Prof. Dr. Ruben Jubeh

Regensburg, den 14. April 2023

Votieren und abgeben bis zum 30. April 2023, 23:00 Uhr

Übungen zur Vorlesung

Algorithmen und Datenstrukturen

SoSe 2023

Blatt 3

Wichtige Hinweise:

- > Falls Sie bei der Bearbeitung einer Aufgabe größere Schwierigkeiten hatten und deswegen die Bearbeitung abgebrochen haben, so versuchen Sie bitte Ihre Schwierigkeiten in Form von Fragen festzuhalten. Bringen Sie Ihre Fragen einfach zur Vorlesung oder zur Übung mit!
- > Musterlösungen werden bei Bedarf in den Übungen besprochen!

Aufgabe 1:
Beweisen Sie folgende Aussagen mit Hilfe der Master-Methode:

1. Sei $T(1) = 1, T(n) = T(n/2) + 1$ für alle $n > 1$, dann: $T(n) = \Theta(\log n)$ (Binäre Suche)
2. Sei $T(1) = 1, T(n) = 3T(n/4) + n \log n$ für alle $n > 1$, dann: $T(n) = \Theta(n \log n)$
3. Sei $T(1) = 1, T(n) = 7T(n/2) + n^2$ für alle $n > 1$, dann: $T(n) = \Theta(n^{2,81})$

Aufgabe 2:
Demonstrieren Sie die Funktionsweise der Sortieralgorithmen InsertionSort, BubbleSort und SelectionSort anhand des Feldes $a[] = \{-5, 13, -32, 7, -3, 17, 23, 12, -35, 19\}$. Überprüfen Sie Ihre Demonstration mit Hilfe eines C, C++, Java, C# oder Python-Programms, das die einzelnen Sortierverfahren implementiert und die wesentlichen Zwischenfolgen zum Vergleich ausgibt.

Aufgabe 3:
Modifizieren Sie die in der Vorlesung vorgestellten Sortieralgorithmen wie folgt:

1. Drehen Sie die Reihenfolge bei InsertionSort: Statt über das Maximum einzusortieren, soll über das Minimum einsortiert werden.
2. Lassen Sie bei BubbleSort das Maximum nach hinten wandern, statt das Minimum nach vorne wandern zu lassen.
3. Setzen Sie bei SelectionSort anstelle des Minimums jeweils das Maximum an die richtige Stelle.
4. Wählen Sie bei QuickSort als Pivot-Element ein zufälliges Element.

Erläutern Sie jeweils Laufzeit und Korrektheit Ihrer Algorithmen.

Aufgabe 4:

1. Entwickeln Sie einen Algorithmus mit Laufzeit $\Theta(n \log n)$, der folgende Spezifikation erfüllt:

   - Eingabe: $a[] = \{a_0, \ldots, a_{n-1}\}, s$ mit $a_i \in \mathbb{Z}, s \in \mathbb{Z}, n \in \mathbb{N}$ ($n+1$ ganze Zahlen)
   - Ausgabe: `true`, falls es zwei Elemente a_i, a_j gibt mit $s = a_i + a_j, i \neq j$, `false` sonst

   Implementieren und testen Sie Ihren Algorithmus in C, C++, Java, C# oder Python.

2. Stellen Sie sich vor, Sie sollen zwei quadratische Matrizen $M, N \in \mathbb{R}^{n \times n}$ miteinander multiplizieren. Sei $n = 2^i$ für ein $i \in \mathbb{N}$, dann kann man M, N und $O = M \cdot N$ wie folgt zerlegen mit $M_{ij}, N_{ij}, O_{ij} \in \mathbb{R}^{n/2 \times n/2}$ für $i, j \in \{1, 2\}$:

$$M := \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}, N := \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix}, O := \begin{pmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{pmatrix}$$

   Zeigen Sie, dass die beiden folgenden Varianten die Produktmatrix O korrekt berechnen:

   - Variante 1:

$$\begin{aligned} O_{11} &:= M_{11} \cdot N_{11} + M_{12} \cdot N_{21} \\ O_{12} &:= M_{11} \cdot N_{12} + M_{12} \cdot N_{22} \\ O_{21} &:= M_{21} \cdot N_{11} + M_{22} \cdot N_{21} \\ O_{22} &:= M_{21} \cdot N_{12} + M_{22} \cdot N_{22} \end{aligned}$$

   - Variante 2:

$$\begin{aligned} H_1 &:= (M_{11} + M_{22}) \cdot (N_{11} + N_{22}) \\ H_2 &:= (M_{21} + M_{22}) \cdot N_{11} \\ H_3 &:= M_{11} \cdot (N_{12} - N_{22}) \\ H_4 &:= M_{22} \cdot (N_{21} - N_{11}) \\ H_5 &:= (M_{11} + M_{12}) \cdot N_{22} \\ H_6 &:= (M_{21} - M_{11}) \cdot (N_{11} + N_{12}) \\ H_7 &:= (M_{12} - M_{22}) \cdot (N_{21} + N_{22}) \end{aligned}$$

$$\begin{aligned} O_{11} &:= H_1 + H_4 - H_5 + H_7 \\ O_{12} &:= H_3 + H_5 \\ O_{21} &:= H_2 + H_4 \\ O_{22} &:= H_1 - H_2 + H_3 + H_6 \end{aligned}$$

   Bestimmen Sie die asymptotische Laufzeitkomplexität beider Varianten und vergleichen Sie diese mit der Komplexität der Standardmethode zur Multiplikation zweier Matrizen.