SoSe 2018

Prof. Dr. Margarita Esponda

Objektorientierte Programmierung

4. Übungsblatt

Ziel: Auseinandersetzung mit Sortieralgorithmen und Komplexitätsanalyse.

1. Aufgabe (3 Punkte)

a) Wie groß muss eine sortierte Liste sein, um bei einer rekursiven linearen Suche innerhalb einer Liste einen Stapelüberlauf der Python-Virtuelle-Maschine zu produzieren?

b) Wie groß kann eine sortierte Liste maximal sein, bevor bei einer rekursiven Binärsuche ein Stapelüberlauf verursacht wird?

c) Erläutern Sie beide Antworten.

2. Aufgabe (4 Punkte)

a) Definieren Sie eine **is_sorted** Hilfsfunktion, die bei Eingabe einer Liste kontrolliert, ob die Elemente der Liste sortiert sind. Das Ergebnis sollte gleich 1 oder -1 sein, wenn die Elemente jeweils in ansteigender bzw. absteigender Reihenfolge sortiert sind und 0, wenn die Elemente unsortiert sind.

b) Definieren Sie eine **gen_randlist** Hilfsfunktion, die bei Eingabe eines Wertbereichs **(a, b)** und einer ganzen Zahl **n** eine Liste mit **n** zufälligen Zahlen zwischen **a** und **b** generiert.

3. Aufgabe (14 Punkte)

a) (2 Punkte) Wann ist es sinnvoll, den **Insertsort**-Algorithmus anstatt den **Quicksort**-Algorithmus zu verwenden? Begründen Sie Ihre Antwort.

b) (2 Punkte) Schreiben Sie eine Variante des **Quicksort**-Algorithmus aus der Vorlesung, der den Median-Wert (mittleren Wert) aus drei zufällig gewählten Elementen des zu sortierenden Teilarrays berechnet und diesen Wert als Pivot verwendet.

c) (2 Punkte) Erläutern Sie mit einem konkreten Zahlenbeispiel, warum der **Quicksort**-Algorithmus der Vorlesung nicht stabil ist.

d) (2 Punkte) Analysieren Sie mit Hilfe der O-Notation den zusätzlichen Speicherverbrauch des **Quick-Sort-Algorithmus á la Haskell** aus der Vorlesung.

e) (3 Punkte) Wie oft kann während der Ausführung des **Quicksort**-Algorithmus das kleinste Element maximal bewegt werden? Begründen Sie Ihre Antwort.

f) (3 Punkte) Entwickeln Sie eine Funktion **quick_insert** als Variante des Quicksort-Algorithmus der Vorlesung, die zusätzlich zum sortierenden Array eine ganze Zahl **k** als Parameter bekommt und die Zahlen mit Hilfe des **Insertsort**-Algorithmus sortiert, wenn der noch zu sortierende Subarray kleiner **k** ist.

g) (**6 Bonuspunkte)** Definieren Sie eine Variante der **partition**-Funktion aus der Vorlesung, so dass zwei Positionen (**p**, **q**) als Ergebnis der Funktion zurückgegeben werden mit folgenden Eigenschaften:

   a) Die Elemente des Arrays am Ende der Funktionsausführung sind von Position **p** bis Position **q** des Arrays gleich.

   b) Alle Elemente vor der Position **p** sind kleiner.

   c) Alle Elemente nach der Position **q** sind größer.

   Verbessern Sie mit der neuen **partition**-Funktion den Quicksort-Algorithmus.

   Analysieren Sie die Komplexität des Algorithmus, wenn alle Zahlen gleich sind.

4. Aufgabe (7 Punkte)

Programmieren Sie eine iterative (nicht rekursive) Variante des Mergesort-Algorithmus aus der Vorlesung, indem Sie nur ein Hilfsarray verwenden, um die Zwischen-Ergebnisse der Merge-Operationen zu speichern. Keine weiteren Listen oder Arrays sollen während das Sortieren produziert werden.

5. Aufgabe (6 Punkte)

a) Zeigen Sie mit Hilfe eines Zahlenbeispiels, dass der Heapsort-Algorithmus nicht stabil ist.

b) Was ist die Komplexität des Heapsort-Algorithmus, wenn die Zahlen in absteigender Reihenfolge sortiert sind? Begründen Sie Ihre Antwort.

c) a) Definieren Sie eine **is_max_heap**-Funktion, die ein Array/Liste mit Zahlen als Argument bekommt und überprüft, ob die Zahlen als heap-Struktur sortiert sind bzw., ob die Liste ein Heap ist.

Schreiben Sie getrennte Test-Funktionen für Ihren Tutor und verwenden Sie dabei die Funktionen **is_sort** und **gen_randlist**.

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Funktionalität der Funktionen darstellen.

2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

3) Kommentieren Sie Ihre Programme und schreiben Sie Hilfsdokumentation.

4) Verwenden Sie geeignete Hilfsvariablen und Hilfsfunktionen in Ihren Programmen.

5) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.

6) Schreiben Sie getrennte Test-Funktionen für alle 4 Aufgaben für Ihren Tutor.