Dr. Andreas Abel
Dr. Ulrich Schöpp
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
22. November 2010

6. Musterlösung zur Vorlesung

Einführung in die Informatik: Programmierung und Software-Entwicklung

Allgemeine Warnhinweise: Ab sofort gibt es mindestens einen Punkt Abzug für jede Aufgabe, bei der die geforderten Dateien im falschen Format abgegeben werden. Es werden nur `.txt`, `.pdf` und `.java` akzeptiert; welche zusammen in *einem* `.zip` Archiv mit UniWorX abgeben werden. Dateien im falschen Format können gegebenenfalls komplett ignoriert werden (also 0 Punkte).

Es gibt ebenfalls Punktabzug, falls Java-Code und Text-Lösungen in einer einzelnen Datei abgegeben werden. Eine `.java` Datei sollte möglichst immer so wie sie ist kompilierbar sein. Kommentare zum Code sind erwünscht, aber verstecken Sie nicht die Lösung einer anderen (Unter-)Aufgabe in einem Kommentar.

Aufgabe 6-1. (4 Punkte) (Abgabeformat: `.txt` oder `.pdf`)

a) Berechnen Sie für alle angegebenen Ausdrücke eine möglichst gute Grössenordung in O-Notation.

i) $27n + \frac{n}{2} + 8$ **Lösung:** $O(n)$

ii) $(n^2 + n + 4)^2$ **Lösung:** $O(n^4)$

iii) $n(n+2)^3$ **Lösung:** $O(n^4)$

iv) $(6n^4 - 3n^2 + 36n)/(3n^2 + \frac{2}{3})$ **Lösung:** $O(n^2)$

v) $n^3 - 1000n^2 + 109$ **Lösung:** $O(n^3)$

vi) $2n + 0.00031415n^3$ **Lösung:** $O(n^3)$

vii) $n + \log n$ **Lösung:** $O(n)$

viii) $n \log n$ **Lösung:** $O(n \log n)$

ix) $n^3 + 3n \log n$ **Lösung:** $O(n^3)$

x) $2^n + n^2$ **Lösung:** $O(2^n)$

b) Ein Algorithmus benötigt exakt $f(n) := 7n^2 + 27n - 4$ Sekunden, um ein Problem der Größe n zu bearbeiten. Die Komplexität des Algorithmus wird üblicherweise mit $O(n^2)$ angegeben.

Berechnen Sie den jeweiligen Faktor der Wertsteigerung für die exakte Formel und für n^2 zwischen einem Problem der Größe 100 und 500. Vergleichen Sie die Faktoren der jeweiligen Wertsteigerung. Was fällt Ihnen auf? **Lösung:**

$$\frac{500^2}{100^2} = \frac{250000}{10000} = 25 \qquad \frac{7(500)^2 + 27(500) - 4}{7(100)^2 + 27(100) - 4} = \frac{1763496}{72696} \approx 24.26$$

Die Faktoren sind annähernd gleich!

c) Ein Algorithmus benötigt 7 Sekunden, um eine Eingabe der Größe $n = 100$ zu bearbeiten. Die folgende Tabelle soll Auskunft darüber geben, wie viele Sekunden der Algorithmus für Probleme der Größe $n = 200, n = 500$ und $n = 1000$ benötigt.

Vervollständigen Sie die Tabelle für die angegebenen Komplexitäten!

n	$O(n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
100	7	7	7	7
200	14	**Lösung:** 28	**Lösung:** 56	**Lösung:** $7 \cdot 2^{100} \approx 7 \cdot 10^{30}$
500	35	**Lösung:** 175	**Lösung:** 875	**Lösung:** $7 \cdot 2^{400} \approx 7 \cdot 10^{120}$
1000	70	**Lösung:** 700	**Lösung:** 7000	**Lösung:** $7 \cdot 2^{900} \approx 7 \cdot 10^{270}$

Hinweis: Für die letzte Spalte reichen gerundete Werte.

Aufgabe 6-2. (4 Punkte) (Abgabeformat: `.txt` oder `.pdf`)

a) Berechnen Sie genau wie in Aufgabe 4-1 die Ausgabe des hier rechts abgedruckten unkommentierten Programms mit Papier & Bleistift. Geben Sie wieder Schritt für Schritt in einer Tabelle an, welche Variable in welcher Programmzeile welchen absoluten Wert zugewiesen bekommt. Markieren Sie mit einem Kommentar in Ihrer Tabelle zusätzlich die Schritte, in denen ein neuer Schleifendurchgang oder Methodenaufruf begonnen wird.

b) Geben Sie für jedes Mal, wenn die Programmausführung Zeile 370 erreicht, den vollständigen Stack- und Heap-Speicher an, wie in der Vorlesung demonstriert.

c) Begründen Sie warum in der Ausgabe des Programms unterschiedliche Werte ausgegeben werden, obwohl die Berechnung für beide Variablen doch anscheinend gleich ist, d.h. warum enthalten die Variablen x und $y[0]$ am Ende unterschiedliche Werte?

```
100 public class Uebung6-2 {
110
120 public static void main(String[] args) {
130     int x = 23;
140     int[] y = { 23 };
150     while (x < 333 && y[0] < 333) {
160         x *= 3;
170         y[0] *= 3;
180         foo(x, y);
190         x--;
200         y[0]--;
210     }
220     System.out.print("Werte: x=" + x + ", ");
230     System.out.println("y=" + y[0] + ".");
240 }
250
260 public static void foo(int x, int[] y) {
270     for (int i = 4; i < 6; i++) {
280         x *= 3 * i - 2;
290         y[0] *= 3 * i - 2;
300         bar(x, y[0]);
310     }
320 }
330
340 public static int bar(int x, int y) {
350     x /= 2;
360     y /= 2;
370     return x;
380 }
390 }
```

Lösung:

a) Das Programm gibt die Antwort "Werte: x=68, y=8969." aus.

Zeile	Zuweisung	Kommentar
130	$x = 23$	
140	$y[0] = 23$	
160	$x = 69$	
170	$y[0] = 69$	
270	$i = 4$	foo, 1.Schleife
280	$x = 690$	
290	$y[0] = 690$	
350	$x = 345$	bar
360	$y = 345$	
300	$(x = 690)$	Alter Wert
300	$(y[0] = 690)$	Alter Wert
270	$i = 5$	bar, 2.Schleife
280	$x = 8970$	
290	$y[0] = 8970$	
350	$x = 4485$	bar
360	$y = 4485$	
300	$(x = 8970)$	Alter Wert
300	$(y[0] = 8970)$	Alter Wert
270	$i = 6$	Schleifenabbruch
180	$(y = 69)$	Alter Wert
190	$x = 68$	
200	$y[0] = 8969$	

b) Beim ersten Erreichen von Zeile 370 gilt:

Stack:

`y`	`int`	345
`x`	`int`	345
`i`	`int`	4
`y`	Zeiger in den Heap	Addresse1
`x`	`int`	690
`y`	Zeiger in den Heap	Addresse1
`x`	`int`	69

Heap:

Addresse1	`int[]`	690

Dabei ist Addresse1 ein beliebiges Label, das in diesem Fall auf ein `int`-Array der Länge 1 zeigt.

Die Reihenfolge der Variablen innerhalb eines Blocks kann im Debugger manchmal variieren. Die Reihenfolge der Blöcke ist jedoch fest, wobei der unterste Block der älteste Block ist. Die Blöcke werden auch Stack-Frames genannt.

Beim zweiten Erreichen von Zeile 370 gilt:

Stack:

`y`	`int`	4458
`x`	`int`	4458
`i`	`int`	5
`y`	Zeiger in den Heap	Addresse1
`x`	`int`	8970
`y`	Zeiger in den Heap	Addresse1
`x`	`int`	69

Heap:

Addresse1	`int[]`	8970

c) Der Grund für das unterschiedliche Ergebnis trotz gleicher Operationen liegt darin, das `x` einen einfachen `int`-Wert auf dem Stack repräsentiert, während `y` eine Referenz auf einen `int`-Wert im Heap ist.

Ein Methodenaufruf in Java kann die Werte der Parameter nicht dauerhaft verändern, da eine Methoden nur mit Kopien der Parameter arbeitet. Der Grund ist die Call-by-Value Parameterübergabe (siehe die Folien über Prozeduren (Folien 11 & 12) sowie den aktuellen Foliensatz).

Im Methodenaufruf `foo(x,y)` wird also nur mit Kopien der Variablen `x` und `y` gearbeitet. Der Wert von `y` ist aber eine Referenz auf ein Array im Heap. Obwohl sich die Referenz selbst durch den Methodenaufruf nicht ändert, kann sich das Array im Heap, welches referenziert wird, schon verändern. So kommt die unterschiedliche Ausgabe zustande.

Im Aufruf `bar(x,y[0])` sind die Parameter dagegen beide einfache `int`-Werte und es werden dementsprechend in der Methode nur Kopien verändert, welche am Ende der Methode verworfen werden. Die Anweisung `return x` ermöglicht zwar die übernahme des neuen Wertes für `x`, aber im Aufruf in Zeile 300 wird dieser Rückgabewert ja verworfen, da dort nur `bar(x,y[0]);` steht und nicht etwa `x = bar(x,y[0]);`

Aufgabe 6-3. (4 Punkte) (Geben Sie alle `.java`-Dateien Ihrer Lösung ab)

Gegeben ist eine beliebige Menge von gefärbten Punkten in der Ebene. Schreiben Sie ein Programm, welches die gesamte Ebene einfärbt und grafisch ausgibt. Dabei soll jedem Punkt der Ebene die Farbe des nächstegelegenen Punktes der gegebenen Menge zugewiesen werden. Den Abstand zweier beliebiger Punkte mit den Koordinaten (x_1, y_1) und (x_2, y_2) definieren wir gemäß der euklidischen Metrik, d.h. der Abstand berechnet sich einfach immer nach dieser Formel

$$\text{Distanz} := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Verwenden Sie das auf der Homepage bereitgestellte Programm Gerüst `Aufgabe6_3.java`, welches Ihnen die Grafik-Handhabung vollständig abnimmt. Füllen Sie darin lediglich die Stelle aus, welche mit "`IHRE AUFGABE`" markiert ist. Damit das funktioniert, müssen Sie die Datei `CanvasFrame.java` aus Aufgabe 3-2 ins gleiche Verzeichnis wie `Aufgabe6_3.java` hineinkopieren.

Zur Erklärung: Das Programmgerüst öffnet zu Beginn ein Grafikfenster mit 500×500 Punkten. Dieses Grafikfenster wird über die Variable `frame` gesteuert, was Sie aber getrost ignorieren dürfen, denn zum Färben eines einzelnen Punktes ist Ihnen die Methode `drawPoint` vorgegeben. Dieser Methode müssen Sie beim Aufruf lediglich die Variablen `frame` (welche immer unverändert bleibt), sowie drei `int`-Werte für x-Koordinate, y-Koordinate und die Farbe übergeben. Sie müssen sich also nur um die Berechnung dieser drei Zahlwerte kümmern!

Im Programmgerüst sehen wir, dass die Menge der gegebenen Punkte als 2-dimensionales Array, also vom Typ `int[][]` unter dem Namen `voronoi` gegeben ist. Das bedeutet, das Sie mit `voronoi[0]` dem ersten Punkt der Menge erhalten, und mit `voronoi[1]` den zweiten, usw. Den ersten Wert des i-ten Punktes, also `voronoi[i][0]`, sollen Sie als x-Koordinate interpretieren; den zweiten Wert als y-Koordinate, und den dritten als die Farbe des Punktes. Den Farbwert müssen Sie nicht interpretieren, sondern nur unverändert an die Methode `drawPoint` durchreichen.

Hinweis: Bevor Sie in Java wild drauflos-programmieren, sollten Sie zuerst Ihren Algorithmus abstrakt niederschreiben. Falls Sie dabei Probleme haben, denken Sie besser nicht an technische Probleme, wie beispielsweise Array-Zugriffe, sondern erst einmal daran, wie Sie vorgehen würden, wenn Sie die Anweisungen mit Papier & Buntstiften ausführen würden. Dabei sollten Sie jedoch beachten, das einem Computer im Gegensatz zu den meisten Menschen endlose Wiederholungen nicht langweilen, d.h. das für einen Computer in ein einfacherer, aber repetitiver Algorithmus oft ausreichend. Beispielsweise könnte ein Mensch auf die Idee kommen, erst die Grenzen der Farbflächen zu berechnen, und dann die entstandenen Flächen ausfüllen. Dies wird aber vermutlich viel schwerer zu implementieren zu sein, als einfach Punkt für Punkt jeweils einzeln die Farbe zu berechnen und den Punkt gleich anzumalen.

Lösung: Siehe `Aufgabe6-3Loesung.java`.

Abgabe: Sie können ihre Lösungen bis Montag, den 29.11., 14 Uhr über UniWorX abgeben. Java Dateien, welche nicht kompilieren, werden nicht beachtet!