Prof. Dr. E. Ábrahám
F. Corzilius, S. Schupp, T. Ströder

2. Klausur Datenstrukturen und Algorithmen SS 2014

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte **genau** einen markieren):

- ○ Informatik Bachelor
- ○ Informatik Lehramt (Bachelor)
- ○ Sonstiges: _____
- ○ Mathematik Bachelor
- ○ CES Bachelor

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**19**	
Aufgabe 2	**13**	
Aufgabe 3	**24**	
Aufgabe 4	**6**	
Aufgabe 5	**30**	
Aufgabe 6	**16**	
Aufgabe 7	**12**	
Summe	**120**	

Allgemeine Hinweise:

- Schreiben Sie **auf alle Blätter** (inklusive zusätzliche Blätter) **Ihren Vornamen, Ihren Nachnamen und Ihre Matrikelnummer**.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Beantworten Sie die Aufgaben auf den ausgeteilten Aufgabenblättern.
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Klausur mit **0 Punkten** bewertet.
- Geben Sie am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

Name: **Matrikelnummer:**

Aufgabe 1 (Laufzeitanalyse): (4 + 6 + 6 + 3 = 19 Punkte)

a) Beweisen oder widerlegen Sie:

$$\log(n) \in \mathcal{O}(\sqrt{n})$$

Falls allgemeine Rechenregeln benutzt werden, müssen diese beim Namen genannt oder formuliert werden.

b) Beweisen oder widerlegen Sie:

$$\sum_{i=1}^{n}(2 \cdot i - 1) \in \Theta(n^2)$$

Falls allgemeine Rechenregeln benutzt werden, müssen diese beim Namen genannt oder formuliert werden.

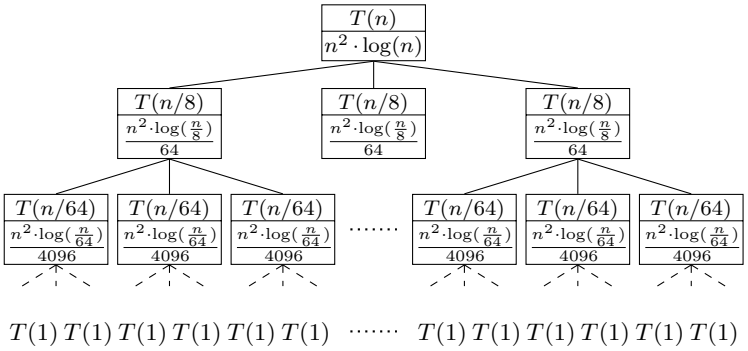
c) Bestimmen Sie die Komplexitätsklasse der folgenden Rekursionsgleichung mithilfe des Mastertheorems oder begründen Sie, warum das Mastertheorem nicht anwendbar ist.

$$T(n) = \begin{cases} 3 \cdot T(\frac{n}{27}) + \sqrt[3]{n+1} & \text{für } n > 1 \\ 5 & \text{sonst} \end{cases}$$

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

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

d) Gegeben sei der folgende Rekursionsbaum:



Ergänzen Sie die folgenden Aussagen:

- Der Rekursionsbaum hat die Tiefe _____ .
- Der Rekursionsbaum hat _____ Blätter.
- Auf der Ebene i summiert sich die Laufzeit zu _____ .

Name: **Matrikelnummer:**

Aufgabe 2 (Sortierverfahren): (3 + 3 + 4 + 3 = 13 Punkte)

a) Beweisen Sie, dass der Selectionsort-Algorithmus **aus der Vorlesung** nicht stabil ist.

b) Sortieren Sie das Array `[ 5,1,7,3,2 ]` durch Anwendung des Bubblesort-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

5	1	7	3	2

c) Sortieren Sie das Array `[ 7,6,5,1,4,2,8,3 ]` durch Anwendung des Mergesort-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array nach jeder Merge-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

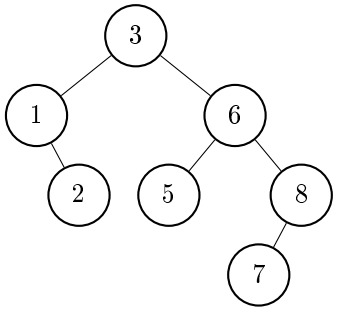
7	6	5	1	4	2	8	3

d) Stellen Sie die Heap-Eigenschaft im Array `[ 2,7,9,3,6,5,1,4 ]` her, wie es zu Beginn des Heapsort-Algorithmus **aus der Vorlesung** geschieht. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

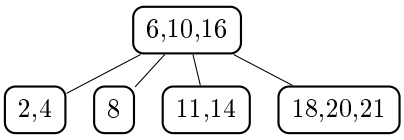
2	7	9	3	6	5	1	4

Aufgabe 3 (Bäume): (5 + 3 + 10 + 3 + 3 = 24 Punkte)

a) Löschen Sie den Wert 3 aus dem folgenden **AVL-Baum** und geben Sie die entstehenden Bäume nach jeder **Löschoperation** sowie jeder **Rotation** an:



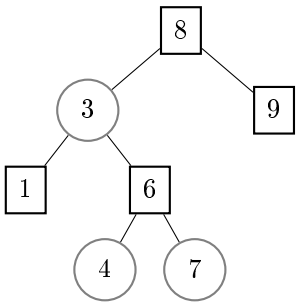
b) Fügen Sie den Wert 13 in den folgenden **2-3-4-Baum** ein und geben Sie den dabei entstehenden Baum an:



c) Fügen Sie den Wert 5 in den folgenden **Rot-Schwarz-Baum** ein und geben Sie die entstehenden Bäume nach

- jeder **Einfügeoperation**,
- jeder **Rotation** sowie
- jeder **Umfärbung** an.

Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



d) Geben Sie den kleinsten Baum (Anzahl Blätter minimal) an, der die **Heap-Eigenschaft** erfüllt und die Schlüssel $1, 2, 4, 4, 5, 6, 6, 6$ enthält.

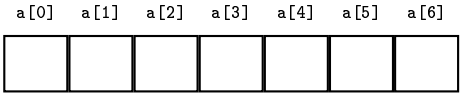
e) Geben Sie eine **Einfügesequenz** der 5 Schlüssel von 0 bis 4 an, sodass der resultierende binäre Suchbaum **maximal unbalanciert** ist.

Geben Sie eine **Einfügesequenz** der 7 Schlüssel von 0 bis 6 an, sodass der resultierende binäre Suchbaum **minimal unbalanciert** ist.

Aufgabe 4 (Hashing): (3 + 3 = 6 Punkte)

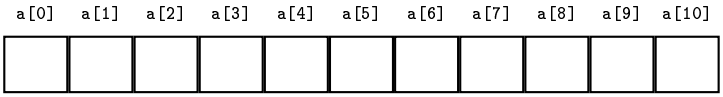
a) Fügen Sie die folgenden Werte in das unten stehende Array `a` der Länge 7 unter Verwendung der **Multiplikationsmethode** ($c = 0.25$) mit **linearer Sondierung** ein ($f(n, i) = \lfloor 7 \cdot (n \cdot 0.25 \mod 1) \rfloor + i \mod 7$), wobei $x \mod 1$ den Nachkommateil von x bezeichnet:

5, 7, 4, 11, 0, 12.



b) Fügen Sie die folgenden Werte in das unten stehende Array `a` der Länge 11 unter Verwendung der **Divisionsmethode** mit **quadratischer Sondierung** ($c_1 = 0.0$, $c_2 = 1.0$) ein ($f(n, i) = ((n \mod 11) + 0.0 \cdot i + 1.0 \cdot i^2) \mod 11$):

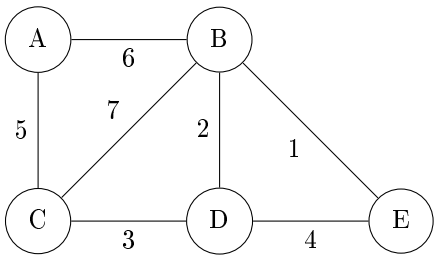
1, 8, 7, 12, 18, 22.



Aufgabe 5 (Graphen): (6 + 8 + 8 + 8 = 30 Punkte)

a) Beweisen oder widerlegen Sie: Der Dijkstra Algorithmus zur Berechnung kürzester Pfade arbeitet korrekt auf gerichteten Graphen, welche zwar negative Kantengewichte enthalten, aber bei denen jeder Zyklus ein positives Gesamtgewicht hat.

b) Führen Sie Prim's Algorithmus auf dem folgenden Graphen aus.



Der Startknoten hat hierbei den Schlüssel A. Geben Sie dazu **vor** jedem Durchlauf der äußeren Schleife an,

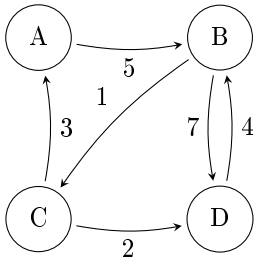
1) welche Kosten die Randknoten haben (d. h. für jeden Knoten `v` in `Q` den Wert `key[v]`)
2) und welchen Knoten `extractMin(Q)` wählt, indem Sie den Kosten-Wert des gewählten Randknoten in der Tabelle unterstreichen (wie es in der ersten Zeile bereits vorgegeben ist).

Geben Sie zudem den vom Algorithmus bestimmten minimalen Spannbaum an.

#Iteration	A	B	C	D	E
1	$\underline{0}$	∞	∞	∞	∞
2					
3					
4					
5					

Minimaler Spannbaum:

c) Betrachten Sie den folgenden Graphen:



Führen Sie den **Algorithmus von Floyd** auf diesem Graphen aus. Geben Sie dazu nach jedem Durchlauf der äußeren Schleife die aktuellen Entfernungen in einer Tabelle an. Die erste Tabelle enthält bereits die Adjazenzmatrix nach Bildung der reflexiven Hülle. Der Eintrag in der Zeile i und Spalte j ist also ∞, falls es keine Kante vom Knoten der Zeile i zu dem Knoten der Spalte j gibt, und sonst das Gewicht dieser Kante. Beachten Sie, dass in der reflexiven Hülle jeder Knoten eine Kante mit Gewicht 0 zu sich selbst hat.

(1)	A	B	C	D
A	0	5	∞	∞
B	∞	0	1	7
C	3	∞	0	2
D	∞	4	∞	0

(2)	A	B	C	D
A				
B				
C				
D				

(3)	A	B	C	D
A				
B				
C				
D				

(4)	A	B	C	D
A				
B				
C				
D				

(5)	A	B	C	D
A				
B				
C				
D				

d) Betrachten Sie die folgende Situation: Sie koordinieren die Platzvergabe für Volkshochschulkurse. Als Grundlage für die Vergabe haben Sie eine Menge von n Kursen $K = \{k_1, \ldots, k_n\}$ und eine Menge von m Personen $P = \{p_1, \ldots, p_m\}$. Jede Person kann an höchstens einem Kurs teilnehmen und Sie haben von allen Personen Angaben darüber, welche Kurse sie interessieren. Diese Angaben liegen in Form einer Funktion $I : P \times K \to \{0, 1\}$ vor, wobei der Funktionswert 1 angibt, dass sich die jeweilige Person für den jeweiligen Kurs interessiert. Außerdem gibt die Kapazitätsfunktion $C : K \to \mathbb{N}$ zu jedem Kurs an, wieviele Personen maximal am jeweiligen Kurs teilnehmen können.

Geben Sie ein Verfahren **basierend auf Verfahren aus der Vorlesung** an, das berechnet, welche Personen welchen Kursen zugeteilt werden sollten, damit so viele Personen wie möglich einen Kurs besuchen, ohne die Kapazitäten der Kurse zu überschreiten und ohne dass eine Person mehreren Kursen zugeteilt wird. Ihr Verfahren muss eine Laufzeitkomplexität in $\mathcal{O}((n + m)^5)$ haben (Sie brauchen nicht zu beweisen, dass diese Laufzeitschranke eingehalten wird).

Aufgabe 6 (Dynamische Programmierung): (6 + 10 = 16 Punkte)

a) Gegeben sei ein Array `a` der Länge `n`, wobei die Einträge im Array `a[0]`, ..., `a[n-1]` ganzzahlig sind. Bestimmen Sie eine **Rekursionsgleichung** $L(i)$, die die **Länge der längsten aufsteigenden Teilsequenz** des Arrays `a` bestimmt, die im i-ten Arrayelement endet, wobei $0 \leq i < n$. Wie kann man mit dieser Rekursionsgleichung die Länge der längsten aufsteigenden Teilsequenz des Arrays `a` bestimmen?

Beispiel: Die längsten aufsteigenden Teilsequenzen des Arrays `[6, 14, 1, 9, 5, 13]` sind `(6, 9, 13)`, `(1, 9, 13)` und `(1, 5, 13)`, wobei $L(0) = L(2) = 1$, $L(1) = L(3) = L(4) = 2$ und $L(5) = 3$.

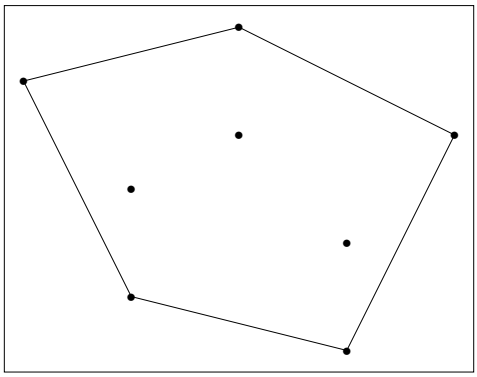
b) Bestimmen Sie die **längste gemeinsame Teilsequenz** der Sequenzen `DOSTAL` und `EDISAFL`. Benutzen Sie hierfür den in der Vorlesung vorgestellten Algorithmus mit dynamischer Programmierung und füllen Sie die folgende Tabelle aus. Beschreiben Sie wie man anhand der Tabelle die längste gemeinsame Teilsequenz der gegebenen Wörter und die Länge dieser Teilsequenz bestimmen kann.

	∅	E	D	I	S	A	F	L
∅								
D								
O								
S								
T								
A								
L								

Aufgabe 7 (Geometrische Algorithmen): (2 + 2 + 3 + 5 = 12 Punkte)

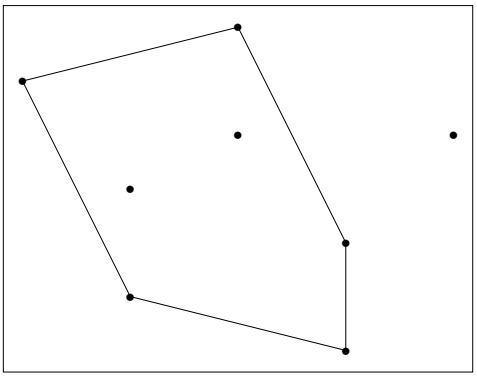
a) Geben Sie die Definition der konvexen Hülle einer endlichen Menge von Punkten an.

b) Welche der folgenden Strukturen ist eine konvexe Hülle der eingezeichneten Punktmenge? Begründen Sie ihre Antwort kurz, wenn keine konvexe Hülle vorliegt.



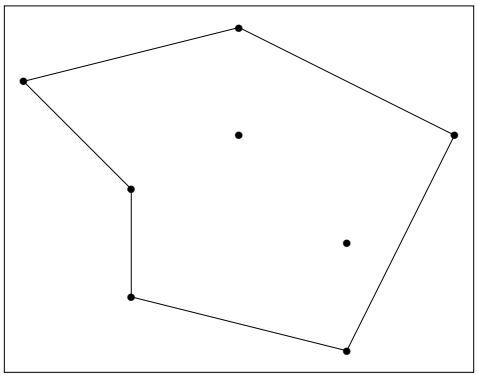
☐ Ja

☐ Nein, weil:



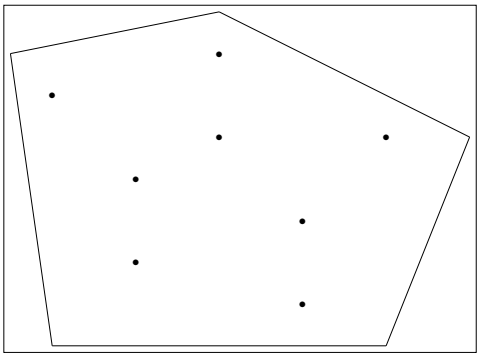
☐ Ja

☐ Nein, weil:



☐ Ja

☐ Nein, weil:



☐ Ja

☐ Nein, weil:

c) Welche Komplexität hat der Grahamsche Scan (O-Notation), wenn ein Sortierverfahren mit der Laufzeit $\mathcal{O}(n \cdot \log n)$ verwendet wird? Sie dürfen dabei die Berechnung des Polarwinkels als konstant annehmen.

d) Für den Grahamschen Scan muss festgestellt werden, ob sich ein Punkt links oder rechts von einem gegebenen Liniensegment befindet. Berechnen Sie mit der **Determinantenmethode** für das Liniensegment zwischen den Punkten $(1,2)^T$ und $(5,3)^T$, ob sich die Punkte $P_1 = (2,2)^T$ und $P_2 = (4,1)^T$ jeweils rechts oder links davon befinden.