Übungsblatt 2

Funke
FMI/ALG
Universität Stuttgart

Aufgabe 1: List Merging and Heaps (schriftlich, 10 Pkt.)

(a) MergeSort muss im letzten Schritt zwei sortierte Listen zu einer sortierten Liste zusammenfügen. Wenn diese Listen jeweils Länge n und m haben, schreiben Sie einen $\mathcal{O}(n+m)$ Algorithmus als Pseudocode.

(b) Jetzt allgemeiner: Angenommen Sie bekommen k sortierte Listen mit gesamt n Elementen in allen Listen. *Beschreiben* Sie einen $\mathcal{O}(n \log(k))$ Algorithmus, der diese Listen zu einer sortierten Liste zusammenfügt.

(c) Beschreiben Sie die Funktionsweise der Heap Funktionen *delete* und *change_key*.

(d) Wie können Sie einen ternären Heap effizient im Speicher ablegen?

Aufgabe 2: Insertionsort

Betrachten Sie den folgenden Sortieralgorithmus *Insertionsort*:

```
InsertionSort(A, n)
 for j=1 to n-1 do
  key = A[j]
  i = j-1
  while (i>=0 AND A[i]>key) do
   A[i+1] = A[i]
   i = i-1
  od
 A[i+1] = key
 od
```

(a) Beschreiben Sie kurz die Funktionsweise des Algorithmus und begründen Sie dessen Korrektheit.

(b) Beweisen Sie die Laufzeitschranken von *Insertionsort* für den best- und worst-case.

Aufgabe 3: Stabile Sortierverfahren

Wir nennen ein Sortierverfahren stabil, wenn es die folgende Eigenschaft erfüllt:

Sei $A = [a_1, \ldots, a_n]$ eine zu sortierende Liste und π die Permutation, die durch ein Sortierverfahren entsteht. Dann ist das Sortierverfahren stabil, wenn für alle $i, j \in \{1, \ldots n\}$ mit $i < j$ und $a_i = a_j$ gilt, dass dann auch $\pi(i) < \pi(j)$ ist, also die Reihenfolge der Elemente mit gleichem Schlüssel bewahrt.

Zeigen Sie für die in der Vorlesung und Übung bisher vorgestellten Sortierverfahren

- Bubblesort
- Mergesort
- Heapsort
- Insertionsort

mit Beweis oder Gegenbeispiel, ob diese stabil sind oder nicht. Gibt es bei den nicht stabilen Verfahren eine Möglichkeit, diese zu adaptieren, um Stabilität zu erzeugen?

Aufgabe 4: Rekursion

Das Closest-Pair Problem ist wie folgt definiert:

Auf Eingabe einer Punktmenge $\mathcal{P} \subseteq \mathbb{R}^2$ suchen wir die zwei Punkte mit dem geringsten Abstand.

Zur Lösung des Problems können wir folgendermaßen vorgehen:

Zunächst sortieren wir die Liste der Punkte nach ihrer x-Koordinate. Dann rufen wir folgenden rekursiven Divide-and-Conquer Algorithmus auf:

```
findShortestDistance(List points)
 if (len(points) == 1) do
  return inf
 od
 middle = floor(len(points)/2)
 dLeft = findShortestDistance(points[0,...,middle])
 dRight = findShortestDistance(points[middle + 1, len(points)])
 dOverall = computeOverallShortestDistance(points, dLeft, dRight)
 return dOverall
```

In jedem Rekursionsschritt sind also die kürzesten Distanzen der rechten und linken Hälfte bekannt.

(a) Wie können Sie die overallShortestDistance finden? Wie finden Sie diese in $\mathcal{O}(n)$?
Hinweis: Sortieren Sie die Liste der Punkte nach ihrer y-Koordinate.

(b) Mit diesem Ansatz ergibt sich eine Laufzeit von $\mathcal{O}(|\mathcal{P}| \log^2 |\mathcal{P}|)$. Können Sie den Ansatz modifizieren, dass sich die Laufzeit auf $\mathcal{O}(|\mathcal{P}| \log |\mathcal{P}|)$ reduziert?