Technische Universität München
Fakultät für Informatik, I-14
Prof. Harald Räcke
Robin Münk

SoSe 2025
Übungsblatt 9
20. Juni 2025

Algorithmen und Datenstrukturen

Aufgabe 1 QuickSort - Median of Medians

Der Median einer Liste an Zahlen ist der Wert, welcher in der sortierten Liste genau in der Mitte stehen würde und somit die Hälfte der größeren Zahlen von der Hälfte der kleineren Zahlen trennt. Es gibt somit gleich viele Werte in der Liste die größer sind, als der Median, wie Zahlen, die kleiner sind als der Median.

Der Median einer Liste kann zum Beispiel mit dem BFPRT-Algorithmus gefunden werden. Für die Laufzeit dieses rekursiven Algorithmus gilt folgende Rekurrenz:

$$T(n) \leq \begin{cases} c & n \leq 1 \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn & \text{sonst} \end{cases}$$

für ein $c \geq 1$.

a) Lösen Sie die Rekurrenz, indem Sie mit vollständiger Induktion beweisen, dass für alle $n \geq 1$ gilt $T(n) \leq 10 \cdot cn$.

b) Stellen Sie eine Rekursionsgleichung für die Laufzeit von QuickSort auf, wenn als Subroutine für die Bestimmung des Pivot-Elements in jedem Schritt der BFPRT-Algorithmus verwendet wird.

c) Lösen Sie die Rekurrenz aus b) und geben Sie einen geschlossenen Ausdruck für eine obere Schranke an die Laufzeit dieser Variante von QuickSort an.
Vergleichen Sie das Ergebnis mit der Laufzeit von MergeSort und der asymptotischen worst-case Laufzeit von QuickSort aus der Vorlesung.

Aufgabe 2 Stack und Queue

a) Gegeben sei ein Stack S:

$$\leftrightarrow (5,8,2,3,9)$$

Dabei sind folgende Operationen auf dem Stack definiert:

- S.`pop()`: Liefert das oberste Objekt und entfernt es vom Stack S.
- S.`push(`x`)`: Legt das Objekt x oben auf den Stack S.
- S.`top()`: Liefert das oberste Objekt ohne es zu entfernen.
- S.`isEmpty()`: Gibt zurück, ob der Stack S leer ist.

Geben Sie jeweils den Zustand von S und die Rückgabe an, wenn folgende Operationen ausgeführt werden:

(i) S.`pop()`

(ii) S.push(8)

(iii) S.top()

(iv) S.push(1)

(v) S.pop()

b) Gegeben sei eine Queue Q:

$$\rightarrow (0,9,1,3,4) \rightarrow$$

Dabei sind folgende Operationen auf der Queue definiert:

- Q.enqueue(x) Fügt das Objekt x and das Ende der Queue Q.
- Q.dequeue() Liefert und entfernt das am längsten gespeicherte Objekt aus Q.
- Q.isEmpty() Gibt zurück, ob die Queue Q leer ist.

Geben Sie jeweils den Zustand von Q und die Rückgabe an, wenn folgende Operationen ausgeführt werden:

(i) Q.dequeue()

(ii) Q.enqueue(7)

(iii) Q.enqueue(5)

(iv) Q.dequeue()

(v) Q.isEmpty()

Aufgabe 3 Queue aus Stacks

a) Realisieren Sie eine Queue mit Hilfe zweier Stacks. Nehmen Sie an, sie haben zwei Stacks mit den zugehörigen Methoden *push*, *pop* und *isEmpty* zur Verfügung.
Implementieren Sie folgende Methoden. Sie können das Testing Framework verwenden.

- enqueue(x)
- dequeue()
- isEmpty()

Tipp: Nutzen Sie Stack 1 zum Einfügen und Stack 2 zum Entfernen von Elementen.

b) Nehmen Sie an, es befinden sich n Objekte in der Queue. Geben Sie die best-case und die worst-case Laufzeit der Operationen *enqueue* und *dequeue* an.

c) Zeigen Sie, dass die drei Operationen *enqueue*, *dequeue* und *isEmpty* konstante amortisierte Laufzeit haben.

Aufgabe 4 Stack - Rekursive Sortierung

a) Gegeben ist ein Stack S. Implementieren Sie eine Funktion **stackSort(S)**, die den Stack S *rekursiv* aufsteigend sortiert. Implementieren Sie dafür eine *rekursive* Hilfsfunktion **sortedInsert(S, val)**, die den Wert *val* in den bereits sortierten Stack S an die richtige Stelle einfügt. Sie können das Testing Framework verwenden.

Der Stack soll dabei nur über die bekannten Stackfunktionen pop, push, top, length und isEmpty benutzt werden.

Beispiel:
Stack vor der Sortierung: $\leftrightarrow$ [5, 3, 2, 6, 7]
Stack nach der Sortierung: $\leftrightarrow$ [2, 3, 5, 6, 7]

b) Analysieren Sie die Laufzeit der Funktion **sortedInsert**. Leiten Sie daraus die Laufzeit von **stackSort** ab.