Albert-Ludwigs-Universität Freiburg
Informatik 1
Wintersemester 2011/12

Prof. Dr. Wolfram Burgard
Dipl. Ing. Jörg Röwekämper
Dipl. Inf. Markus Kuderer

Übungsblatt 6

Abgabe bis Donnerstag, 8.12.11, 12:00 Uhr

Hinweis: Kompilieren und testen Sie Ihre Programme mit Hilfe des `ant`-Buildsystemes. Schicken Sie Ihrem Tutor eine Kopie des kompletten Projektordners, der alle benötigten Dateien enthält.

Aufgabe 6.1

1. Erstellen Sie eine Klasse `MyVectorClass`, die einen Integer-Vektor[1] enthält und erzeugen Sie das Vektor-Objekt in dem Konstruktor.

2. Auf der Vorlesungshomepage finden Sie die Datei `pi.txt` in der die ersten 100 Stellen der Zahl π aufgelistet sind. Implementieren Sie eine Java-Methode, die jede Ziffer einzeln in den Integer-Vektor einfügt. Sie können davon ausgehen, dass die Datei nur Zahlen enthält (keine Sonderzeichen oder Buchstaben).

3. Implementieren Sie eine Java-Methode `void toggleSign()`, die das Vorzeichen aller Elemente dreht, deren Index durch 2 teilbar ist, d.h. $i \in \{0, 2, 4, ..\}$.

4. Implementieren Sie die Methode `int computeSum(int begin, int end)`, die die Summe

$$\sum_{i=\text{begin}}^{\text{end}} v.elementAt(i)$$

   zurückgibt, wobei `v` das Vektor-Objekt ihrer Klasse ist.

5. Implementieren Sie die Methode `int maxSum()`, die die maximale Summe aller Elemente in einer zusammenhängenden Teilfolge des Integer-Vektors berechnet. Beispiel: In dem Vektor $[-3, 2, 1]$ gibt es die Teilfolgen:

$$[-3] \quad [2] \quad [1] \quad [-3, 2] \quad [2, 1] \quad \text{und} \quad [-3, 2, 1].$$

   Die maximale Summe ist somit $2 + 1 = 3$.

6. Schreiben Sie einen `JUnit`-Test, der Ihre Klasse testet. Die Methode `maxSum` sollte für den in Teilaufgabe 2 eingelesenen und durch `toggleSign` modifizierten Vektor das Ergebnis 17 liefern.

7. Bestimmen Sie die `Best-Case` und `Worst-Case`-Komplexität Ihres Algorithmus `maxSum` in Abhängigkeit der Länge des Vektors.

[1] http://docs.oracle.com/javase/6/docs/api/java/util/Vector.html

Aufgabe 6.2

Die folgende Methode `findDuplicates` identifiziert Duplikate in einem String-Vektor.

```
static void findDuplicates(Vector<String> v) {
int n = v.size();
for (int i = 0; i < n-1; i++) {
  for (int j = i+1; j < n; j++) {
    if (v.elementAt(i).equals(v.elementAt(j))) {
      System.out.println("Duplikat");
    }
  }
}
```

1. Führen Sie für `findDuplicates` eine Aufwandsabschätzung für die Laufzeit in Abhängigkeit von der Länge n des Vektors `v` durch. Gehen Sie davon aus, dass die Methoden `elementAt()`, `equals()` sowie `println()` jeweils konstante Laufzeit haben.

2. Implementieren Sie eine Methode
   `boolean hasDuplicates(Vector<String> v)`,
   indem Sie obige Methode so modifizieren, dass genau dann `true` zurückgegeben wird, wenn der Vektor ein Duplikat enthält. Wie verändert sich die `Best Case` und `Worst Case` Laufzeit im Vegleich zu der Methode `findDuplicates`?

Aufgabe 6.3

Betrachten Sie folgende Tabelle. In welche Komplexitätsklassen gehören die Funktionen $f_i(n)$ jeweils? Beachten Sie, dass eine Funktion auch mehreren Klassen angehören kann.

	$O(n)$	$O(n^4)$	$O(\log(n))$	$O(2^n)$	$O(4^n)$
$f_1(n) = \sqrt{n} + \log(n) + n^3$					
$f_2(n) = n + \sqrt{n}$					
$f_3(n) = 2^{n+2}$					
$f_4(n) = 2^{n+n} + 1000$					
$f_5(n) = 3^n$					
$f_6(n) = 7777 \cdot n^4 + n^3$					
$f_7(n) = \frac{1}{n}$					