Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Tobias Pröger

Prüfung
Datenstrukturen und Algorithmen
D-INFK

24. Januar 2013

Name, Vorname: _____

Stud.-Nummer: _____

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die untenstehenden Hinweise gelesen und verstanden habe.

Unterschrift: _____

Hinweise:

- Ausser einem Wörterbuch dürfen Sie keine Hilfsmittel verwenden.
- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt. Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden.
- Bitte schreiben Sie **lesbar** mit blauer oder schwarzer Tinte. Wir werden nur bewerten, was wir lesen können.
- Sie dürfen alle Algorithmen und Datenstrukturen aus der Vorlesung verwenden, ohne sie noch einmal zu beschreiben. Wenn Sie sie modifizieren, reicht es, die Mofikationen zu beschreiben.
- Die Prüfung dauert 120 Minuten.

Viel Erfolg!

Stud.-Nummer: _____

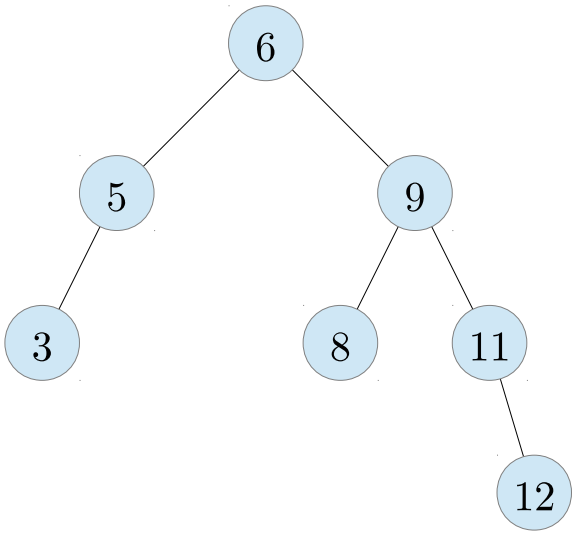
Aufgabe	1	2	3	4	5	**Σ**
Mögl. Punkte	8	7	9	9	10	43
$\sum$ Punkte						

Aufgabe 1.

Hinweise:

1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2) Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Datenstrukturen & Algorithmen" verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch andere Methoden benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
3) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

1 P (a) Fügen Sie in den untenstehenden AVL-Baum den Schlüssel 15 ein, und löschen Sie *danach* im entstandenen AVL-Baum den Schlüssel 3.



Nach Einfügen von 15:	Nach Löschen von 3:

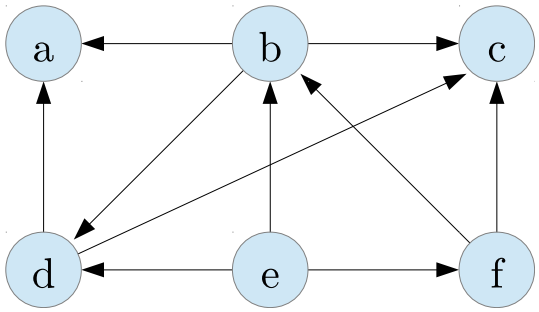
1 P (b) Im untenstehenden Array sind die Elemente eines Max-Heaps in der üblichen Form gespeichert. Wie sieht das Array aus, nachdem das Maximum entfernt wurde und die Heap-Bedingung wieder hergestellt wurde?

35	33	22	20	24	10	16	12	11	17	14	9	5
1	2	3	4	5	6	7	8	9	10	11	12	13

1 P (c) Fügen Sie die Schlüssel 16, 6, 24, 41, 18, 38, 28 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie offenes Hashing mit der Hashfunktion $h(k) = k \bmod 11$ und lösen Sie Kollisionen mittels quadratischem Sondieren auf.

0	1	2	3	4	5	6	7	8	9	10

1 P (d) Geben Sie eine topologische Sortierung des untenstehenden Graphen an.



Topologische Sortierung: _____, _____, _____, _____, _____, _____

1 P (e) Geben Sie einen zusammenhängenden Graphen mit 6 Knoten an, der **genau** 3 verschiedene perfekte Matchings besitzt.

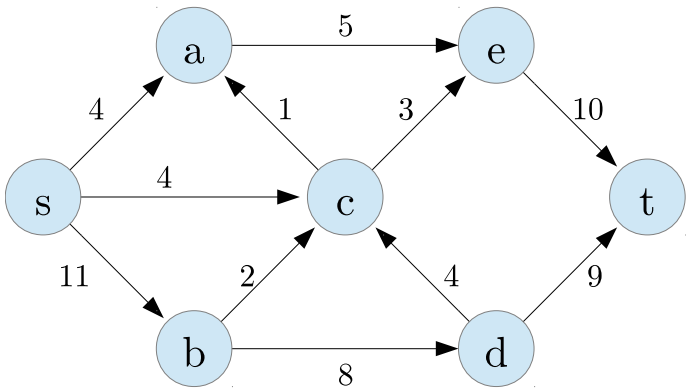
1 P (f) Wie viele Kanten kann ein bipartiter Graph mit n Knoten maximal besitzen? Sie dürfen der Einfachheit halber annehmen, dass n gerade ist.

1 P (g) Das untenstehende Array wird mit *Sortieren durch Einfügen* sortiert. Geben Sie die an den ersten vier Vergleichen beteiligten Schlüsselpaare an.

8	15	6	12	4	9	10	11	13	19
1	2	3	4	5	6	7	8	9	10

Schlüsselpaare: (____, ____), (____, ____), (____, ____), (____, ____)

1 P (h) Gegeben ist das folgende Netzwerk mit Quelle s und Senke t. Die einzelnen Kapazitäten sind durch die Zahlen neben den Kanten gegeben.



Ein Fluss ϕ ist partiell durch die folgende Funktion gegeben:

(v, w)	(s, a)	(s, c)	(b, d)	(c, e)	(d, t)
$\phi(v, w)$	4	4	8	3	8

Ergänzen Sie die Flusswerte auf den übrigen Kanten, sodass ϕ ein gültiger Fluss ist. Dieser wird in jedem Fall maximal sein. Zeichnen Sie in die obige Abbildung einen Schnitt ein, der zeigt, dass ϕ maximal ist.

(v, w)	(s, b)	(a, e)	(b, c)	(c, a)	(d, c)	(e, t)
$\phi(v, w)$						

Aufgabe 2.

1 P (a) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^7)$ und $n^7 \in \mathcal{O}(n^9)$ gilt.

- n^2
- $\binom{n}{4}$
- $\log(n^{17})$
- $(\log n)^5$
- $n!$
- $\sqrt{3^n}$
- 10^{50}

3 P (b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 8 + 5T(n/3) & n > 1 \\ 3 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und möglichst einfache Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion.

Hinweise:

(1) Sie können annehmen, dass n eine Potenz von 3 ist.

(2) Für $q \neq 1$ gilt: $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

1 P (c) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für den folgenden Algorithmus (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 0; i <= n; i ++) {
2         for(int j = 1; j <= n*n*n; j *= 2)
3                 ;
4         for(int k = 1; k <= n; k += 2)
5                 ;
6 }
```

1 P (d) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für den folgenden Algorithmus (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

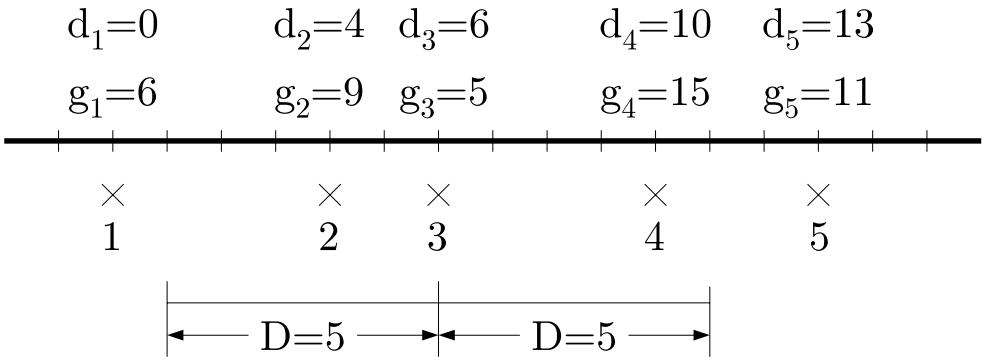
```
1 for(int i = 1; i <= ⌈log_5(n)⌉; i ++) {
2         int k = n;
3         while(k > 1)
4                 k = k/4;
5 }
```

1 P (e) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für den folgenden Algorithmus (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
1 int f(int n) {
2         if(n == 1) return 1;
3         else {
4                 for(int i = 1; i <= n; i ++)
5                         ;
6                 return f(n/3)+1;
7         }
8 }
```

Aufgabe 3.

Entlang einer Strasse sollen Windräder zur Stromgewinnung aufgestellt werden. Aufgrund geographischer Gegebenheiten kommen n verschiedene Positionen in Betracht, aber Gesetze schreiben vor, dass zwischen zwei Windrädern ein Mindestabstand von D eingehalten wird. Die möglichen Positionen $d_1, ..., d_n$ sind als Koordinaten auf einer Linie angegeben, wobei die linkeste Position den Wert 0 hat. Anders ausgedrückt: Der Abstand der i-ten Position zur erstmöglichen Position beträgt d_i, es gilt $d_1 = 0$ und für alle $i \in \{1, ..., n-1\}$ ist $d_i < d_{i+1}$. Wenn ein Windrad an der Position i aufgestellt wird, dann beträgt der Gewinn $g_i > 0$. Die Aufgabe besteht nun darin, eine Positionierung von Windrädern mit maximalem Gewinn zu finden.



Beispiel: In der obigen Abbildung ist eine Situation für $n = 5$ mögliche Positionen dargestellt. Wird beispielsweise ein Windrad an der Position 3 aufgestellt, dann können an den Positionen 2 und 4 keine Windräder mehr aufgestellt werden. Werden die Windräder an den Positionen 1, 3 und 5 aufgestellt, dann beträgt der Gewinn $6 + 5 + 11 = 22$. Dies ist aber nicht die optimale Lösung: Eine Installation auf den Positionen 2 und 4 besitzt einen Gewinn von $9 + 15 = 24$.

1 P (a) Zeigen Sie anhand eines (möglichst einfachen) Beispiels, dass die folgende Greedy-Strategie nicht notwenigerweise eine optimale Lösung liefert: *"Wähle so lange aus den möglichen Positionen diejenige mit maximalem Gewinn aus, bis keine weiteren Windräder mehr platziert werden können."*

5 P (b) Beschreiben Sie einen möglichst effizienten Algorithmus der *dynamischen Programmierung*, der den maximal erreichbaren Gewinn berechnet.

1 P (c) Geben Sie die Laufzeit Ihres Algorithmus an.

2 P (d) Beschreiben Sie im Detail, wie der obige Algorithmus abgeändert werden muss, um eine Positionierung von Windrädern mit maximalem Gewinn zu berechnen.

Aufgabe 4.

An einer Universität gibt es n Studenten, von denen jeder im nächsten Semester genau 5 Kurse besuchen muss. Insgesamt werden m Kurse angeboten, wobei am Kurs k maximal $T_k \in \mathbb{N}$ Studenten teilnehmen können.

Jeder Student i ist an einer Menge K_i von 10 Kursen interessiert. Die Universität möchte nun die Studenten so auf die Kurse verteilen, dass jeder Student nur Kurse besucht, die ihn interessieren, und dass kein Kurs k existiert, den mehr als T_k Studenten besuchen.

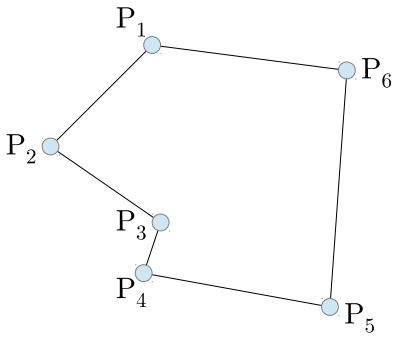
4 P (a) Es soll zunächst entschieden werden, ob eine solche Verteilung der Studenten auf die Kurse überhaupt existiert. Modellieren Sie dieses Problem als Flussproblem. Erläutern Sie genau, wie aus $K_1, ..., K_n$ und $T_1, ..., T_m$ ein geeignetes Netzwerk $N = (V, E)$ konstruiert werden kann. Geben Sie die Knoten V und die Kanten E der Konstruktion an und beschreiben Sie, welche Kapazitäten den Kanten zugewiesen werden müssen. Überlegen Sie, wie Sie aus dem Wert eines maximalen Flusses schlussfolgern können, ob eine geeignete Verteilung existiert oder nicht.

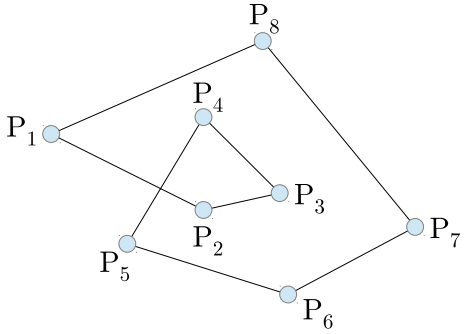
2 P (b) Nennen Sie einen Flussalgorithmus, der das Problem aus (a) möglichst effizient löst. Nehmen Sie an, dass es mehr Studenten als Kurse gibt, und geben Sie die Laufzeit des genannten Verfahrens in Abhängigkeit von n und m an.

3 P (c) Wenn wir davon ausgehen, dass eine geeignete Verteilung der Studenten auf die Kurse existiert, dann interessieren wir uns natürlich auch dafür, wie diese aussieht. Beschreiben Sie einen Algorithmus, der für jeden Studenten i die Menge der 5 Kurse berechnet, die i im nächsten Semester belegen muss. Welche Laufzeit besitzt Ihr Verfahren, wenn ein maximaler Fluss im Netzwerk N bereits vorab berechnet wurde?

Aufgabe 5.

Es seien P_1, ..., P_n eine Menge von Punkten in $\mathbb{R}^2$. Wenn für alle $i \in \{1, ..., n-1\}$ jeweils die Punkte P_i mit P_{i+1} und zusätzlich P_n mit P_1 durch ein Liniensegment verbunden werden, dann entsteht ein *Polygon*.





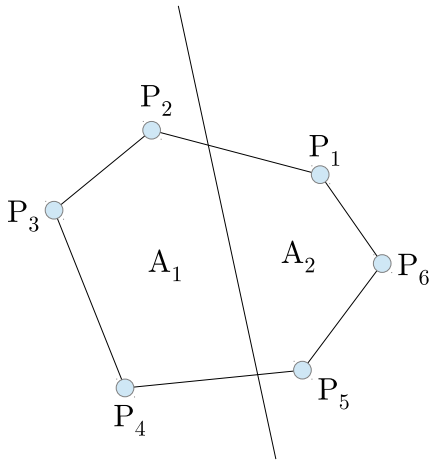
4 P (a) Ein Polygon heisst *einfach*, wenn sich keine zwei Kanten eines Polygons kreuzen. In der obigen Abbildung ist nur das linke Polygon einfach. Das rechte Polygon ist nicht einfach, da sich die Segmente $\overline{P_1P_2}$ und $\overline{P_4P_5}$ kreuzen.

Geben Sie einen Algorithmus an, der nach dem Scanline-Prinzip arbeitet und in Zeit $\mathcal{O}(n \log n)$ testet, ob ein Polygon einfach ist.

4 P (b) Wir nehmen nun an, dass P_1, ..., P_n ein konvexes Polygon definiert (Zur Erinnerung: Ein Polygon P heisst *konvex*, wenn die gesamte Verbindungsstrecke zwischen zwei beliebigen Punkten in P ebenfalls in P liegt).

Geben Sie einen möglichst effizienten Algorithmus an, der den Flächeninhalt des durch P_1, ..., P_n definierten konvexen Polygons berechnet. Welche Laufzeit besitzt Ihre Lösung?

2 P (c) Wie in (b) sei das durch P_1, ..., P_n definierte Polygon konvex. Zusätzlich sind noch die Parameter m und b einer Geraden $y = mx + b$ gegeben, die das Polygon in zwei Teile A_1 und A_2 teilt.



Beschreiben Sie einen effizienten Algorithmus, der die Resultate aus dem Aufgabenteil (b) benutzt, um die Flächeninhalte der Teile A_1 und A_2 zu berechnen. Geben Sie auch die Laufzeit Ihres Verfahrens an.