Eidgenössische
Technische Hochschule
Zürich

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

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

6. Oktober 2016

Datenstrukturen & Algorithmen — Blatt 3 — HS 16

Abgabe: Am Donnerstag, den 13.10.2016, vor Beginn der Vorlesung um 10 Uhr im Eingangsbereich vor ML D28. Bitte heften Sie Ihre Blätter zusammen und benutzen Sie dieses Blatt als Deckblatt. Füllen Sie auch die ersten zwei der untenstehenden Felder aus.

Übungsstunde (Raum & Zeit): _____

Abgegeben von: _____

Korrigiert von: _____

erreichte Punkte: ____

Aufgabe 3.1 *Asymptotische Laufzeit einschätzen I.*

Geben Sie für die folgenden Codefragmente in Θ-Notation (so knapp wie möglich) an, wie oft die Funktion f in Abhängigkeit von $n \in \mathbb{N}$ asymptotisch gesehen aufgerufen wird. Begründen Sie Ihre Antwort.

```
1 for(int i = 1; i <= n/3; i = i+3)
2         for(int j = 1; j <= i; j = j+1)
3                 f();
```

```
1 for(int i = 1; i <= n; i = i+1) {
2         for(int j = 100; j*j >= 1; j = j–1)
3                 f();
4         for(int k = 1; k <= n; k = k*2)
5                 f();
6 }
```

```
1 for(int i = 1; i <= n; i = i+1)
2         for(int j = 1; j*j <= n; j = j+1)
3                 for(int k = n; k >= 2; k = k–1)
4                         f();
```

Bitte wenden.

Aufgabe 3.2 *Asymptotische Laufzeit einschätzen II.*

Betrachten Sie das folgende sehr naiv formulierte Codefragment:

```
Funktion f(n)
 1  Wenn n = 0 ist:
 2        Setze a ← g(0). Setze b ← g(0). Gib a + b zurück.
 3  Wenn n = 1 ist:
 4        Setze a ← g(1). Gib a zurück.
 5  Wenn n ≥ 2 ist:
 6        Setze a ← f(n − 1).
 7        Setze b ← f(n − 2).
 8        Setze c ← f(n − 2).
 9        Setze z ← g(a + b + c).
10        Gib z zurück.
```

a) Zeigen Sie, dass die Funktion f bei Eingabe $n \in \mathbb{N}_0$ die Funktion g insgesamt mindestens 2^{n-1} Mal aufruft. Gehen Sie dazu wie folgt vor: Die Funktion $c(n)$ gebe an, wie oft f bei Eingabe n die Funktion g aufruft. Geben Sie eine rekursive Definition für $c(n)$ an, und zeigen Sie mittels verallgemeinerter Induktion über n, dass $c(n) \geq 2^{n-1}$ gilt.

b) Ändern Sie den obigen Pseudocode so, dass die Funktion den gleichen Wert berechnet, aber g so selten wie möglich aufgerufen wird. Mit wie vielen Aufrufen von g kommt Ihr modifizierter Pseudocode aus?

Aufgabe 3.3 *Algorithmenentwurf: Divide-and-Conquer.*

Gegeben sei eine Menge von n Objekten $O_1, \ldots, O_n$. Wir wollen entscheiden, ob diese ein *Mehrheitsvotum* besitzt. Dies ist genau dann der Fall, wenn (echt) mehr als die Hälfte aller Objekte gleich sind. Entwerfen Sie einen Algorithmus für dieses Problem. Die einzig erlaubte Elementaroperation ist der Gleichheitstest, d.h. für zwei Objekte O_i und O_j können Sie in Zeit $\mathcal{O}(1)$ prüfen, ob sie gleich sind. Sie dürfen aber **nicht** annehmen, dass die Einträge aus einem geordneten Universum (z.B. $\mathbb{N}$) stammen, also mittels "$<$" verglichen werden können.

a) Überlegen Sie zunächst, wie ein naiver Algorithmus für das Problem aussehen könnte. Welche Laufzeit hat Ihre Lösung?

b) Geben Sie einen Algorithmus an, der das Problem mittels Divide-and-Conquer in Zeit $\mathcal{O}(n \log n)$ löst. Beweisen Sie auch, dass die Laufzeit Ihres Algorithmus in $\mathcal{O}(n \log n)$ liegt.

Hinweis: Sie dürfen annehmen, dass $n = 2^k$ für ein $k \in \mathbb{N}$ gilt. Wenn man die gegebene Menge der Objekte in zwei gleich grosse Teile aufspaltet, wie kann man dann aus den beiden Teillösungen etwas für die Gesamtlösung schlussfolgern?

c) **Bonusfrage:** Ist die Lösung aus Aufgabenteil b) bestmöglich, oder können Sie eine noch schnellere angeben?

Bitte beschreiben Sie Ihren Algorithmus anschaulich und/oder in **kommentiertem** Pseudocode, sodass Ihre Idee verständlich wird. Bitte verzichten Sie darauf, Java-Code abzugeben.