Übungsblatt 3

Abgabe via Moodle.
Deadline Fr. 27ter Mai

Aufgabe 1 (*Amortisierte Analyse,* $2+4+2$)

Auf einer Datenstruktur wird eine Sequenz $\sigma = (\sigma_1, \cdots, \sigma_n)$ von Operationen ausgeführt. Die Operation σ_i kostet i, wenn i eine Potenz von vier ist, sonst 1.

1. Berechnen Sie die Kosten für 256 Operationen $T(256)$.
2. Geben Sie eine geschlossene Form $T(n)$ für die Kosten von n Operationen an. Nehmen Sie dabei vereinfachend an, dass $n = 4^m$ für ein $m \in \mathbb{N}$ ist. **Tipp:** Die Formel für endliche Partialsummen einer geometrischen Reihe könnten hilfreich sein.
3. Wie groß sind dann die amortisierten Kosten pro Operation? Schätzen Sie die amortisierten Kosten auch in $\mathcal{O}$-Notation ab.

Aufgabe 2 (*Stack, Queue und Deque mittels einfach verketteter Liste,* $3+3+2$ *Punkte*)

1. Wie baut man einen unbeschränkten Stack mittels einer *einfach* verketteten Liste? Geben Sie hierzu "Implementierungen" von *pushFront* und *popFront* in Pseudocode an, so dass die Operationen jeweils nur konstante Zeit benötigen. Dabei darf *keine* bereits fertig "'implementierte"' Datenstruktur verwendet werden, d.h. Sie müssen alles selbst bauen.
2. Wie baut man eine unbeschränkte Queue mittels einer *einfach* verketteten Liste? Erweitern Sie hierzu Ihre "'Implementierung"' aus Teilaufgabe a) um eine Operation *pushBack* und geben Sie diese in Pseudocode an, so dass die Operation nur konstante Zeit benötigt.
3. Eine *Double-Ended Queue* (Deque) hat die Operationen *pushFront*, *pushBack*, *popFront* und *popBack*. Lässt sich auch eine Deque mit einer einfach verketteten Liste so konstruieren, dass die Operationen *pushFront*, *pushBack*, *popFront* und *popBack* alle nur konstante Zeit benötigen? Oder braucht man z.B. eine doppelt verkettete Liste? Begründen Sie kurz!

Aufgabe 3 (*Queue mittels Arrays,* $3+3$ *Punkte*)

Wie baut man eine unbeschränkte Queue mit Hilfe von (unbeschränkten) Arrays?

1. Geben Sie hierzu "'Implementierungen"' von *pushBack* und *popFront* in Pseudocode an, so dass die Operationen jeweils nur *amortisiert* konstante Zeit benötigen.
2. Zeigen Sie, dass *pushBack* und *popFront*, wie von Ihnen in a) angegeben, tatsächlich das gewünschte Zeitverhalten aufweisen.

Aufgabe 4 (*Unbounded Arrays,* 2 *Punkte*)

Gegeben sei ein unbounded Array, dessen Größe halbiert wird, sobald das Array nur noch zur Hälfte gefüllt ist. Sei $n \in \mathbb{N}, n \geq 8$ eine Viererpotenz. Geben Sie eine Folge von n Operationen an, so dass ein derart schlecht implementierter unbounded Array mit diesen Operationen einen Aufwand von $\Theta(n^2)$ verursacht.

Aufgabe P3 (*Merging Arrays, optional*)

Für die praktischen Übungen verwenden wir die Plattform `www.hackerrank.com`. Hier müssen Sie sich registrieren um an den Übungen teilzunehmen. Unter dem Link

`https://www.hackerrank.com/ads-i-praktische-uebung`

finden die praktischen Übungen in der Form eines Programmierwettbewerbs statt.

Die dritte Challenge heißt "Merging Arrays". Ihre Aufgabe ist es zwei gegebene, bereits sortierte Arrays zu einem zusammen zu führen. Das Resultat sollte ebenfalls ein sortiertes Array sein. Dieser Vorgang entspricht dem letzten Schritt des Algorithmus **Merge Sort**, den Sie in der Vorlesung kennen lernen.

Um die Aufgabe zu lösen müssen Sie die Funktion **merge**, sowie die Funktion **array_size** vervollständigen.

Da Sortierung ein wichtiger Bestandteil vieler Problemlösungen ist sollte dieses Subproblem möglichst effizient gelöst werden.

Um die Effizienz ihres Algorithmus zu testen gibt es einen großen Test (Case 3) auf Hackerrank. Wenn Ihr Algorithmus schnell genug ist, wird der Test in der vorgegebenen Zeit von 2 Sekunden meistens durchlaufen. Leider laufen diese Tests nicht immer gleich schnell, weshalb eine mehrmalige Abgabe Ihres Algorithmus hilfreich sein könnte.

Falls dieser Test trotzdem immer aufgrund von Zeitmangel fehlschlägt, gibt es eventuell die Möglichkeit Ihren Algorithmus noch zu verbessern.

Eine genauere Beschreibung, sowie ein Beispiel finden Sie auf HackerRank.