Übungsblatt 2

Abgabe bis 03.05.2018

Besprechung: 07.05.2018 – 10.05.2018

Aufgabe 1: Ternäre Suche (1 + 1 + 4 + 3 + 3 Punkte)

Bei der binären Suche ist der Input ein sortiertes Array A und eine Zahl x (die nicht notwendig im Array A vorkommen muss). Dabei wird A in zwei gleich große Teile A_1 und A_2 geteilt und ermittelt, in welchem der beiden Teile sich x befinden müsste. Dieses Verfahren wird *rekursiv* fortgesetzt. Betrachten Sie nun die *ternäre Suche*, bei der A statt in zwei Teile, in *drei* etwa gleich große Teile A_1, A_2 und A_3 geteilt wird.

(a) Geben Sie ein Array A und ein zu suchendes Element x an, so dass die binäre Suche mit weniger Vergleichen auskommt als die ternäre Suche.

(b) Geben Sie ein Array A und ein zu suchendes Element x an, so dass die ternäre Suche mit weniger Vergleichen auskommt als die binäre Suche.

(c) Geben Sie Pseudocode für die ternäre Suche an. Verwenden Sie dabei Rekursion. Nummerieren Sie die Zeilen in Ihrem Pseudocode und erklären Sie detailliert jede Zeile Ihres Codes.

(d) Analysieren Sie die Zeitkomplexität der ternären Suche. Was können Sie über die asymptotische Laufzeit der ternären Suche im Vergleich zur binären Suche sagen?

(e) Bei jedem Rekursionsschritt werden ein oder zwei Vergleiche benötigt, um zu entscheiden, in welchem Teil des Arrays A das Element x liegt. Was ist die minimale, die durchschnittliche und die maximale Anzahl an Vergleichen die benötigt wird, wenn x nicht in A liegt.

Aufgabe 2: Eine Anwendung der Binären Suche (2 + 4 + (3 optionale) Punkte)

Sei A ein sortiertes Array der Größe n und sei z eine gegebene Zahl. Das Ziel dieser Übung ist es, folgdende Frage zu beantworten: Gibt es in A zwei verschiedene Elemente x und y, so dass $x + y = z$?

(a) Es sollte nicht schwierig sein, einen Algorithmus zu finden, der diese Frage in quadratischer Zeit $\mathcal{O}(n^2)$ beantwortet. Geben Sie Pseudocode für einen solchen Algorithmus an und erklären Sie warum Ihr Algorithmus die Laufzeit $\mathcal{O}(n^2)$ hat.

(b) Verwenden Sie nun die binäre Suche, um einen effizienteren Algorithmus zu finden, der die obige Frage in einer Laufzeit von $\mathcal{O}(n \log n)$ beantworten kann. Geben Sie auch hier Pseudocode an und begründen Sie die Korrektheit Ihres Algorithmus. Erklären Sie, warum Ihr Algorithmus die angegebene Laufzeit hat.

(c) Es ist klar, dass ein Algorithmus für die obige Frage mindestens die Laufzeit $\Omega(n)$ benötigt. Versuchen Sie, einen Algorithmus zu finden, der obige Frage in einer Laufzeit von $\mathcal{O}(n)$ beantwortet. Geben Sie Pseudocode an und begründen Sie die Laufzeit und die Korrektheit Ihres Algorithmus.

Aufgabe 3: Implementierung von Suchalgorithmen (2 + 4 + 6 Punkte)

Laden Sie die Java-Vorlage aus dem Moodle herunter und implementieren Sie die folgenden Methoden:

(a) die Lineare Suche in `linearSearch(int[] array, int key)`.

(b) die Binäre Suche in `binarySearch(int[] array, int key)`.

(c) die Interpolationssuche in `interpolationSearch(int[] array, int key)`.

Für die Implementierung der Interpolationssuche benutzen Sie die folgende Variante aus der Vorlesung, um das jeweils nächste Element zu bestimmen:

$$next \leftarrow \left\lceil \frac{a - S[unten - 1]}{S[oben + 1] - S[unten - 1]} \cdot (oben - unten + 1) \right\rceil + (unten - 1)$$

Verwenden Sie bei Ihrer Implementierung sinnvolle Variablennamen und kommentieren Sie Ihren Code! Laden Sie Ihre `Main.java` ins Moodle. Nicht kompilierende Abgaben werden **mit 0 Punkten** bewertet.