Assignment 2

CSCI 2010U - Principles of Computer Science

Due: 25 October 2013 at Midnight

Introduction

This assignment focuses on sorting plus empirical algorithm evaluation. You will experiment with the following three algorithms:

1. Bubble sort;
2. Quick sort; and
3. Merge sort.

Sorting

In this section you will need to implement Bubble sort, Quick sort and Merge Sort algorithms by extending a base abstract class. You will start with the following abstract class

```
public abstract class ArraySort<T> {
      ...
      abstract public void iSort(T[] inArray);
      abstract public T[] oSort(T[] inArray);
      public long iSortTimed(T[] inArray) { ... }
      public void setComparator(Comparator<T> comparator) { ... }
}
```

You must 1) complete this class and 2) extend this abstract class (to implement the three sorting algorithms) by implementing its abstract methods.

Method iSort

Performs in-place sorting. The input array is replaced by a sorted version.

Method oSort

Performs out-of-place sorting. It preserves the input array, and returns a new sorted array.

Method iSortTimed

Performs in-place sorting and records the time it took to perform this sorting.

Method setComparator

Sets the comparator used by the sorting procedures.

Comparator Interface

It is important to emphasize that it is expected that your implementations will work with generic data type (<T>), so you need to provide a mechanism of comparing two generic values. You will implement a Comparator interface for the generic types used in your program.

```
public interface Comparator<T> {
      public int compare(T a, T b);
}
```

The only method that requires implementation is compare(). The return value of compare(a,b) is defined as: −1 if a<b; 0 if a==b; and +1 otherwise.

Implementation Goals

- Implement the abstract class ArraySort
- Extend the abstract class ArraySort to implement BubbleSort, MergeSort and Quicksort.
- Implement the comparator interface for the following types: Integer, String, and Float.
    - The comparator implementations must also accept a Boolean flag indicating whether to compare in ascending order or descending order, e.g. the constructor for IntegerComparator looks like

```
public IntegerComparator(boolean ascend)
```

Performance Evaluation Goals

Write a program sortNumbers which generates an array of random integers from 0 to 100 and performs in place sorting, recording the time it takes to complete the sorting. You can use method iSortTimed to record the times. We will use your program as follows

java sortNumbers <array length> <method> [<order>]

Here <array length> is a positive integer between 1 and 10000000 and method is either “bubble”, “merge” or “quick”, indicating which algorithm to choose. <order> is either “ascend” or “descend”. <order> is optional and when absent its value is assumed to be “ascend”.

If the <array length> is less than 10 you must also print the contents of the array (both before and after sorting) on the screen.

Here are some sample uses:

java sortNumbers 4 bubble ascend
1 99 23 14
1 14 23 99
0.00001

java sortNumbers 100000 bubble
0.341

Carry out the necessary tests to complete the following table

	Average times over 5 runs each		
Array length	Bubble sort	Quick sort	Merge sort
5			
20			
100			
500			
1000			
2000			
10000			
100000			
500000			
1000000			

Write down NC if your program is taking too long to sort an array of a given length. Say if your program is taking more than 5 hours to sort a 1000000 size array.

Submission

Please follow these instructions to the letter. Thank you.

Create a zip file called A2_<studentnumber>.zip with the following two files: sortNumbers.java and a2.pdf

- sortNumbers.java is your implementation. This file contains all of your code.
- a2.pdf contains the filled table

Both files should contain your name and student number at the top.