Algorithmen und Datenstrukturen

Blatt 2

Uni Hamburg, Wintersemester 2018/19

Präsentation am 13.–15. November 2019

Jede Teilaufgabe (a, b, ...) zählt als ein einzelnes Kreuzchen.

Übung 1.

Betrachten Sie folgende Funktionsdefinition zur Prüfung ob zwei gegebene Listen disjunkt sind, also keine gleichen Elemente enthalten:

```
1 def disjoint(a:list, b:list):
2     different_counter = 0
3     for x in a:
4         for y in b:
5             if x != y:
6                 different_counter += 1
7     return different_counter == len(a) * len(b)
```

(a) Erläutern Sie die Vorgehensweise des Algorithmus und beweisen Sie seine Korrektheit.

(b) Geben Sie die Laufzeitkomplexität von `disjoint` an – welches sind die entscheidenden Einflussfaktoren auf die Laufzeit?

(c) Schlagen Sie Möglichkeiten vor, den Code zu optimieren. Lohnt es sich, die Listen `a` und `b` extra für den Aufruf dieser Funktion zu sortieren?

Übung 2.

Wir betrachten den rekursiven Algorithmus Multiply zur Multiplikation zweier Zahlen. Dabei sollen folgende Annahmen gelten:

- n und m sind natürliche Zahlen.
- Length(n) berechnet die Länge (also die Anzahl an Ziffern) von n in Binärdarstellung in Laufzeit $\mathcal{O}(1)$.
- Split(n, k) gibt natürliche Zahlen n_1 und n_0 zurück, sodass $n = n_1 \cdot 2^k + n_0$ sowie $n_0 < 2^k$ gilt, und hat eine Laufzeit von $\mathcal{O}(\text{Length}(n))$.
- ShiftLeft(n, k) berechnet $n \cdot 2^k$ in einer Laufzeit von $\mathcal{O}(\text{Length}(n) + k)$.
- Vergleichsoperationen mit 0 oder 1 haben eine Laufzeit von $\mathcal{O}(1)$.
- Die Laufzeit der Multiplikation und Division von k mit 2 ist $\mathcal{O}(\text{Length}(k))$.
- $n + m$ bzw. $n - m$ haben eine Laufzeit von $\mathcal{O}(\max(\text{Length}(n), \text{Length}(m)))$.

```
MULTIPLY(n, m)
 1  if n == 0 or m == 0
 2      return 0
 3  elseif n == 1
 4      return m
 5  elseif m == 1
 6      return n
 7
 8  k = max(LENGTH(n), LENGTH(m)) / 2
 9  n_1, n_0 = SPLIT(n, k)                    // n = n_1 · 2^k + n_0;  n_0 < 2^k
10  m_1, m_0 = SPLIT(m, k)                    // m = m_1 · 2^k + m_0;  m_0 < 2^k
11
12  a_2 = MULTIPLY(n_1, m_1)
13  a_0 = MULTIPLY(n_0, m_0)
14  a_1 = MULTIPLY(n_1 + n_0, m_1 + m_0) − a_2 − a_0
15
16  return SHIFTLEFT(a_2, 2k) + SHIFTLEFT(a_1, k) + a_0
```

(a) Sind die Annahmen (insbesondere in Bezug auf die Laufzeit) plausibel? Erklären Sie, warum bzw. warum nicht.

(b) Gehen Sie MULTIPLY Schritt für Schritt durch und erklären Sie die Funktionsweise. (Hinweis: Versuchen Sie, sowohl das Produkt $n \cdot m$ als auch den von MULTIPLY zurückgegebenen Term durch n_1, n_0, m_1, m_0 und k auszudrücken.) *Optional:* Versuchen Sie, ihre Erklärung als (Korrektheits-)Beweis zu formulieren.

(c) Geben Sie die asymptotische Laufzeit von MULTIPLY(n, m) an, wobei wir Annehmen, dass LENGTH(N) = LENGTH(M) gilt. Stellen Sie dazu eine Rekurrenzgleichung für die Laufzeit von MULTIPLY(n, m) auf und wenden Sie daraufhin das Master-Theorem an.

Übung 3.
zur Heap-Datenstruktur:

(a) Ausgehend von einem leeren (max-)Heap, fügen Sie die Elemente 28, 37, 55, 31, 22, 40, 7 in dieser Reihenfolge ein. Geben Sie jeweils den Inhalt des Heaps nach jeder Einfügung an.

(b) Welche Höhe h hat ein Heap mit n Elementen höchstens? Welche Höhe hat er mindestens? Beweisen Sie die Korrektheit Ihrer Antwort.

(c) Was ist die Laufzeit von Heap-Sort für n Elemente wenn diese schon auf- oder absteigend geordnet sind? Begründen Sie Ihre Antwort.