Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Übung 3

Aufgabe 4 (Laufzeitanalyse): 10 Punkte

Betrachten Sie den folgenden Algorithmus, welcher für ein Array `a` der Länge n mit Zahlen zwischen 0 und $k-1$ zurückgibt, ob in `a` eine Zahl mehrfach enthalten ist.

```
1 seen = [False] * k
2 for i in a
3   if seen[i]:
4     return True
5   else:
6     seen[i] = True
7 return False
```

Hinweis:

- `[False]*k` erzeugt ein Array der Länge k in dem alle Einträge `False` sind.

Bestimmen Sie die Best-Case, Worst-Case, und Average-Case Laufzeit unter der Annahme, dass

- `a` genau die Einträge 0 bis $k-1$ sowie eine zusätzliche 0 enthält (`a` hat also die Länge $n = k+1$) und
- jede Reihenfolge gleich wahrscheinlich ist.

Zur Einfachheit nehmen wir an, dass das Prüfen der `if`-Bedingung in Zeile 3 genau c Zeiteinheiten benötigt (für eine Konstante $c > 0$) und alle weiteren Operationen keine Zeit benötigen. Begründen Sie Ihre Antworten kurz.

Hinweise:

- Das Tauschen der beiden 0en ändert die Reihenfolge der Liste nicht.
- Ihre Lösung beim Average-case darf Summenzeichen enthalten.

Aufgabe 5 (Rekursionsgleichung): 10 Punkte

Finden Sie für die Rekursionsgleichung F, die wir als

$$F(1) = 1 \text{ und } F(n) = 2 \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \text{ für } n > 1$$

definieren, eine Funktion $g : \mathbf{R}_+ \to \mathbf{R}_+$ als obere Abschätzung an, welche nicht rekursiv definiert ist. Beweisen Sie anschließend, dass ihr Vorschlag tatsächlich eine obere Abschätzung ist, also dass $F(n) \leq g(n)$ ist. Geben Sie schließlich auch eine Abschätzung der Funktion g in O-Notation an.

Hinweise:

- $\lfloor\ \rfloor$ ist die untere Gaußklammer, d.h. falls der Wert nicht natürlich ist, runden wir ihn nach unten ab.

Aufgabe 6 (Master Theorem): 2 + 2 + 2 + 2 + 2 = 10 Punkte

Benutzen Sie das Master Theorem um für folgende Rekursionsgleichungen $T(n)$ die beste O-Klasse anzugeben, in der $T(n)$ liegt. Anbei haben wir Ihnen erneut das Master Theorem angegeben:
Für $T(1) = c$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ falls $n > 1$ gilt

Wenn	Dann
$f \in O(n^p)$ und $a < b^p$	$T(n) \in O(n^p)$
$f \in O(n^p)$ und $a = b^p$	$T(n) \in O(n^p \cdot \log(n))$
$f \in O(n^p)$ und $a > b^p$	$T(n) \in O(n^{\log_b(a)})$

a)

$T(1) = 2$

$T(n) = 2 \cdot T(\frac{n}{4}) + \log_2(n) \qquad \text{for } n > 1$

b)

$T(1) = 5$

$T(n) = 8 \cdot T(\frac{n}{2}) + 3 \cdot n^2 + 5 \cdot n \qquad \text{for } n > 1$

c)

$T(1) = 10$

$T(n) = 9 \cdot T(\frac{n}{3}) + 2 \cdot n^5 + \log_3(n) \qquad \text{for } n > 1$

d)

$T(1) = 7$

$T(n) = 9 \cdot T(\frac{n}{3}) + n \qquad \text{for } n > 1$

e)

$T(1) = 1$

$T(n) = 16 \cdot T(\frac{n}{4}) + 5 \cdot n^2 + \log_3(n) \qquad \text{for } n > 1$

Aufgabe 7 (Greedy): 10 Punkte

Während einer Klausur sollen die Prüflinge verschiedene Aufgaben bekommen, um das Risiko von Täuschungen zu vermindern. Dabei dürfen jedoch nur benachbarte Prüflinge nicht die gleiche Klausur bekommen. Ein ungerichteter Graph $G = (V, E)$ gibt an, welche Prüflinge benachbarte Plätze haben. Dabei sind V die Prüflinge und die Kanten $(i, j), (j, i) \in E$ geben an, dass die Prüflinge i und j benachbart sind.
Geben Sie einen Greedy Algorithmus an, der eine Abbildung von Prüflingen auf Klausuren berechnet, sodass zwei benachbarte Prüflinge nicht die gleiche Klausur erhalten. Sie müssen keinen Code angeben. Es reicht ihren Algorithmus ausreichend detailliert zu beschreiben.
Minimiert Ihr Algorithmus die Anzahl an nötigen unterschiedlichen Klausuren?