

# CMPE 242

## Spring 2020

### Programming Homework #2

This assignment is due by 23:55 on Sunday, 19 April 2020.

#### 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.

- WE RECOMMEND THAT YOU COMPLETE ONE STEP PER DAY.
- IF YOU ARE STUCK IN ONE STEP, **DO NOT** GO TO THE NEXT STEP BUT **ASK FOR HELP** FROM THE PROFESSOR OR THE TA. (It is important that you get our supervision if you are stuck).

Here are the steps of the homework:

**Step 1.** In this step, first write a program that (i) 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 Person class. Each person should have three fields: a `String` attribute, called name, a second `String` attribute called surname, a third attribute of type `long`, called id. The Person class should implement the `Comparable` interface to be able to compare it with respect to their ids. 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 Person objects. For this purpose, create 10 different Person objects, each with a different name, surname, ID. Then, call this method to sort these objects with respect to their ID's.

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

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

## PART II (50 Points)

In this part of the homework, you are given a package named **cmpe242-sort01.jar** that is available at the course Moodle site. This jar file contains a class that has implemented 5 sorting algorithms: **QuickSort**, **MergeSort**, **Insertion Sort**, **Bubble Sort**, and **Selection Sort**. These sorting algorithms have been implemented using almost similar source code as in the course slides. Note that the QuickSort algorithm selects the first item as the pivot value.

The **cmpe242-sort01.jar** package includes the following class.

---

**CLASS:** `SortingAlgorithm`

**Methods:**

- ❖ `public static void sort1(int [] ar, long studentID)`
- ❖ `public static void sort2(int [] ar, long studentID)`
- ❖ `public static void sort3(int [] ar, long studentID)`
- ❖ `public static void sort4(int [] ar, long studentID)`
- ❖ `public static void sort5(int [] ar, long studentID)`
- ❖ `public static void printArray (int [] ar )`

---

The first argument of the sort methods is the input array, and the second parameter is your TEDU student ID. You should add the letter “L” to the end of your student ID to make it a long parameter (e.g. `SortingAlgorithm.sort1(ar, 1234567801L);`).

In this assignment, you are expected to guess the sorting algorithm that the class uses for each sort method, by looking and comparing execution time of the program to sort different sizes of **ascending/descending** and **random ordered** integer arrays.

1. To use the jar package, which is including the five sorting algorithms, you should follow the following steps in Eclipse: right-click on the Project → Build Path → Configure Build Path. Under Libraries tab, click Add Jars or Add External JARs and give the Jar. For further and more detailed information, follow the tutorial in the following link: <https://www.youtube.com/watch?v=UtzAf8tyuAM>
2. To use the **SortingAlgorithm** class, 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.
3. As you test the algorithms, collect time measures to make a guess about which sorting algorithm was running.
4. For your homework, you should submit your **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**

**justifying your answer.** You should also write the individual steps that you took to complete the homework.

You are expected to add visualization (plot, graph etc.) showing the time complexity of the sorting methods with different type of input (e.g. ascending, descending, random input). Note that for some sort methods, you may not differentiate the corresponding sorting algorithms by examining the experimental results. That is, there may be more than one answer for some methods values. If that is the case, you should list all sorting algorithms that may correspond to the given case.

5. **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>).

### BONUS PART (20 Points)

In this bonus part of the homework, you are asked to design and implement your own sorting algorithm. Then, you compare it with the 5 implemented sorting algorithms from Part II (**cmpe242-sort01.jar**). These sorting algorithms may have been implemented using almost similar source code as in the course slides.

You need to implement your own sorting algorithm as an extra function into existing **SortingAlgorithmTester** class and follow the same Step 2 and 3 from Part II. Example:

```
public static void myOwnSort(int [] ar)
//-----
// Summary: Sorts the values in a given integer array
// Precondition: ar is an integer array
// Postcondition: The values in ar is sorted.
//-----
{
    // Body of the function
}
```

You are expected to add visualization (plot, graph etc.) showing the time complexity results of your own sorting method and the existing sorting methods with different type of input (e.g. ascending, descending, random input) into your same **PDF report** (Step 4, Part II). If your own sorting algorithm beats the one or more of 5 existing ones, you are to be graded due to their execution times difference respectively.

### IMPORTANT

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

1. This assignment is due by 23:55 on Sunday, April 19<sup>th</sup>.
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. 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 have to 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: Sorting tester class  
// Author: Ibrahim Ileri  
// ID: 2100000000  
// Section: 1  
// Assignment: 2  
// 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 TAs, İbrahim İleri and Hamid Ahmadlouei ([ibrahim.ileri@tedu.edu.tr](mailto:ibrahim.ileri@tedu.edu.tr), [hamid.ahmadlouei@tedu.edu.tr](mailto:hamid.ahmadlouei@tedu.edu.tr)). Thus, you may ask them your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructor Tolga Çapın and Tansel Dökeroğlu for help.

## GRADING RUBRIC

Performance Element	Weight	Master (4/4)	Advanced (3/4)	Developing (2/4)	Beginner (1/4)	Insufficient (0/4)
<b>Information</b>	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
<b>Documentation</b>	10%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
<b>Code Design</b>	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
<b>Functionality</b>	25%	All user commands are implemented with no missing functionality. Runs without any crash. Lists are in descending order.	Missing some minor user commands or minor output problems (e.g. no fixed string size). Runs without any crash. Lists are in descending order.	Attempted to implement all user commands but about half of them do not work. Major deviations from the specification.	Only few commands are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
<b>Testing: Test data creation &amp; generation</b>	10%	Provided a tester class. Able to identify key test points. Clear on what	Provided a tester class. Able to identify key test points. Clear on what aspects of the	Provided a tester class. Able to identify key test points.	Sporadic	No test case presented.

	aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	solution are being tested with each set.			
--	--	--	--	--	--