Programming Project 4: Benchmarking Sort Algorithms

Due date: May 3, 11:55PM EST.

You may discuss any of the assignments with your classmates (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating.

In this assignment you will evaluate relative performance of different sorting algorithms. We discussed them all in class and several different versions of code are in the textbook, OpenDSA website and other resources that you are using. Your job is to design a benchmark program for performance evaluation.

Problem Description

We spent some time discussing performance of the algorithms, but in practice what one often wants to test is how a given implementation performs on a given machine. You will write a several implementations of the sort algorithms and evaluate their performance on your own computer.

Problem Solution

The outline of your program should be as follows:

- Validate the command line arguments that provide parameters for the current run of your benchmark program.
- Generate two input arrays (using the parameters specified on the command line).
- For each of the implemented sort algorithms
  - make a copy of each input array,
  - start the timer
  - run the sort algorithm on first array
  - stop the timer and display the running time
  - start the timer
  - run the sort algorithm on the second array
  - stop the timer and display the running time.

Implementation Details

Command Line Arguments

The program should take two command line arguments: the number of elements in the list to be sorted and an ordering type. The first command line argument should be an integer value indicating the size of the arrays. The second command line argument should be a single character: A, D, or R indicating that the elements in the array before it is sorted should be organized in the ascending, descending, or random order respectively. Your code has to verify correctness of the command line arguments before using them.
Generating Data

The program needs to generate two different arrays that will be used for sorting using different methods.

The first array should be of type Integer (using the wrapper class, so you can use the generic sort methods).

The second array should be of type MyBigObject (the class that you need to implement, see below for more details). Both arrays should be either in ascending, descending or random order as indicated by the command line argument. (Yes, that means that sometimes your program will be sorting an array of objects that are already in the sorted order.)

Before running each of the sort methods on the array, you should make a copy of it and run the algorithm on the copy. You have to make sure that each algorithm starts with the exact same array as the previous one. You can use `System.arraycopy()` if you wish.

MyBigObject class

This class is used to evaluate if the size of the object itself influences overall performance of the sort algorithm. The objects or class Integer are very small (not much larger than the primitive type integer). The objects of class MyBigObject are much larger.

```java
public class MyBigObject implements Comparable<MyBigObject> {
    static Random rand = new Random(0);
    private int i;
    private double[] d;

    public MyBigObject() {
        i = (rand.nextInt(500) + 100);
        d = new double[rand.nextInt(1000) + 100];
    }

    public MyBigObject(int size) {
        i = size;
        d = new double[rand.nextInt(1000) + 100];
    }

    @Override
    public int compareTo(MyBigObject other) {
        if (this.i < other.i) return -1;
        if (this.i > other.i) return 1;
        // otherwise determine by the size of the StringBuffer object
        if (this.d.length < other.d.length) return -1;
        if (this.d.length > other.d.length) return 1;
        // otherwise, they are equal
        return 0;
    }
}
```

Sorting and Timing

The program needs to apply sort methods to each of the two arrays. Make sure you are making a copy of the array before it gets sorted so you are always starting from the same array for all different algorithms. Your need to test the following sort methods:

- bubble sort
- selection sort
- merge sort
- quick sort
- Java’s Arrays.sort() method
A single run of a program should run each of these four methods on the Integer array and then on the MyBigObject array. The goal is to time how long it takes for each sort to complete. Use `System.nanoTime()` method to get the time before and after the call to the sort function. The difference between the two will tell you how many nanoseconds it took for the algorithm to complete its task. Examples of how to time various operations and display the results are in the code that was posted for the previous programming project.

**Tricks to increase timing accuracy:**

A. Create a small dummy array and run each algorithm with that dummy array before running it with the actual array for which you are timing. Java does not bring all the code into the memory at the beginning and you do not want your timing to include the loading of the code.

B. Run Java’s garbage collector manually using `System.gc()`. This decreases the chances of it running automatically during the sorting operation.

C. Run the code on the same size arrays multiple times and take an average or median of the times that you record. This way if something else happens to run on your computer during the timing, you do not use it to affect your results.

A call to one of the sorts should look similar to what follows:

```java
long start, end;
Integer[] dummy = new Integer {1, 2, 3, 4, 5};
...
// make copy of the actual array used for testing, call it bArray
bubblySort( dummy );
System.gc();
start = System.nanoTime();
bubbleSort( bArray );
end = System.nanoTime();
```

Your program should display the time that the algorithm took in milliseconds and then in seconds (again, see the previous assignment for an example of how this is done).

**Report**

You need to write a short report based on multiple runs of your program. For each type of ordering of the initial array (ascending, descending, random) and for each type of the array (Integer or MyBigObject), you should run your program using arrays of different sizes: 10, 50, 100, 500, 1 000, 5 000, 10 000, 50 000, 100 000, 200 000 (the last few values may take some time, so be patient).

Your report should contain six tables and three graphs/plots (one for each type of initial ordering and type of the array) showing performance of each algorithm on different sizes of arrays (the number of elements in the array should be on the horizontal axis, the time used should be on the vertical axis).

In addition, for each of the six table-plot combinations you should write a short paragraph describing what you observed. If you ended up revising the code for any of your algorithms based on the observations, include that in your report as well.

**Working on This Assignment**

You should start right away! The code itself is not hard to develop, but you may want to spend some time optimizing and fine-tuning you implementations. You also need to complete your code a few days before the assignment is due so that you can prepare the report.
Extra Credit (20 points)

The textbook describes another sorting algorithm called heap sort. It is based on the data structure that we did not cover called heaps. Read and study the material on your own and add the heap sort to your program and to your report.

Grading

30 points - completeness and correctness of the report
15 points - code documentation (using Javadoc)
25 points - development and correctness of the application (parsing and validating the command line arguments, setting up the arrays, running and timing of the different sort implementations)
30 points - development and correctness of sorting methods

How and What to Submit

Your should submit the report as a PDF document and all of the source code files (the ones with .java extensions only) in a single zip file to NYU Classes.

Make sure that all the code is properly documented (using Javadoc). All files have to contain a preamble (or class documentation) that contains your name. All the code should be your own. This is not a collaborative project.