Übungsblatt 5

Abgabe via Moodle.
Deadline Fr. 9ter Juni

Aufgabe 1 (*Algorithmendesign,* $4 + 6$ *Punkte*)

Gegeben sei eine sortierte Liste mit n Elementen. Nun haben Sie $k < n$ neue unsortierte Elemente in einem Array, die in die sortierte Liste eingefügt werden sollen, so dass die Liste nach dem Einfügen wieder sortiert ist.

1. *Warmup:* Geben Sie einen Algorithmus an, der Laufzeit $\mathrm{O}(kn)$ hat und das Problem löst.
2. Geben Sie nun einen Algorithmus an, der (erwartete) Laufzeit $\mathrm{O}(k \log k + n)$ hat und das Problem löst.

Lösung:

1. Man füge die Elemente nacheinander in die sortierte Liste ein. Um ein Element einzufügen macht man lineare Suche: man geht die Liste Stück für Stück durch und vergleicht den Wert des aktuellen Elements mit dem Wert des einzufügenden Elements bis man die richtige Stelle gefunden hat.
   Die lineare Suche hat pro einzufügendem Element Aufwand $\mathrm{O}(n + k)$, da im schlimmsten Fall die ganze Liste durchlaufen werden muss. Da wir k Elemente einfügen, ergibt sich somit ein Gesamtaufwand von $\mathrm{O}(k(n + k))$ und da $k < n$ die Abschätzung des Gesamtaufwands $\mathrm{O}(kn)$.
2. Nun gehen wir ein wenig anders vor. Zuerst sortieren wir das Array der neuen Elemente in erwarteter Zeit $\mathrm{O}(k \log k)$ zum Beispiel mit dem Algorithmus Quicksort aus der Vorlesung. Das sortierte Array wird dann in eine Liste konvertiert. Dann haben wir zwei sortierte Listen: in der einen sortierten Liste befinden sich die k neuen Elemente und in der anderen befinden sich die n alten sortierten Elemente. Nun verwenden wir die Funktion *merge* aus der Vorlesung und erhalten so in Zeit $\mathrm{O}(n)$ eine sortierte Liste, die die Elemente beider Listen enthält. Insgesamt ergibt sich also ein Aufwand von $\mathrm{O}(k \log k + n)$

Aufgabe 2 (*Anwendungsproblem,* $6 + 2$ *Punkte*)

In einer Fabrik werden n Waren pro Monat hergestellt, die zu unterschiedlichen Zeitpunkten von den Lieferanten abgeholt werden. In der Lagerhalle der Fabrik gibt es k Lagerplätze für die produzierten Waren. Der Fabrikleiter muss nun den Produktionsplan kontrollieren, ob genügend Lagerplätze vorhanden sind, um die entsprechenden Waren von Produktionsdatum bis Abholdatum zu lagern. Eine Produktionseinheit besteht also im Plan aus Produktionsdatum und Abholdatum. Der Manager möchte wissen, ob genug Plätze vorhanden sind um die Anforderungen zu erfüllen. Die Lieferanten kommen immer zu Beginn des Tages um die Ware abzuholen, also bevor Waren dieses Tages gelagert werden müssen.

1. Entwerfen Sie einen Algorithmus, der dieses Problem in höchstens $\mathrm{O}(n \log n)$ Zeit löst.

   Hinweis. Betrachten Sie die Menge aller Produktions- und Abholdaten. Sortieren Sie diese Menge und verarbeiten Sie sie dann in sortierter Reihenfolge.
2. Begründen Sie kurz, warum Ihr Algorithmus das gewünschte Laufzeitverhalten aufweist.

Lösung:

1. Man alloziere ein beschränktes Array A der Größe $2n$ mit Einträgen aus $\mathit{Datum} \times \{\uparrow, \downarrow\}$. Nun iteriere man über alle Produktionseinheiten $P_1, \ldots P_n$: Für jede Einheit P_i mit Produktionsdatum a_i und Abreisedatum b_i setze man $A[2i] := (a_i, \uparrow)$ und $A[2i+1] := (b_i, \downarrow)$. Danach sortiere man mittels Mergesort das Array A gemäß dem Datum, bei gleichem Datum sollen "'Abholtupel"' jedoch bevorzugt werden. Also gemäß der Ordnung

   $$(t, x) < (t', y) \iff (t \text{ früher als } t') \text{ oder } (t = t' \text{ und } x = \downarrow \text{ und } y = \uparrow) ,$$

   wobei man durch Hinzunahme der Gleichheit die Relation $\leq$ auf den Tupeln gewinnt. Als letztes lese man nun A von links nach rechts durch. Dabei verwalte man einen Zähler c, der initial den Wert 0 hat. Jedes mal, wenn man ein Element $(t, \uparrow)$ betrachtet, erhöht man c um eins. Jedes mal, wenn man ein Element $(t, \downarrow)$ betrachtet, vermindert man c um eins. Tritt während dieses Vorgangs jemals der Fall $c > k$ ein, hat das Lager nicht genug Platz und der Produktionsplan muss erneuert werden.

2. Iterieren über n Produktionseinheiten: Zwei Zuweisungen an Arrayeinträge dauern $\mathrm{O}(1)$, das ganze n-mal dauert $\mathrm{O}(n)$ Zeit. Um $2n$ Elemente zu sortieren benötigt Merge-Sort

   $$\mathrm{O}(2n \log 2n) = \mathrm{O}(2n \log 2 + 2n \log n) = \mathrm{O}(n \log n)$$

   Zeit. Erneutes Iterieren über A mit Erhöhung oder Verminderung eines Zählers dauert wieder nur $\mathrm{O}(n)$. Gesamtlaufzeit ist offenbar in $\mathrm{O}(n \log n)$.

Aufgabe 3 (*Sortieren,* 6 *Punkte*)

Sortieren Sie die Ziffern Ihrer Matrikelnummer.

1. Benutzen Sie Insertionsort. Geben Sie den Zustand des Feldes nach jedem der 6 Insert-Schritte an.
2. Benutzen Sie Mergesort. Verwenden Sie das Schema aus dem Beispiel der Vorlesung (Folie 143, Stand 12.4., Überschrift “Mischen”).
3. Benutzen Sie Quicksort. Verwenden Sie das Schema aus dem Beispiel der Vorlesung (Folie 165, Stand 12.4., Überschrift “Beispiel: Rekursion”). Als Pivot soll das erste Element verwendet werden.

Lösung:
Beispiel mit Matrikelnummer 1229203.

1. | 1,2 | 2,9,2,0,3 |
   |---|---|
   | 1,2,2 | 9,2,0,3 |
   | 1,2,2,9 | 2 0,3 |
   | 1,2,2,2,9 | 0,3 |
   | 0,1,2,2,2,9 | 3 |
   | 0,1,2,2,2,3,9 | |

2. $\langle 1, 2, 2, 9, 2, 0, 3 \rangle$
   split
   $\langle 1, 2, 2 \rangle \ \langle 9, 2, 0, 3 \rangle$
   split
   $\langle 1 \rangle \ \langle 2, 2 \rangle \ \langle 9, 2 \rangle \ \langle 0, 3 \rangle$
   split
   $\langle 1 \rangle \ \langle 2 \rangle \ \langle 2 \rangle \ \langle 9 \rangle \ \langle 2 \rangle \ \langle 0 \rangle \ \langle 3 \rangle$
   merge

$\langle 1 \rangle$ $\langle 2, 2 \rangle$ $\langle 2, 9 \rangle$ $\langle 0, 3 \rangle$
merge
$\langle 1, 2, 2 \rangle$ $\langle 0, 2, 3, 9 \rangle$
merge
$\langle 0, 1, 2, 2, 2, 3, 9 \rangle$

3. 1 2 2 9 2 0 3
   0|2 2 9 2 1 3
   |1 2|9 2 2 3
   |1 2|3 2 2|9
   | |2 2|3| |
   | |2 2| | |

Aufgabe P5 (*Algorithmendesign-praktisch, 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/adsi-2023`

finden die praktischen Übungen in der Form eines Programmierwettbewerbs statt.

In der fünften Challenge geht es um Aufgabe 1 dieses Übungsblattes.
Gegeben ist eine sortierte, einfach verkettete Liste mit n Elementen. Nun haben Sie k neue unsortierte Elemente in einem Array, welche in die sortierte Liste eingefügt werden sollen, sodass die Liste nach dem Einfügen wieder sortiert ist.

Die Aufgabe ist es nun die beiden Algorithmen Ihrer Lösung aus Aufgabe 1 zu implementieren. Sie können in der Challenge die Laufzeiten der beiden Algorithmen vergleichen. Dazu müssen die in der main Funktion markierten Zeilen (164 - 184) auskommentiert werden.

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