Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme
Universität Würzburg

Würzburg, den 14. November 2019

Prof. Dr. Alexander Wolff
Andre Löffler, M. Sc.

1. Präsenzübungsblatt zur Vorlesung Algorithmen und Datenstrukturen (Winter 2018/19)

Aufgabe 1 – Algorithmen erkennen (2)

Geben Sie für jeden der beiden Algorithmen an, was er bewirkt und bestimmen sie möglichst genau seine Laufzeit (Groß-Oh-Notation).

a) Algorithmus1(int[ ] A, int $\ell = 1$, int $r = A.length$)

```
if 1 ≤ ℓ and ℓ < r and r ≤ A.length then
    key = A[ℓ]
    A[ℓ] = A[r]
    A[r] = key
    Algorithmus1(A, ℓ + 1, r − 1)
```

b) Algorithmus2(int[ ] A)

```
n = A.length
for i = 1 to n do
    if A[i] ≠ 0 then
        a = A[i]
        A[i] = 0
        b = Algorithmus2(A)
        if a > b then
            return b;
        return a;
return 0
```

Aufgabe 2 – O-Notation

Sei $f\colon \mathbb{N} \to \mathbb{R}$ eine Funktion. Beweisen oder widerlegen Sie folgende Behauptungen. Arbeiten Sie mit der Definition aus der Vorlesung, nicht mit Grenzwertbetrachtungen.

a) Für $f\colon n \mapsto (n+1)^5$ gilt $f \in \Omega(n^6)$.

b) Für $f\colon n \mapsto 2^{n+1}$ gilt $f \in O(2^n)$.

Aufgabe 3 – FunSort

Gegeben sei folgender Sortieralgorithmus in Pseudocode, wobei die Methode Merge aus MergeSort übernommen wurde.

```
FunSort(array of int A)
i = 1
while i < A.length do
    Merge(A, 1, i, i + 1)
    i = i + 1
```

a) Welchem Algorithmus, den Sie aus der Vorlesung kennen, ähnelt FunSort? Begründen Sie Ihre Antwort.

b) Sei $T_{max}(n)$ die maximale Laufzeit von FunSort über alle Eingaben der Größe n. Geben Sie eine Funktion f an, so dass $T_{max} \in \Theta(f)$.

   Begründen Sie Ihr Ergebnis.

c) Sei $T_{min}(n)$ die minimale Laufzeit von FunSort über alle Eingaben der Größe n. Geben Sie eine Funktion g an, so dass $T_{min} \in \Theta(g)$.

   Begründen Sie Ihr Ergebnis.

d) Beweisen Sie die Korrektheit des Algorithmus mit folgender Schleifeninvarianten:

   Bei der i-ten Ausführung des while-Schleifenkopfes gilt, dass

   (i) $A[1..i]$ *dieselben Elemente wie zu Beginn der Ausführung des Algorithmus enthält – jedoch sortiert.*

   (ii) $A[i + 1..A.length]$ *sich seit der Ausführung des Algorithmus nicht verändert hat.*

Aufgabe 4 – Mehr Rekursionsgleichungen

Geben Sie für die folgenden Rekursionsgleichung je ein $g(n)$ an, sodass $T(n) \in \Theta(g(n))$ gilt. Verwenden Sie je die vorgegebene Vorgehensweise und geben Sie alle nötigen Zwischenschritte an.

a) Verwenden Sie die Meister Methode. Geben Sie auch den verwendeten Fall an.

$$T(n) = 4T(n/2) + n^3 + 5$$

b) Verwenden Sie die Rekursionsbaum Methode. Gehen Sie davon aus, dass $T(0) = 1$ gilt.

$$T(n) = 3T(n/3) + n^2$$

PABS **Aufgabe 5 – Prioritätsschlange mit Heaps implementieren**

Die folgende Aufgabe wird nicht in der Übung besprochen, sondern ist als freiwillige Vorbereitung für den Kurztest gedacht.

Sie sollen eine Prioritätsschlange in Java implementieren, die auf Heaps basiert und die Laufzeitvorgaben aus der Vorlesung einhält. Sie können hiermit die Funktionsweise eines Heaps nachvollziehen und ihre Lösung von PABS automatisch testen lassen.

In der entsprechenden Aufgabe in PABS sind bereits einige Dateien vorgegeben:

- Es gibt eine Klasse `Element`, die die einzelnen Elemente der Queue darstellen soll. Neben der Priorität verfügt jedes Element über einen `String value`, der die Nutzdaten für dieses Element darstellt. Außerdem wird aus praktischen Gründen jeweils noch der aktuelle Index im Heap gespeichert (`index`).

  An dieser Klasse müssen Sie keine Änderung vornehmen und können Sie direkt so verwenden. Es sind Getter- und Setter-Methoden implementiert um auf die einzelnen Eigenschaften zuzugreifen.

- In einer zweiten Datei ist ein Rahmen für die Klasse `PriorityQueue` vorgegeben. Hier ist bereits die Funktion `swap` implementiert, die zwei Elemente im Feld vertauscht. Hierbei werden auch die in den Elementen gespeicherten Indizes richtig aktualisiert.

  Außerdem ist hier eine Variante von `IncreaseKey` vorgegeben, die (abweichend zur Vorlesung) statt eines Indizes ein Element erwartet, dessen Priorität erhöht werden soll. Sie brauchen diese Funktion nicht zu modifizieren.

Implementieren Sie eine Prioritätsschlange in der Klasse `priority.PriorityQueue`, die Elemente vom Typ `priority.Element` verwaltet.

In der Klasse `PriorityQueue` sind die Methoden mit einem Todo-Kommentar markiert, die Sie selbst noch ergänzen müssen. Sie können sich natürlich noch weitere private Hilfsmethoden anlegen. Hierfür bieten sich zum Beispiel die Funktionen `left`, `right`, `parent` und `maxHeapify` aus der Vorlesung an.

Um die PABS-Tests zu bestehen, muss ihre Implementierung, die Funktionen `Insert`, `FindMax`, `ExtractMax` und `IncreaseKey` aus der Vorlesung umsetzen. Außerdem muss es einen Konstruktor geben, der ein Feld von `Element` übergeben bekommt und daraus die PriorityQueue implementiert entsprechend der Funktion `BuildMaxHeap` aus der Vorlesung. In den Fällen, in denen im Pseudocode aus der Vorlesung Fehlermeldungen zurückgegeben werden, soll jeweils eine `IllegalArgumentException` geworfen werden.

Diese Aufgabe muss nicht abgegeben werden und wird auch nicht korrigiert. PABS testet mittels einiger Stichproben die Korrektheit Ihrer Implementierung. Dabei wird aber insbesondere natürlich nicht getestet, ob die asymptotische Laufzeit Ihrer Implementierung der aus der Vorlesung entspricht.

Diese Aufgaben werden eventuell gemeinsam in den Übungen am 19. und 20. November 2019 gelöst. Sie brauchen Sie nicht vorher zu lösen und auch nicht abzugeben.