**Homework 1**

**Software Design Decisions and Decomposition**

Due to: **January 22, 2018**

This document is composed of two sections. The first section titled “Review” covers some basic materials that were talked in the class. The details are NOT presented, and you need to study more details for answering the questions. It is just to let you know what you are required to know. The second section includes some questions for you to answer and argue about your reasons. Please note that all the questions are based on a piece of code called “Eval sort”. The code is attached to the end of this document and also the link to the java file (the same as the attached code) is available in the assignment page on Canvas.

Please note that the answers MUST be typed and submitted through Canvas before the deadline. This assignment will be graded.

**1. Review: Read before answering the questions**

***1.1. Design paradigms***

As we discussed in the class, one basic conceptual model of software is a step-by-step recipe. Design paradigms that encourage this recipe-like thinking are more formally known as imperative design paradigms. Main idea of an imperative design: describe a conceptual solution by writing down its steps.

* Steps are thought of as taken in order
* Each step may change the states of the software
* Steps describe exactly what to do

|  |
| --- |
| Step 1 → Step 2 → Step 3 → … |

There are different more sophisticated imperative design paradigms that offer complementary strategies to decompose and structure concepts into manageable chunks. These paradigms are:

* Procedural design: define conceptual model in terms of procedures, which consists of steps and/or other procedures.
* Module-based design: define conceptual model in terms of module based on functionality.
* Object-oriented design: define conceptual model in terms of entities and instances of these entities (objects).

***1.2. Decomposition***

By Decomposition techniques, or more precisely software decomposition techniques, we mean practices that help us split a software development project into smaller chunks.

When a software development project is large and complex, we need to be able to decrease its complexity to a manageable level so that mere mortals can work on the project. Decomposition is a key technique in software design and development to achieve that.

Modularization is the process of decomposing the software into modules. When the use of the term does not cause any confusion the term “module” can also be used instead of “component”, “class”, etc. For example, decomposing the software into object-oriented classes is also called modularization where classes act as the module.

**2. Questions:**

**Suggest five changes that you think users might want to make to the eval sorts code. For each of those changes, identify all of the lines in the original code that would need to be changed. (For each change, make a copy of the code and highlight the spots that need to change. )**

**We are expecting you to suggest changes similar to, but different than these examples:**

1. Users don’t like typing numbers for yes and now. What would need to change for them to be able to type ‘y’ or ‘n’? (Discuss)
2. We need to have an extensible version of “eval sort”. It means, we need a code that we can easily add other sort algorithms to it in future. To provide such this scalability, what parts of code should be changes? (Discuss)

You should assume the following methods are *not* modifiable by you (i.e., they are meant to simulate primitives in the language.)

int getInput(); // returns next input from standard input.

void prompt(String query); //prints query to the console.

String getFileName(); //returns the next string token from standard input.

int[] readFile(String fname); //returns an array of ints read from file named fname.

                                            // assumes the first int is a count of data items (ints to be read)

int[] makeCopy(int[] source); //returns a new array which is a copy of source

int[] insertionSort(int[] data); // sorts data and returns a reference to data.

int[] selectionSort(int[] data); // sorts data and returns a reference to data.

int[] mergeSort(int[] data); // sorts data and returns a reference to data.

int[] qSort(int[] data); // sorts data and returns a reference to data.

void setRandomSeed(long seed) // initializes the random number generator.

int nextRandomInt(); // returns a random integer

long getTime(); // returns the current system clock (in milliseconds);

The Program

package edu.iastate.cs362.imperative;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Random;

import java.util.Scanner;

public class EvalSorts {

public static void main(String[] args) throws FileNotFoundException {

long seed;

int length;

long startTime;

long endTime;

do {

// input data source

prompt("Data source? (0 = file, 1 = generated)");

if (getInput() == 0L){

//get data from file

String filename;

int[] fileMaster ;

int[] fileData ;

//process data from file

//input file name

prompt("File name? ");

String fname = getFileName();

fileMaster = readFile(fname);

// perform insertion sort

fileData = makeCopy(fileMaster);

startTime = getTime();

insertionSort(fileData);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "insertion Sort", fileData.length, endTime-startTime);

// perform selection sort

fileData = makeCopy(fileMaster);

startTime = getTime();

selectionSort(fileData);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "selection Sort", fileData.length, endTime-startTime);

// perform merge sort

fileData = makeCopy(fileMaster);

startTime = getTime();

mergeSort(fileData);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "merge Sort", fileData.length, endTime-startTime);

// perform quick sort

fileData = makeCopy(fileMaster);

startTime = getTime();

qSort(fileData);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "quick Sort", fileData.length, endTime-startTime);

} else {

int[] randomData;

int[] ranCopy;

// get experiment parameters

prompt("How many values?");

length = getInput();

prompt("What seed? 0 = default ");

seed = getInput();

// generate random data

randomData = new int[length];

setRandomSeed(seed);

for (int i = 0; i < length; i++){

randomData[i] = nextRandomInt();

}

// perform insertion sort

ranCopy = makeCopy(randomData);

startTime = getTime();

insertionSort(ranCopy);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "insertion Sort", ranCopy.length, endTime-startTime);

// perform selection sort

ranCopy = makeCopy(randomData);

startTime = getTime();

selectionSort(ranCopy);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "selection Sort", ranCopy.length, endTime-startTime);

// perform merge sort

ranCopy = makeCopy(randomData);

startTime = getTime();

mergeSort(ranCopy);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "merge Sort", ranCopy.length, endTime-startTime);

// perform quick sort

ranCopy = makeCopy(randomData);

startTime = getTime();

qSort(ranCopy);

endTime = getTime();

System.out.format("%20.20s %10d %10d %n", "quick Sort", ranCopy.length, endTime-startTime);

}

prompt("Another experiment? (0 = no, 1 = yes) ");

} while (getInput() == 1L);

}

private static Scanner scan = new Scanner(System.in);

public static int getInput(){

return scan.nextInt();

}

public static String getFileName(){

return scan.next();

}

public static void prompt(String query){

System.out.println(query);

}

public static int[] readFile( String fname) throws FileNotFoundException{

Scanner fscan = new Scanner(new File(fname));

int numEntries = fscan.nextInt();

int[] data = new int[numEntries];

for (int i = 0; i < numEntries; i++){

data[i] = fscan.nextInt();

}

fscan.close();

return data;

}

public static int[] makeCopy(int[] source){

int[] retData = new int[source.length];

for (int i = 0; i < source.length; i++){

retData[i] = source[i];

}

return retData;

}

public static int[] insertionSort(int[] data){

randomDelay();

return data;

}

public static int[] qSort(int[] data){

randomDelay();

return data;

}

public static int[] mergeSort(int[] data){

randomDelay();

return data;

}

public static int[] selectionSort(int[] data){

randomDelay();

return data;

}

public static Random generator ;

private static void setRandomSeed(long seed) {

generator = new Random(seed);

}

private static int nextRandomInt() {

return generator.nextInt();

}

private static long getTime() {

return System.currentTimeMillis();

}

public static void randomDelay(){

try {

Thread.sleep(generator.nextInt(30));

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}