UE DGA WS2020-2021

Übungsblatt 1

Aufgabe 1:

Überlegen Sie sich einen Pseudocode für die folgenden Algorithmen und bestimmen Sie die Anzahl der notwendigen Schritte, die (in Ihrem Pseudocode) nötig sind, um eine n-elementige Menge zu sortieren. Wenden Sie die Algorithmen aud den Datensatz $6, 77, 45, 103, 4, 17$ an.

(a) Selection Sort: der Algorithmus sucht zunächst das kleinste Element und bringt es an die erste Position. Anschließend sucht er das zweitkleinste Element und bringt es an die zweite Position, usw.

(b) Bubble-Sort: Der Algorithmus vergleicht der Reihe nach je zwei benachbarte Zahlen und vertauscht diese, falls sie nicht in der richtigen Reihenfolge angeordnet sind. Dieses Verfahren wird so lange wiederholt, bis alle Zahlen der Eingabe sortiert sind.

Aufgabe 2:

Sequentielle Suche: Gegeben sei ein n-elementiger Datenfeld $A[1, \ldots, n]$ und ein Wert x. Überlegen Sie sich einen Pseudocode, der x durch sukzessive Vergleiche mit den Elementen $A[1], A[2], \ldots$ sucht und einen Wert $j \in \{1, \ldots, n\}$ ausgibt, falls $x = A[j]$, oder NIL ausgibt, falls x nicht in der Liste A enthalten ist.

(a) Beweisen Sie, dass Ihr Algorithmus ein korrektes Ergebnis liefert (etwa mit Zuhilfenahme einer Schleifeninvariante).

(b) Machen Sie für diesen Algorithmus eine best-case-Analyse, eine worst-case-Analyse und eine average-case-Analyse (für die average-case-Analyse soll das Modell der Zufallspermutationen verwendet werden, d.h. alle Permutationen von $\{1, \ldots, n\}$ sind gleich wahrscheinlich. Weiters soll angenommen werden, dass x im Datensatz enthalten ist).

Aufgabe 3:

(a) Binäre Suche: Gegeben sei ein (aufsteigend) sortiertes Datenfeld $A[1, \ldots, n]$ und ein Wert x. Das sogenannte Suchproblem, also einen Index j mit $x = A[j]$ auszugeben, falls x in A enthalten ist, und einen speziellen Wert NIL auszugeben, falls x nicht in A vorkommt, kann hier mittels Divide-and-Conquer gelöst werden. Man vergleicht x mit dem mittleren Element des Datenfelds und ist nach diesem Vergleich entweder fündig geworden oder braucht nur noch das halbe Datenfeld mit der gleichen Prozedur zu durchsuchen. Schreiben Sie ein Programm in Pseudocode für die binäre Suche. Begründen Sie, warum die Laufzeit der binären Suche im schlechtesten Fall $O(\log n)$ ist.

(b) Beim Algorithmus Einfügesortieren wird die sequentielle Suche verwendet, um das bereits sortierte Teilfeld $A[1, \ldots, n]$ (rückwärts) zu durchsuchen. Kann stattdessen die binäre Suche verwendet werden, um die worst-case-Laufzeit von Insertion Sort auf $O(\log n)$ zu verbessern?

Aufgabe 4:

Zeigen Sie, dass die harmonischen Zahlen $H_n = \sum_{k=1}^{n} \frac{1}{k}$ die Abschätzung $H_n = O(\log n)$ gilt, indem Sie

(a) die Summe durch $N = \lfloor \log_2 n \rfloor$ Blöcke der Gestalt $\sum_{j=0}^{2^i-1} \frac{1}{2^i+j}$ (wobei $i = 1, \ldots N$) abschätzen und anhand dieser Aufteilung $\sum_{k=1}^{n} \frac{1}{k} \log_2(n) + 1$ verifizieren.

(b) das Cauchy'sche Integralkriterium verwenden.

Aufgabe 5:

Das Horner-Schema dient zur Auswertung von Polynomen. Die Grundidee dahinter ist die Umformung

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \ldots)).$$

(a) Wiederholen Sie die Funktionsweise des Horner-Schemas und schreiben sie einen Pseudocode für die Auswertung von Polynomen mittels Horner-Schema.

(b) Schreiben Sie einen Pseudocode für die direkte Auswerung von Polynomen (Einsetzen).

(c) Vergleichen Sie die Schrittzahlen beider Codes.

Aufgabe 6:

(a) Zeigen Sie mittels vollständiger Induktion, dass ein Algorithmus, dessen Laufzeit $T(n)$ (wobei $n = 2^k, k \in \mathbb{Z}^+$) der Rekursion

$$T(n) = \begin{cases} 1 & \text{für } n = 2 \\ 2T(\frac{n}{2}) + 1 & \text{für } n = 2^k, k > 1 \end{cases}$$

genügt, $T(n) = n - 1$ erfüllt.

(b) Zeigen Sie mittels vollständiger Induktion, dass ein Algorithmus, dessen Laufzeit $T(n)$ (wobei $n = 2^k, k \in \mathbb{Z}^+$) der Rekursion

$$T(n) = \begin{cases} 2 & \text{für } n = 2 \\ 2T(\frac{n}{2}) + n & \text{für } n = 2^k, k > 1 \end{cases}$$

genügt, $T(n) = n \log_2(n)$ erfüllt.