Präsenzblatt 2

Hinweis: Dieses Aufgabenblatt wurde von Tutoren erstellt. Die Aufgaben sind für die Klausur weder relevant noch irrelevant.

Aufgabe 2.1: Einfache Laufzeit

Bestimmen Sie die Laufzeiten der folgenden Algorithmen:

Algorithm 1: Recursion

```
1 Function f(N):
2     if N = 1 then
3         return 1
4     else
5         return f(N − 1) + f(N − 1)
```

Algorithm 2: Iteration

Input: N

```
1 c ← 0;
2 for i = 0 to N do
3     for j = 0 to i do
4         c ← c + 1
```

Aufgabe 2.2: Online

Welchen Sortieralgorithmus würden Sie wählen, wenn die Schlüssel über einen längeren Zeitraum nach und nach ankommen? Insbesondere soll die Sortierung nach Eintreffen des letzten Schlüssels schnell fertig sein. Sortieralgorithmen, die eine solche Eigenschaft besitzen, bezeichnet man auch als Online-Sortieralgorithmen.

Aufgabe 2.3: Baumvermessung

Wir betrachten einen balancierten Binärbaum der Höhe n. Welche Größe hat der Baum? Wie viele Blätter hat er? Beweisen Sie Ihre Aussagen.

Aufgabe 2.4: MergeSort

Aus der Programmierung 1 kennen Sie noch MergeSort. Wie arbeitete dieser Algorithmus? Welche Laufzeitkomplexität können Sie beweisen? Was sind Nachteile gegenüber HeapSort in Bezug auf den Speicherverbrauch?

Aufgabe 2.5: Nicht-einfache Laufzeit

(a)

Algorithm 3: Iteration Runde 2

Input: N

```
1 c ← 0;
2 for i ← 1; i ≤ N; i ← 2i do
3     for j ← 0; j < i; j ← j + 1 do
4         c ← c + 1
```

Hinweis: Wählen Sie eine Laufzeit hiervon aus und versuchen Sie den Beweis:

$$\Theta(\log^2 N) \quad \Theta(N) \quad \Theta(N \log N) \quad \Theta(N \log\log N) \quad \Theta(N\sqrt{N}) \quad \Theta(N^2)$$

(b) Betrachten Sie MergeSort mit folgender Merge-Funktion:

Algorithm 4: Merge'

Input: sortierte Listen X, Y

```
1 A ← [ ];
2 while X und Y nicht leer sind do
3     bestimme das maximale Element aus X und Y ;        // in Θ(n) Zeit
4     entferne das Element und füge es vorne an A an
5 return A
```

Bestimmen Sie die Laufzeit des neuen MergeSort.

Aufgabe 2.6: Mach 'nen Heap

In der Vorlesung haben Sie gesehen, dass `makeHeap(A, n)` eine Laufzeit von $\sum_{v \in V} \mathcal{O}(h_v)$ hat, wobei V die Menge der Baumknoten sei und h_v für die Höhe des Teilbaums mit Wurzel v stehe. Zeigen Sie, dass $\sum_{v \in V} \mathcal{O}(h_v) = \mathcal{O}(n)$ gilt.

Aufgabe 2.7: Zufall ist echt mächtiger

Gegeben sei ein n Elemente langes Array A mit $\frac{n}{2}$ 0en und $\frac{n}{2}$ 1en.

(a) Entwickeln Sie einen deterministischen Algorithmus um eine 1 zu finden. Bestimmen Sie die Laufzeit im Worst-, Average- und Best-Case.

(b) Entwickeln Sie einen randomisierten Algorithmus um eine 1 zu finden. Bestimmen Sie die Laufzeit im Worst-, Average- und Best-Case. Stellen Sie Unterschiede zum deterministischen Algorithmus fest?

(c) Beweisen Sie, dass jeder deterministische Algorithmus eine Worst-Case Laufzeit $\mathcal{O}(n)$ hat.

(d) Wie sehen die Laufzeiten aus, wenn sich die Verteilung an 0en und 1en ändert?

Merge' etwas formaler

Algorithm 5: Merge'

Input: X_0, X_1 mit $\forall 0 \leq a < 1, \forall 1 \leq i < |X_a|, X_a[i-1] \leq X_a[i]$

```
1  A ← [ ];
2  while X_0 ≠ [ ] ∧ X_1 ≠ [ ] do
3      for a = 0 to 1 do
4          max_a ← −∞;
5          ind_a ← 0;
6          for i = 0 to |X_a| − 1 do
7              if max_a < X_a[i] then
8                  max_a ← X_a[i];
9                  ind_a ← i
10     if max_0 > max_1 then
11         X_0.remove(ind_0);
12     else
13         X_1.remove(ind_1);
14     A.push_front(max_{0≤a≤1} max_a)
   // Ein X_a is leer und das andere kleiner als alles in A
15 A ← X_0 ++X_1 ++A;
16 return A
```