimale Spannbäume): (3 + 8 = 11 Punkte)

Betrachten Sie den folgenden ungerichteten gewichteten Graphen G_2:



a) Geben Sie den minimalen Spannbaum an, den Prims Algorithmus findet, wenn wir mit Knoten A starten und bei jeder Wahlmöglichkeit für einen Knoten jeweils den ersten Knoten bzgl. seiner alphabetischen Sortierung wählen.

b) Beweisen Sie die folgende Aussage: Jeder zusammenhängende, ungerichtete, gewichtete Graph G, bei dem keine zwei Kanten das gleiche Gewicht haben, hat genau einen minimalen Spannbaum (d.h. für solche Graphen ist der minimale Spannbaum eindeutig).

Prof. Dr. Ir. Joost-Pieter Katoen

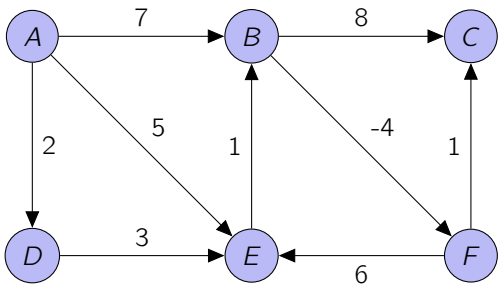
Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 2. Juli um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Bellman-Ford Algorithmus): (5 + 6 + 3 = 14 Punkte)

a) Bestimmen Sie für den Startknoten A die Längen der kürzesten Wege zu jedem Knoten im unten gegebenen Graphen G_1 mit Hilfe des Bellman-Ford Algorithmus. Dabei sei die Reihenfolge der Knoten gegeben durch die alphabetische Sortierung $(A, B, C, \dots)$. Geben Sie für jeden Knoten K die Kosten an, die dem Knoten während der Berechnung zugewiesen werden. Z.B. im Format $K : \infty, 10, 7, 6, 3$. Geben Sie außerdem an, ob der Algorithmus einen Zyklus mit negativem Gewicht erkennt.



b) Passen Sie die Implementierung von Bellman-Ford, die Sie in der Vorlesung kennengelernt haben (Vorlesung 17, Folie 13), so an, dass der kürzeste Pfad zwischen zwei Knoten i und j ausgegeben (nicht zurückgegeben!) wird. Sie dürfen davon ausgehen, dass der übergebene Graph keine negativen Zykel besitzt.

Ihre Funktion soll die folgende Signatur haben:

```
void bellFord(List adjLst[n], int n, int i, int j)
```

Zur Ausgabe können Sie annehmen, dass eine Funktion `ausgabe` mit folgender Signatur existiert:

```
void ausgabe(String text)
```

Außerdem können Sie annehmen, dass `int`-Werte automatisch nach `String` konvertiert werden können.

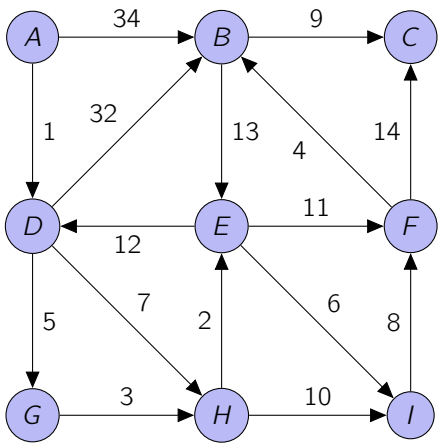
c) Einem Studenten gefällt die Laufzeit des Bellman-Ford Algorithmus nicht und er schlägt das folgende alternative Verfahren vor, um Zyklen mit negativem Gesamtgewicht zu erkennen:

Wir benutzen Sharirs Algorithmus, um alle starken Zusammenhangskomponenten zu finden. Dies kostet $O(|V| + |E|)$. Da $|E| \in O(|V|^2)$ sind die Kosten damit in $O(|V| + |V|^2) = O(|V|^2)$. Für jede starke Zusammenhangskomponente berechnen wir dann die Summe der Gewichte ihrer Kanten. Dies kostet insgesamt $O(|E|)$ und ist also wiederum in $O(|V|^2)$. Ist eine dieser Summen negativ, geben wir TRUE aus, sonst FALSE.

Wo liegt der Fehler, den der Student begangen hat?

Aufgabe 2 (Dijkstra Algorithmus): (5 + 4 + 4 + 3 = 16 Punkte)

a) Gegeben sei der folgende Graph G_2:



Ermitteln Sie mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege von Startknoten A zu allen anderen Knoten des Graphen. Verwenden Sie dazu eine Tabelle der folgenden Form. Notieren Sie für jeden Rechenschritt den aktuell gewählten Knoten zur Verbesserung der Wege und die Länge der bis zu diesem Zeitpunkt möglichen kürzesten Wege für jeden noch nicht abgeschlossenen Knoten ($D[\ldots]$). Streichen Sie Felder der Tabelle, die nicht mehr benötigt werden, durch (dies geschieht, sobald ein Knoten als Baum-Knoten klassifiziert wurde).

Schritt	0	1	2	3	4	5	6	7	8
Knoten									
$D[A]$									
$D[B]$									
$D[C]$									
$D[D]$									
$D[E]$									
$D[F]$									
$D[G]$									
$D[H]$									
$D[I]$									

b) Beweisen Sie die Korrektheit des Dijkstra Algorithmus, d.h. zeigen Sie, dass dieser Algorithmus für einen Graphen G den kürzesten Abstand vom Startknoten s zu jedem anderen von s erreichbaren Knoten in G berechnet. Sie dürfen dazu das Theorem aus Vorlesung 17, Folie 19 nutzen.

c) Beweisen oder widerlegen Sie die folgenden Aussagen für einen gewichteten, zusammenhängenden, ungerichteten Graphen G:

i) Der vom Dijkstra Algorithmus berechnete SSSP-Baum ist ein Spannbaum.

ii) Der vom Dijkstra Algorithmus berechnete SSSP-Baum ist ein minimaler Spannbaum.

d) Arbeitet der Dijkstra Algorithmus immer korrekt, wenn man ihn auf einen Graphen mit negativen Gewichten anwendet, der keine Zyklen mit negativem Gesamtgewicht enthält? Begründen Sie Ihre Antwort!

Prof. Dr. Ir. Joost-Pieter Katoen

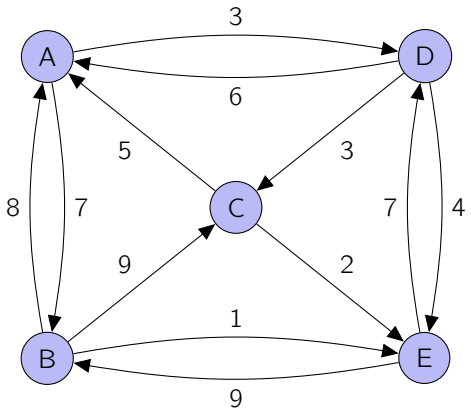
Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 9. Juli um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

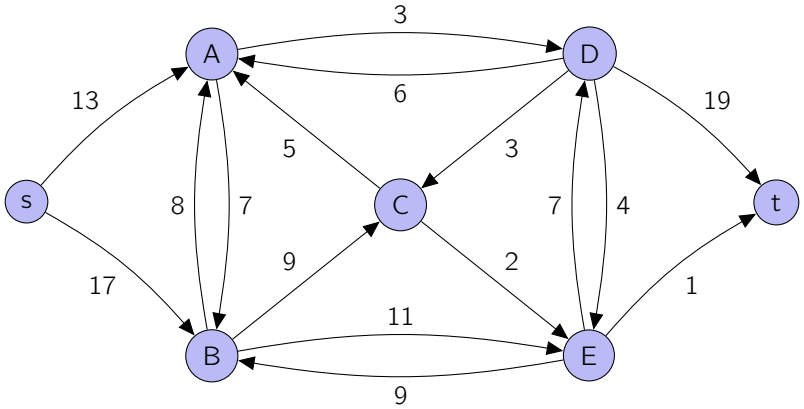
Aufgabe 1 (Floyd-Algorithmus): (8 Punkte)

Bestimmen Sie für den folgenden Graphen die Werte der kürzesten Pfade zwischen allen Paaren von Knoten. Nutzen Sie hierzu den Algorithmus von Floyd. Geben Sie das Distanzarray vor dem Aufruf, sowie nach jeder Iteration an.
Die Knotenreihenfolge sei mit A, B, C, D, E fest vorgegeben.



Aufgabe 2 (Maximaler Fluss): (6 Punkte)

Verwenden Sie die Ford-Fulkerson-Methode, um den maximalen Fluss des folgenden Flussnetzwerkes mit den gegebenen Kapazitäten zu berechnen. Geben Sie nach jeder Flusserhöhung das Flussnetzwerk an und kennzeichnen Sie den von Ihnen gewählten augmentierenden Pfad. Geben Sie ebenfalls das Restnetzwerk nach der ersten, zweiten und letzten Flusserhöhung an.



Aufgabe 3 (Eigenschaften von Flüssen): (2 + 3 + 4 + 2 = 11 Punkte)

Beweisen Sie die folgenden Aussagen über einen Fluss f und ein Flussnetzwerk $G(V, E, c)$ mit Quelle s und Senke t:

a) Für alle $X, Y \subseteq V$ gilt:

$$f(X, Y) = -f(Y, X)$$

b) Für alle $X, Y, Z \subseteq V$ mit $X \cap Y = \emptyset$ gilt:

$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$

c) Sei (S,T) ein Schnitt über G, so gilt:

$$f(S, T) = |f|$$

d) Sei (S,T) ein Schnitt über G, so gilt:

$$|f| \leq c(S, T)$$

Aufgabe 4 (Ausgabe des kürzesten Pfades): (5 + 3 = 8 Punkte)

a) Erweitern Sie die in der Vorlesung gegebene Implementierung von Floyd (Vorlesung 18, Folie 21), sodass jeweils die zugehörigen Vorgänger in einem zusätzlichen Array `int &V[][]` abgelegt werden (vgl. Folie 22). Ihre Methode soll die folgende Signatur besitzen:

```
void floydSP(double W[][], int n, double &D[][], int &V[][])
```

b) Geben Sie eine Implementierung der Methode `void pfadAusgeben(int &V[][], int start, int ziel)` an, die auf Basis des von der Methode aus **a)** berechneten Vorgängerarrays den kürzesten Pfad zwischen `start`- und `ziel`-Knoten ausgibt.

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis **Freitag, den 13. Juli um 14:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können die Lösungen auch zu Beginn der zugehörigen Globalübung im Audimax abgegeben werden.
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Dynamische Programmierung): (4 + 6 + 5 = 15 Punkte)

a) Schreiben Sie eine Funktion, welche die Länge der Longest Common Subsequence (LCS) für zwei gegebene Zeichensequenzen berechnet. Die Zeichensequenzen seien dabei der Einfachheit halber `int` Arrays `seq1` und `seq2` mit den Längen `l1` and `l2`. Demnach soll Ihre Funktion die folgende Signatur haben:

```
int lcs(int seq1[], int l1, int seq2[], int l2)
```

b) Bestimmen Sie gemäß dem in der Vorlesung vorgestellten Verfahren die LCS der Wörter BACHELOR und AACHEN. Geben Sie hierzu die berechnete Matrix an und kennzeichnen Sie den Pfad, der zur Rekonstruktion des Ergebnisses genutzt wird.

c) Betrachten Sie folgendes Szenario. Ihnen wird eine Klausur gestellt, welche n Aufgaben umfasst. Jede dieser Aufgaben hat eine Punktzahl p_i und eine von Ihnen geschätzte Zeit t_i, die Sie zum Lösen der Aufgabe benötigen ($1 \leq i \leq n$). Geben Sie die maximale Punktzahl, welche Sie in T Zeiteinheiten erreichen können, als Rekursionsgleichung an (inklusive der passenden Argumente).

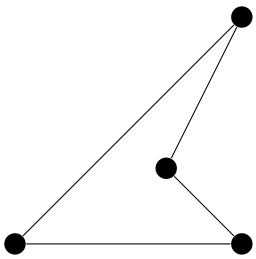
Aufgabe 2 (Geometrie): (8 + 7 = 15 Punkte)

a) Gegeben seien die folgenden Punkte:

(13, 7), (3, 8), (5, 2), (9, 4), (6, 11), (12, 14), (7, 13), (10, 8), (17, 14), (18, 5)

Bestimmen Sie mit Hilfe des Graham-Scans die konvexe Hülle der oben angegebenen Punkte. Geben Sie alle berechneten Determinanten und den jeweils aktuellen Zustand des Stacks an.

b) Schreiben Sie eine Funktion, die zu einem einfachen (aber nicht notwendigerweise konvexen) Polygon und einem Punkt einen Wahrheitswert zurück liefert, der angibt, ob sich der Punkt innerhalb des Polygons befindet (Punkte auf dem Rand des Polygons sind dabei innerhalb des Polygons). Hierbei können Sie davon ausgehen, dass eine Datenstruktur `P` existiert, die zwei ganzzahlige Felder `x` und `y` besitzt. Diese Datenstruktur modelliert Punkte im zweidimensionalen Raum. Solche Punkte können mittels des Vergleichsoperators == auf Gleichheit getestet werden. Das Polygon sei für die Eingabe als Array von Punkten gegeben, wobei jeweils aufeinander folgende Punkte sowie der letzte und erste Punkt miteinander verbunden sind. Als Beispiel ist nachfolgend ein Polygon und seine Repräsentation als Array von Punkten angegeben.



```
polygon[0].x = 0, polygon[0].y = 0
polygon[1].x = 3, polygon[1].y = 0
polygon[2].x = 2, polygon[2].y = 1
polygon[3].x = 3, polygon[3].y = 4
```

Ihre Funktion soll folgende Signatur haben, wobei `n` die Anzahl der Punkte ist, aus denen das Polygon besteht:

```
boolean inPolygon(P point, P polygon[], int n)
```

Sie dürfen davon ausgehen, dass eine Funktion `float winkel(P p1, P p2, P p3)` existiert, welche den Winkel zwischen den Strecken `p2p1` und `p2p3` berechnet. Sie liefert Fließkommazahlen w mit $-180 < w \leq 180$ zurück. Außerdem dürfen Sie annehmen, dass keine Rundungsfehler bei Berechnungen mit Fließkommazahlen auftreten.

Hinweis: Eine mögliche Lösung arbeitet ähnlich wie der Graham-Scan. Überlegen Sie sich, wie Sie Winkel zwischen bestimmten Strecken dazu nutzen können, das Problem zu lösen.

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sollen in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden.
- Die Lösungen müssen bis Montag, den 16. April um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Reihen): (2+2 Punkte)

Zeigen Sie, dass die folgenden Aussagen für beliebige $n \in \mathbb{N}^{>0}$ gelten:

a) $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ **b)** $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Aufgabe 2 (Laufzeit): (1+3+4 Punkte)

Gegeben sei der folgende Algorithmus zur Suche von Duplikaten in einem Array:

```
bool duplicate(int E[], int n) {
  for(int i = 0; i < n; i++)
    for(int j = i+1; j < n; j++)
      if(E[i] == E[j])
        return true;

  return false;
}
```

Er erhält ein Array E mit n Einträgen, für das er überprüft, ob zwei Einträge des Arrays denselben Wert besitzen.

Für die Average-Case Analyse gehen wir von folgenden Voraussetzungen aus:

- Mit einer Wahrscheinlichkeit von 0.3 gibt es im Array ein Duplikat.
- Es gibt immer maximal ein Duplikat.
- Wenn ein Wert doppelt enthalten ist, so steht dieser Wert an der ersten Position im Array.
- Das zweite Auftauchen des Wertes ist an jeder (anderen) Position gleich wahrscheinlich.

Bestimmen Sie in Abhängigkeit von n ...

a) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Best-Case.

b) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Worst-Case.

c) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Average-Case.

Aufgabe 3 (Ungleichungen): (2+2+2+2 Punkte)

Zeigen Sie die folgenden Aussagen für beliebige $n \in \mathbb{N}^{>0}$:

a) $n! \leq n^n$ **b)** $\sum_{i=1}^{n}(4 \cdot i + 3) \leq 4(n^2 + n)$ **c)** $\sum_{i=1}^{n} \log_2 i \leq \log_2 n^n$ **d)** $\log_2 n \leq 3 \cdot \log_4 n$

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sollen in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden.
- Die Lösungen müssen bis Montag, den 23. April um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Baumeigenschaften): (2+2+2 Punkte)

Beweisen Sie folgende in der Vorlesung eingeführte Fakten für Binärbäume:

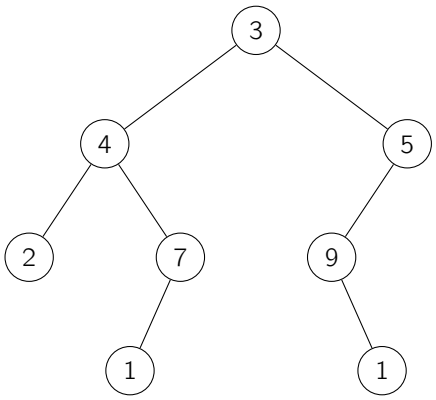
a) Ein Binärbaum enthält höchstens 2^d Knoten in Ebene d.

b) Ein Binärbaum mit Höhe h kann maximal $2^{h+1} - 1$ Knoten enthalten.

c) Ein Binärbaum mit n Knoten hat mindestens die Höhe $\lceil \log_2(n+1) \rceil - 1$.

Aufgabe 2 (Baumtraversierung): (1+2+2 Punkte)

a) Geben Sie jeweils das Ergebnis der in-, pre- und post-order Traversierung des folgenden Baumes an:



b) Bestimmen Sie zu den folgenden Paaren von Linearisierungen den jeweils zugehörigen Baum:

(i) in-order: 5 8 4 7 3 1 6 9 2
pre-order: 3 4 5 8 7 2 6 1 9

(ii) in-order: 5 1 2 4 6 7 3 9 8
post-order: 5 2 4 1 3 8 9 7 6

c) Geben Sie ein minimales Beispiel für zwei unterschiedliche Bäume an, die sowohl die gleiche pre- als auch post-order Linearisierung besitzen. Minimal bedeutet hier mit der kleinstmöglichen Anzahl von Elementen wobei jedes Element maximal einmal pro Baum enthalten sein darf.

Aufgabe 3 ($\mathcal{O}$-Notation): (2+2+2+3+2 Punkte)

Zeigen oder widerlegen Sie die folgenden Aussagen:

a) $g(n) = 2n^3 + 142n^2 + 462 \in \Theta(n^3)$
b) $2^{n+1} \in \Theta(2^n)$
c) $\log n \in \mathcal{O}(\sqrt{n})$
d) $max(f(n),\ g(n)) \in \Theta(f(n) + g(n))$
e) $g(n) + f(n) \in \mathcal{O}(g(f(n)))$

Aufgabe 4 (Beweise): (3+3 Punkte)

Zeigen oder widerlegen Sie die folgenden Aussagen:

a) $o(f(n)) \cap \omega(f(n)) = \emptyset$
b) Aus $f(n) \in \Omega(g(n))$ und $g(n) \in \Omega(h(n))$ folgt $f \in \Omega(h(n))$

Aufgabe 5 (Laufzeitanalyse): (3 Punkte)

Bestimmen Sie die Komplexitätsklasse für den Aufruf `berechne(n)` in Abhängigkeit von n. Gehen Sie davon aus, dass die Grundrechenarten +, -, *, / in konstanter Zeit O(1) ausgeführt werden, ebenso die Zuweisungen = und Vergleiche >.

```
int calculate(int value){

    int result = value;

    for(int i = value; i > 0; i = i/2){
        result = result * result;
        for(int j = i; j > 0; j--){
            result = 2 * result;
        }
    }

    return result;
}
```

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sollen in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden.
- Die Lösungen müssen bis Montag, den 30. April um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die **Nummer der Übungsgruppe** sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!
- Bitte beachten Sie, dass am 1. Mai keine Übungen stattfinden. Die Ausweichtermine finden Sie auf unserer Webseite.

Aufgabe 1 (Lineare Suche): (5 Punkte)

Geben Sie einen Algorithmus an, der in $\mathcal{O}(n)$ für eine Folge von n ganzen Zahlen (gegeben als Array) eine maximale Teilfolge findet. Eine Teilfolgen wird hierbei von beliebig vielen (maximal n) *aufeinanderfolgenden* Array Einträge gebildet. Sie ist maximal, wenn die Summe ihrer Elemente maximal ist, d.h. wir suchen aus allen möglichen Teilfolgen eine mit maximaler Summe.
Die Teilfolge soll dabei als *Startindex*, *Endindex* sowie *Summe* der Folge ausgegeben werden. Die Eingangsfolge

$$12,\ -34,\ 56,\ -5,\ -6,\ 78,\ -32,\ 8$$

liefert beispielsweise die Indizes 3 und 6 sowie die Summe $56 - 5 - 6 + 78 = 123$.

Aufgabe 2 (Rekursionsgleichungen): (3+4 Punkte)

a) Zeigen Sie mit Hilfe der Substitutionsmethode, dass für $T(n) = 2 \cdot T(\frac{n}{3}) + T(\frac{n}{4}) + 4n + 3$ mit $T(0) = -\frac{3}{2}$ gilt, dass $T(n) = 48n - \frac{3}{2}$

b) Nutzen Sie die Methode der Variablentransformation um die exakte Lösung der Rekursionsgleichung $T(n) = T(\sqrt[3]{n}) + 3$ mit $T(2) = 2$ zu bestimmen.

Aufgabe 3 (Rekursionsbäume): (5+4 Punkte)

Erstellen Sie zu den folgenden Rekursionsgleichungen die entsprechenden Rekursionsbäume und lesen Sie eine möglichst kleine obere Schranke der Lösung ab. Überprüfen Sie die Schranke mit Hilfe des Substitutionsverfahrens.

a)

$$T(n) = 4 \cdot T(\frac{n}{3}) + \frac{n}{2} + 3 \text{ mit } T(0) = 1$$

b)

$$T(n) = T(n-2) + T(n-1) + 7 \text{ mit } T(0) = 1 \text{ und } T(1) = 7$$

Aufgabe 4 (Analyse rekursiven Codes): (2+2+2+2 Punkte)

Geben Sie für die folgenden Programme die Laufzeitkomplexität als Rekursionsgleichung in Abhängigkeit des Eingabeparameters n an:

a)

```
int rekursion1(int n){
    if(n == 0)
        return 1;

    int sum = 1;
    while(sum < n){
        sum = sum * 2;
    }

    return n * rekursion1(n/2) + sum
}
```

b)

```
int rekursion2(int n){
    if(n <= 0)
        return n*n;

    int wert = rekursion2(n-1) * (rekursion2(n-1) + 2);
    n--;

    for(int i = 0; i < n; i++){
        for(int j = n; j > 0; j--){
            sum += i * j;
        }
    }
    return wert * rekursion2(n);
}
```

c)

```
int rekursion3(int n){
    if(n/2 <= 0){
        int value = 3;
        for(int i = 0; i < n; i++)
            value *= value;
        return value;
    }else
        return rekursion3(n-1) * rekursion3(n-2) * rekursion3(n-4);
}
```

d)

```
int rekursion4(int n){
    if(n <= 0)
        return foo(5);
    return rekursion4(n/3) * foo(n*n);
}

int foo(n){
    int k = 1;
    for(int i = 0; i < n; i++)
        k *= 2;
    for(int i = 0; i < k; i++)
        n *= n;
    return n;
}
```

Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Hinweise:

- Die Übungsblätter sind in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung zu bearbeiten.
- Die Lösungen müssen bis Montag, den 7. Mai um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!
- Für die Übungsgruppen 7,8,9 und 10 bieten wir in den nächsten Wochen Ersatztermine an. Eine Liste der Ausweichtermine befindet sich auf unserer Webseite.
- Am Freitag den 11. Mai, sowie am Dienstag den 15. findet keine Vorlesung statt.

Aufgabe 1 (Sortieralgorithmus I): (2 + 1 + 2 + 2 + 3 Punkte)

Gegeben sei der folgende Sortieralgorithmus:

```
void sort(int E[]) {
  int i,j,m;
  for (i = 0; i < E.length; i++) {
    m = i;
    for (j = i + 1; j < E.length; j++) {
      if (E[j] <= E[m]) {
          m = j;
      }
    }
    int v = E[i];
    E[i] = E[m];
    E[m] = v;
  }
}
```

a) Nutzen Sie den gegebenen Algorithmus, um das folgende Array zu sortieren. Geben Sie den Zustand des Arrays nach jedem Durchlauf der äußeren Schleife an.

1	3	2	7	0	4	8	5	7	6

b) Geben Sie in wenigen Worten wieder, wie der gegebene Algorithmus funktioniert. In der Vorlesung wurden mehrere Sortierverfahren genannt (siehe Folien). Welcher Name passt zu diesem Algorithmus?

c) Ist der Sortieralgorithmus stabil? Falls nicht geben Sie an wie er angepasst werden muss, damit er stabil wird?

d) Welche Average-Case Laufzeit besitzt der gegebene Sortieralgorithmus für eine Eingabe der Länge n? Geben Sie die Komplexitätsklasse $\Theta(T_{sort}(n))$ für ein Array E mit Länge `n = E.length` an und begründen Sie Ihre Antwort.

e) Implementieren Sie eine Version des obigen Sortieralgorithmus, der die Elemente einer doppelt verketten Liste aufsteigend nach ihren Schlüsseln sortiert. Die Signatur der Sortiermethode sei `void sort(List l)`. Die Klassendefinition eines Listenelements ist unten gegeben. Gehen Sie davon aus, dass der Algorithmus das erste Element einer nicht zyklischen, doppelt verketteten Liste erhält, d.h. der `prev`-Zeiger des ersten Elements ist ein Nullpointer, ebenso der `next`-Zeiger des letzten Elements. Achten Sie darauf tatsächlich die Reihenfolge der Elemente zu ändern und nicht lediglich die Schlüsselwerte zu vertauschen.

```
class List{
    int key;
    List next, prev;
}
```

Aufgabe 2 (Sortieralgorithmus II): (3 + 3 + 3 Punkte)

Betrachten Sie den folgenden Sortieralgorithmus.

```
static void sort(int E[]) {
  int sorted = 0;
  while(sorted < E.length - 1) {
    int pos = 0;
    for(int i = sorted + 1; i < E.length; i++){
      if(E[i] <= E[sorted]){
        pos++;
      }
    }
    if (pos == 0){
      sorted++;
    }else{
      swap(E, sorted, sorted + pos);
    }
  }
}

static void swap(int E[], int i, int j){
  int tmp = E[i];
  E[i] = E[j];
  E[j] = tmp;
}
```

a) Nutzen Sie den gegebenen Algorithmus, um das folgende Array zu sortieren. Geben Sie den Zustand des Arrays nach jedem Durchlauf der `while`-Schleife an.

4	1	7	9	0	6	2	3	5	8

b) Geben Sie die Best-Case sowie Worst-Case Laufzeit des gegebenen Sortieralgorithmus an. Geben Sie hierzu jeweils die Komplexitätsklasse $\Theta(T_{sort}(n))$ für ein Array E mit Länge `n = E.length` an und begründen Sie Ihre Antwort.

c) Ergibt sich eine bessere Worst-Case Laufzeit wenn der Vergleich `E[i] <= E[sorted]` durch den Vergleich `E[i] < E[sorted]` ersetzt wird?

Aufgabe 3 (Rekursionsgleichungen): (2 + 2 + 4 + 3 Punkte)

Geben Sie für die folgenden Rekursionsgleichungen die Komplexitätsklasse (Θ) an. Begründen Sie Ihre Antwort.

a) $T(n) = 23T(\frac{n}{23}) + 42n$

b) $T(n) = 9T(\frac{n}{3}) + 27n$

c) $T(n) = 2T(\frac{n}{2}) + n\log_2 n$

d) $T(n) = 2T(\frac{n}{2}) + T(\frac{n}{3}) + 2n^2 + 5n + 42$