CMPE 221_223_242
Fall 2022
Programming Homework 2

This assignment is due by 23:59 on Sunday, 27 November 2022.

You are welcome to ask your HW related questions. You should use only one of these options:

1. Moodle Homework **Question Forum**: HW Question-and-Answer (Q&A) Forum on Moodle is always available. Use the "Forum" link at the course Moodle page.
2. Homework **RECITATION HOURS**: There will be two Q&A RECITATION HOURs on the following days:

CMPE223-HW1-OfficeHour1: 14 Nov, 19:30-21:30 PM, Zoom ID: 926 1140 6270
CMPE223-HW1-OfficeHour2: 21 Nov, 19:30-21:30 PM, Zoom ID: 926 3491 8019

Note: Please make sure that you have read the HW document well before participating. However, no HW related questions will be accepted except from the above options.

PART I (30 points)

In this part of the assignment, you are asked to write different sorting algorithms. The goal of the homework is to exercise the details of sorting algorithms and have hands-on experience with programming with sorting APIs.

IMPORTANT: The homework consists of steps, and you should follow them sequentially, i.e. do not go to the next step before fixing this step. It is recommended that you complete one step per day.

Here are the steps of the homework:

Step 1. In this step, first write a program (i) that loads N integers from a file and (ii) that creates a random array and uses the insertion sort algorithm on Page 251 of the textbook. (In the text file, the first line in the input file should be the size of the array, and each following line should consist of a single integer).

Step 2. In this second step, modify this insertion sort algorithm such that it sorts the input integer array in **descending** order, and start sorting the array **FROM-RIGHT-TO-LEFT**.

Step 3. Use the `Insertion sort` algorithm on Page 251, now to sort an array of **double** values instead of integers.

Step 4. Modify the top-down `Merge Sort` algorithm on page 273 to sort the input integer array in **descending** order**.**

Step 5. In this step, create a `Route` class. Each route should have two fields: a `String` attribute, called *source*, and a second `String` attribute called *destination*. The `Route` class should implement the `Comparable` interface to be able to compare `Route` objects with respect to their source and destination. The `Route` objects are first ordered (alphabetically) with respect to their source. If their sources are the same, they are ordered (alphabetically) with respect to their destinations. Check the example on Page 247 of the textbook for implementing the `Comparable` interface.

Step 6. Use the `Quick Sort` method on Page 289 to sort the given `Route` objects. For this purpose, create 10 different `Route` objects, each with a *source* and *destination* from of one these four cities: Ankara, Istanbul, Antalya, and Izmir. Then, call this method to sort these objects with respect to source and destination.

Step 7. In this step, modify the `Quick Sort` partition method on Page 291 to sort the elements in **descending** order.

Step 8. In this last step, assume that (recursive) `Quick Sort` method receives an `int` parameter `depth`; from the main driver that is initially approximately 2 log N. Modify this recursive `Quick Sort` to call `Merge Sort` on its current subarray if the level of recursion has reached depth. (**Hint**: Decrement depth as you make recursive calls; when it is 0, switch to `Merge Sort`.) Call this new function to sort the elements in **descending** order.

You do not need to prepare a PDF report, just document your code well.

PART II (70 Points)

In this part of the homework, first, you are expected to write a static method that takes an array of n integers and prints the smallest pairwise absolute difference between them along with the corresponding pair of numbers.

Your solution is expected to be composed of two steps:

- Sort the given array.
- Print **the smallest pairwise absolute difference** along with the corresponding pair of numbers. The running time complexity of this step has to be *O(n).*

If there are more than one pair with the smallest absolute difference, only the one whose sum is the smallest is printed.

Example input and outputs should be as follows:
->There is one pair with the smallest absolute difference
Input: 23, 1, 5, 102, 34, 99
Output: 3 [99 102]

->There are more than one pair with the smallest absolute difference, only the one whose sum is the smallest is printed
Input: 23, 1, 4, 102, 34, 99
Output: 3 [1 4]

Input: 113, 23, 1, 109, 4, 102, 7, 105, 100, 107
Output: 2 [100 102]

After writing your method as described above, you are expected to make its experimental run time analysis in the four different settings where in each setting you are going to use a different sorting algorithm:

- Setting 1: `Selection sort` (use the algorithm on Page 249 of the textbook)
- Setting 2: `Insertion sort` (use the algorithm on Page 251 of the textbook)
- Setting 3: `Merge sort` (use the algorithm on Page 273 and 271 of the textbook)
- Setting 4: `Quick sort` (use the algorithm on Page 289 and 291 of the textbook)

Note: To make your work easier while testing your method in four different settings, you could design your method in a way that it takes an additional input specifying which sorting algorithm is to be used.

In this way, you are going to observe and report how the performance of your method changes when different sorting algorithms are used. Based on the experimental results that you obtain, comment on the complexity of your method using each sorting method.

While doing experimental analysis of the method in each setting, you are expected to run it for **different sizes** of **ascending/descending** and **random ordered** integer arrays.

1. To make experimental analysis of your method, create a new *SortingAlgorithmTester* class, and within the main method create an array with (i) ascending, (ii) descending, or (iii) random integers, to be sorted with each algorithm. You can get a random list using the java class Random, located in *java.util.Random.*

2. As you test your method in different settings, collect time measures. Time is measured in milliseconds. You should use the `System.currentTimeMillis()` method, which returns a long that contains the current time (in milliseconds). It is recommended that you do several runs of your method on a given array and take the median value (throw out the lowest and highest time). In order to get the most accurate runtimes possible, you may want to close other programs that might be taking a lot of processing power and memory and affecting the runtimes of your method.

3. For your homework, you should submit the corresponding .java file of your method,

SortingAlgorithmTester.java file and your at most 3-page report in PDF format. The report should give a detailed explanation of your experimental setup, procedure, and experimental results showing how the performance of your method changes when different sorting algorithms are used for different types of inputs (ascending / descending /random ordered). You should also write the individual steps that you took to complete the homework.

You are expected to add visualization (table, plot, graph, etc.) showing the time complexity of your method when different sorting algorithms are used with different size and type of inputs (e.g. ascending, descending, random input). For this purpose, you can use tools like MS Excel, R etc. (e.g. For MS Excel, you can follow the tutorial in the given link: https://www.youtube.com/watch?v=DAU0qqh_I-A).

4. **Hint:** for especially large arrays, you will likely get a stack overflow error and your program will not run. This is due to the limited area that JVM reserves for its own call stack. The call stack is used for storing local variables of the methods, method call arguments, etc.

To increase the size of the call stack size, try to use **"-Xss"** command line argument when you run your program (e.g. use "-Xss8m" to increase the stack size to 8 Mbytes, of course you can increase it more). Do research on how to add this option in your Java development environment (e.g. For Eclipse, follow the tutorial in the following link: https://www.youtube.com/watch?v=OU0H3d1rhfw).

WHAT TO HAND IN

A zip file for both parts containing:

- The Java sources for your program.
- The Java sources should be WELL DOCUMENTED as comments, as part of your grade will be based on the level of your comments.
- You should test your Java source files on (if) available Moodle VPL environment to ensure your code solution's correctness before submitting. VPL simply tests your program's output by checking against given sample input. You should pass that task's VPL test case successfully.
- A **maximum-3 pages** PDF report document that explains your own answers for programming task in a clearly readable PA report format (refer to **PA REPORT FORMAT** section).
- For given task, only code or report submission will not be graded. In other words, you should submit **both correct code solution and its related report for the task in order to be graded**.

A programming assignment report is a self-description of a programming assignment and your solution. The report must not be hand-written. You may use a word processor or the on-line editor of your choice and prepare as a PDF document. The report must be grammatically correct and use complete English sentences. Each report should include the following sections, in the order given:

Information (%2.5): This section includes your ID, name, section, assignment number information properly.

Problem Statement and Code Design (%15): Include a brief summary of the problem and/or your sub-tasks to be completed in this assignment. You should show your modular design rationale by creating a structure chart that indicates your top-down, stepwise refinement of the problem solution. You may create the structure chart using available graphical tools like MS PowerPoint, SmartDraw etc.

Implementation, Functionality (%20): Since you have modular source code, you should describe each sub-module (program) in this section. Each sub-module should include names and types of any input/output parameters as well as the pseudocode algorithm that used for completing its task. By this way, you give meaning to each chart boxes from the previous section.

Testing (%7.5): You should provide a tester class that is able to identify key test points of your program. This class should be able to generate additional (apart from the given sample input/output) test data for the purpose of being clear on what aspects of the solution are being tested with each set. This section should also include a description of any program *bugs* that is, tests which has incorrect results. You should write these to describe your tests, summarize your results, and argue that they cover all types of program behavior.

Final Assessments (%5): In this final section, you should briefly answer the following questions:

- What were the trouble points in completing this assignment?
- Which parts were the most challenging for you?
- What did you like about the assignment? What did you learn from it?

GRADING:

- Codes (%50, Q1:30, Q2:70)
  - Available test cases evaluation on VPL: %15, (Q1:7.5, Q2:7.5)
  - Hidden test cases evaluation: %15, (Q1:7.5, Q2:7.5)
  - Approach to the problem: %20, (Q1:10, Q2:10)
- Report (%50)
  - Information: %2.5
  - Problem Statement and Code design: %15
  - Implementation, Functionality: %20
  - Testing: %7.5
  - Final Assessments: %5

Submitting report without codes will not be evaluated!!

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. This assignment is due by 23:59 on Sunday, November 27th.

2. You should upload your homework to Moodle before the deadline. No hardcopy submission is needed. You should upload files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., zip, rar).
3. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.
4. You ARE NOT ALLOWED to modify the given method names (if there is any given methods). However, if necessary, you may define additional data members and member functions.
5. Your classes' name MUST BE as shown in the homework description.
6. The submissions that do not obey these rules will not be graded.
7. To increase the efficiency of the grading process as well as the readability of your code, you must follow the following instructions about the format and general layout of your program.
8. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-----------------------------------------------------------------------------------------------------------------------
// Title: Scheduler tester class
// Author: Name/Surname
// ID: 2100000000
// Section: 1
// Assignment: 1
// Description: This class tests the …
//-----------------------------------------------------------------------------------------------------------------------
```

9. Since your codes will be checked without your observation, you should report everything about your implementation. Add detailed comments to your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-----------------------------------------------------------------------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-----------------------------------------------------------------------------------------------------------------------
{
 // Body of the function
}
```

10. Indentation, indentation, indentation...
11. This homework will be graded by your TA, Merve Işıl Peten. Thus, you may ask her your homework related questions through HW forum on Moodle course page.