Universität Stuttgart

Übungsblatt 2

Datenstrukturen und Algorithmen (SS 2022)

Abgabe: Montag, 02.05.2022, 15:30 Uhr — Besprechung: ab Montag, 09.05.2022

Abgabevorschriften: Die Aufgaben auf diesem Blatt sind unter Einhaltung der Abgabevorschriften[1] zu lösen und abzugeben. Achtung: manche Aufgaben auf diesem Blatt enthalten sowohl Implementierungs-, als auch Theorieaufgaben. Achten Sie deshalb auf die entsprechenden Kommentare in den Aufgaben.

Lernziele: Nach dem Tutorium zu diesem Blatt sollten Sie folgende Lernziele erreicht haben. Wenn nicht, zögern Sie nicht, ihre:n Tutor:in anzusprechen um die Lücken zu füllen, Unklarheiten zu klären oder Fragen zu beantworten.

- Sie können die Komplexität von Algorithmen mit Hilfe der $\mathcal{O}$-Notation einordnen.
- Sie können die Komplexität von Algorithmen in Form von Java-Code, Pseudocode oder Struktogrammen ermitteln.
- Sie können entscheiden, welches der schnellste (Sortier)Algorithmus für eine gegebene Situation ist.
- Sie können Algorithmen verbessern, sodass sie effizienter arbeiten.

Punkte: Dieses Übungsblatt beinhaltet 5 Aufgaben mit einer Gesamtzahl von 30 Punkten. Zum Bestehen werden also 15 Punkte benötigt.

Aufgabe 1 Asymptotische Komplexität von Funktionen *[Punkte: 4]*

In dieser Aufgabe werden Sie Funktionen Komplexitätsklassen zuordnen und das Verständnis für die Unterschiede zwischen diesen Komplexitätsklassen vertiefen.

(a) *(2 Punkte)* Zeichnen Sie die folgenden Funktionen in ein gemeinsames Koordinatensystem ein. Wählen Sie hierbei für das Koordinatensystem sinnvolle Achsenbereiche und -skalierungen, bei denen das Wachstumsverhalten der Funktionen sichtbar wird. Wir empfehlen hierfür die Verwendung einer geeigneten Software, die das Wählen geeigneter Achsenbereiche und -skalierungen erleichtert.
$f_1(n) = n$
$f_2(n) = n^2 log(n)$
$f_3(n) = n!$
$f_4(n) = log(n)$
$f_5(n) = 2^n$
$f_6(n) = n^3$

(b) *(2 Punkte)* Bestimmen Sie für jede Funktion aus Teilaufgabe a) die **asymptotische Komplexität** in $\mathcal{O}$-Notation und ordnen Sie diese **der Größe nach, beginnend mit der größten Komplexität**. Nennen Sie zu jeder Funktion zudem jeweils die entsprechende **Komplexitätsklasse** (z.B. linear oder quadratisch).

Aufgabe 2 Komplexität von Sortierverfahren *[Punkte: 3]*

Im Folgenden sind die schnellsten Sortierverfahren für unterschiedliche Anwendungsfälle gesucht. Markieren Sie für jeden Fall, welches Sortierverfahren aus der Vorlesung die gegebenen Daten am schnellsten (aufsteigend) sortiert. Falls es mehrere Verfahren gibt, die gleich schnell sind, markieren Sie alle. Begründen Sie außerdem Ihre Auswahl und geben Sie dabei zu jedem Anwendungsfall an, welche Komplexität das ausgewählte Verfahren besitzt.

(a) *(1 Punkt)* Fall 1: Gegeben sind absteigend sortierte Daten.
◯ BubbleSort ◯ InsertionSort ◯ MergeSort ◯ QuickSort ◯ SelectionSort

[1] https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_file_2904210.html

(b) *(1 Punkt)* Fall 2: Gegeben sind diese sortierten Daten → [5, 6, 8, 9, 11, 23, 32, 34, 58, 59, 123, 233, 436, 543].
◯ BubbleSort ◯ InsertionSort ◯ MergeSort ◯ QuickSort ◯ SelectionSort

(c) *(1 Punkt)* Fall 3: Gegeben sind unsortierte ("chaotische") Daten.
◯ BubbleSort ◯ InsertionSort ◯ MergeSort ◯ QuickSort ◯ SelectionSort

Aufgabe 3 Asymptotische Komplexität von Algorithmen *[Punkte: 7]*

Bestimmen Sie die *asymptotische Komplexität* der folgenden Algorithmen in *Abhängigkeit von n*. Dabei sei *n* jeweils eine *positive natürliche Zahl*. Begründen Sie Ihre Antwort kurz in maximal fünf Hauptsätzen. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

(a) *(1 Punkt)* Gegeben ist der folgende Algorithmus `alg1`:

```
n := beliebige Zahl
a := 1000
for i := 2*n to i < 0; i := i - 1
    a:= a + 1
for j := 0 to j < n; j := j + 2
    a:= a + 1
return a
```

(b) *(1 Punkt)* Gegeben ist der folgende Algorithmus `alg2`:

```
Input: Integer n
Output: Integer b
Integer a = 1000;
Integer b = 0;
for Integer i = 0; i < a; i++ do
    for Integer j = 0; j < a; j = j * 2 do
        b += (i * n);
    end
end
for Integer k = 0; k < a/2; k++ do
    b = b - i;
end
return b;
```

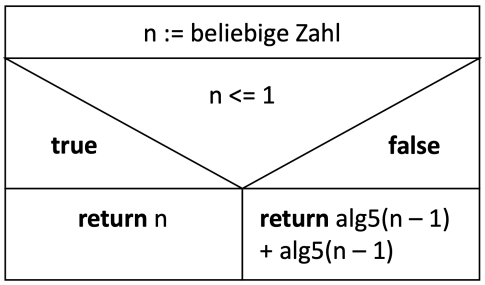
(c) *(1 Punkt)* Gegeben ist der folgende Algorithmus `alg3`:

```
1  public int alg3 (int n) {
2      int result = 10;
3      int a = 100
4      while (a>-1) {
5          for (int i=n;i>=2;i--) {
6              result++;
7          }
8      }
9      return result;
10 }
```

(d) *(1 Punkt)* Gegeben ist der folgende Algorithmus `alg4`:

Input: Integer n
Output: Integer result
Integer result = 1;
while *result* $<$ *n* **do**
  result = result * 2;
end
for *Integer i = 0; i* $<$ *result; i++* **do**
  for *Integer j = i; j* $\leq$ *result/2; j++* **do**
    result = 0;
  end
end
return result;

(e) *(1.5 Punkte)* Gegeben ist der folgende Algorithmus `alg5`:



(f) *(1.5 Punkte)* Gegeben ist der folgende Algorithmus `alg6`:

```
1  public int alg5(int n) {
2      int result = 0;
3      if (n > 1) {
4          result = alg5(n - 1) + 1;
5      } else {
6          result = 1;
7      }
8      for (int i = 0; i < n / 2; i++) {
9          result = result - 1;
10     }
11     return result;
12 }
```

Aufgabe 4 Impl Effizientere Algorithmen *[Punkte: 6]*

Manchmal kann man Algorithmen so verbessern, dass sie die gleichen Ergebnisse berechnen, aber um Komplexitätsklassen schneller sind. Im zugehörigen Eclipse-Projekt sind in dieser Aufgabe drei Algorithmen gegeben. Diese sind allerdings nicht sonderlich effizient realisiert. Sie sollen deshalb in dieser Aufgabe äquivalente, aber effizientere Algorithmen implementieren und mit Hilfe der asymptotischen Komplexitäten begründen, warum Ihre Algorithmen schneller sind.

Geben Sie den Code im Eclipse-Projekt ab. Ihre Begründungen, sowie bestimmten Komplexitäten sind allerdings in der PDF zusammen mit den Lösungen der anderen theoretischen Aufgaben abzugeben.

Hinweis: Um einen Algorithmus zu verbessern und zu verstehen kann es hilfreich sein, die Ergebnisse für $n = 1, 2, 3, \ldots$ *zu berechnen.*

(a) *(2 Punkte)* Im Eclipse Projekt finden Sie den Algorithmus `couldBeBetter1`. Implementieren Sie einen verbesserten Algorithmus `isDoneBetter1`, welcher für ein gegebenes n das gleiche Ergebnis zurückgibt wie `couldBeBetter1`, dieses aber schneller berechnet.

Bestimmen Sie die asymptotische Komplexität der beiden Ansätze und begründen Sie inwiefern Ihr Ansatz schneller ist.

(b) *(2 Punkte)* Im Eclipse Projekt finden Sie den Algorithmus `couldBeBetter2`. Implementieren Sie einen verbesserten Algorithmus `isDoneBetter2`, welcher für ein gegebenes n das gleiche Ergebnis zurückgibt wie `couldBeBetter2`, dieses aber schneller berechnet.

Bestimmen Sie die asymptotische Komplexität der beiden Ansätze und begründen Sie inwiefern Ihr Ansatz schneller ist.

(c) *(2 Punkte)* Im Eclipse Projekt finden Sie den Algorithmus `couldBeBetter3`. Implementieren Sie einen verbesserten Algorithmus `isDoneBetter3`, welcher für ein gegebenes n das gleiche Ergebnis zurückgibt wie `couldBeBetter3`, dieses aber schneller berechnet.

Bestimmen Sie die asymptotische Komplexität der beiden Ansätze und begründen Sie inwiefern Ihr Ansatz schneller ist.

Aufgabe 5 Impl BubbleSort auf verketteten Listen *[Punkte: 10]*

Einige Sortierverfahren wie BubbleSort führen bei der Ausführung zu vielen Vergleichen und damit auch zu vielen Zugriffen auf die Elemente. Deswegen hängt die Effizienz solcher Sortieralgorithmen zusätzlich zur asymptotischen Komplexität auch stark von der verwendeten Datenstruktur ab, und wie sie darauf angewendet werden. Dies werden Sie in dieser Aufgabe am Beispiel von BubbleSort und verketteten Listen selbst testen.

Geben Sie den Code im Eclipse-Projekt ab. Ihre textuellen Antworten (Teilaufgabe a) sind allerdings in der PDF zusammen mit den Lösungen der anderen theoretischen Aufgaben abzugeben.

(a) *(1 Punkt)* Die Komplexität von BubbleSort hängt von der verwendeten Implementierung ab. Betrachten wir hierfür zunächst eine BubbleSort Implementierung, die auf einer verketteten Liste realisiert wurde und diese sortiert, indem sie auf die Indizes der Elemente in der Liste zugreift, um sie zu vergleichen und zu tauschen. Erklären Sie, warum diese Implementierung in einer höheren Komplexitätsklasse liegt, als man sie aufgrund der in der Vorlesung besprochenen asymptotischen Komplexität von BubbleSort vermuten würde.

(b) *(9 Punkte)* Implementieren Sie nun eine effizientere Version des BubbleSort Algorithmus auf verketteten Listen, die nicht über Indizes auf die Elemente der Liste zugreift, um sie zu vergleichen und zu tauschen. Im Eclipse-Projekt sind folgende Codefragmente für eine einfache Liste gegeben:

- Schnittstelle `ISimpleListNode` und unvollständige Klasse `SimpleListNode`
- Schnittstelle `ISimpleList` und unvollständige Klasse `SimpleList`

Vervollständigen Sie zunächst die Klasse `SimpleListNode`, die das Interface `ISimpleListNode` implementiert. Vervollständigen Sie dann die Klasse `SimpleList`, die das Interface `ISimpleList` implementiert, die unter anderem eine `sort` Methode enthält.

Achten Sie dabei auf die zugehörigen Kommentare in den Interface-Klassen.