Lab Exercise 6: Sorting [60 Marks]

SE2205: Data Structures and Algorithms using Java – Fall 2022

Open Day: November 11, 2023; **Cut off time**: November 18, 2023, Sunday @11.55pm

---

Prepared by Dr. Quazi Rahman (qrahman3@uwo.ca).

A. Rationale and Background

In this lab Assignment, for question 1, we will write all the sorting methods we discussed in the class in their generic form (except for Bucket-sort) and check their execution time for an Integer type dataset. For question 2 we will create a StudentGrade type class, and then after populating a list with certain number of students and their corresponding grades, we will sort that list based on last name, or first name or the grade.

B. Evaluation and Submission Instructions

Submit your Lab-Exercise online by carrying out the following instructions:

1. Create a Project with the following name: *username_LabExercise6*
2. For this question create a package for each Question (LE6Q1, LE6Q2,)
3. Use meaningful names for each class and the associated variables by following the general naming conventions.
4. For this question, use the static header and footer methods your created before.
5. Comments: **Writing short but clear comments for Lab Exercises is <u>mandatory</u>**. If the comments are clear, full credit will be given to the written comments.
6. Once completed, select the project folder (e.g. *username_ LabExercise6*). Right-click to select 'Send to' 'Compressed zipped folder'. Make sure it has the same naming convention (e.g., *username_ LabExercise6*.zip). Upload this file to OWL as your submission.
7. You will be graded on this lab Exercise based on the comments and running code.
8. Note: if your code does not run, zero grade will be awarded even if it is a minor fix. No extension is available for this lab and any other lab as outlined in the course outline.

C. Lab Question

1. Question 1 [20 Marks]

Most of the required codes associated to this question have been made available either in the lecture-handout or in the Unit 4 Companion Document.

(a) Define a class called *YourFirstnamesTestingSortingMethods*. Inside the class, define the following generic sorting methods, where each method accepts a generic array and returns the time elapsed in sorting the array (use System.nanoTime() method as you did for first lab exercise; also, you can check Unit 2, P3 for nanoTime() method-usage in a code):

- public static <T extends Comparable <? super T>>long selectionSort (T [] a)
- public static < T extends Comparable <? super T >> long bubbleSort(T[] a)
- public static < T extends Comparable <? super T >> long insertionSort(T[] a)
- public static <T extends Comparable<? super T>> long mergeSort(T[] S) [you can tweak the recursive merge-sort code given in the Unit 4 – companion document]

- public static <T extends Comparable<? super T>> long quickSort(T[] s, int a, int b)[you can implement the in-place recursive quick-sort algorithm given in the lecture handout]
- public static long bucketSort(Integer[] a, int first, int last, int maxDigits) [You are not supposed to make it generic becaus it can only be used for specific data-sets. To understand the method-header, please check the algorithm first, which is given in the lecture handout, and it is implemented and given in the Unit 4 – companion document. You can use it as is.]

(b) Define the driver method with the following specs.

i) Call your header method (Reuse from the previous labs).
ii) Declare an **Integer** type array of size 'sz' which you can pre-set to 5.
iii) Create a backup **Integer** type array with the same size.
iv) Populate the first array with random values from 13 to 93 inclusive, by using Math.random() method.
v) Copy the content of the first array to the backup array (You can use **System.arraycopy()** method).
vi) Create a List<> for the above Integer Array using asList() method from Arrays class (List<Integer> = Arrays.asList(arrayName);) This will help you take advantage of List<>'s toString() method.
vii) Sort the first array using Collections' sort method. Print both the sorted and unsorted arrays with the help of List<>'s toString() metod. Check the time and print it on the screen (see the sample output).
viii) Now copy the backup array to the first array (you need to do this to make sure that you are sorting the same unsorted array every time)
ix) Call your selectionSort() method by passing the first array, and print the time along with the sorted and unsorted array contents.
x) Repeat step viii) before calling bubble, insertion, merge, quick and bucket sort methods, and print the unsorted array, sorted array and the elapsed time for every scenario. See the sample output 1.
xi) Since you will not get a clear idea of the elapsed time with a dataset of 5 items only, change the size to 50,000. Now repeat all the above after commenting out the sorted and unsorted array-printing calls. See the sample output 2. During your demo, your TA may ask you to change the size back to 5, uncomment those print statements, and demo that your code is sorting the array properly.
xii) Call your footer method (Reuse from previous lab).

(c) Note: Based on the processor speed, memory usage etc., your code-execution time will not be same as shown in the sample output but fundamentally, you should observe that Collections sorting method will be faster than selection, bubble, and insertion sorts. Bubble sort will take the maximum execution time. Bucket sort will generate fastest execution time. Merge sort and quick sort execution time will be better than the bubble, insertion, and selection sorts. For smaller dataset, quick sort, merge sort will perform worse than the others. Collections' sort method will be the fastest. Java's sort (in Collections, Arrays etc.) methods implements Timsort algorithm (https://en.wikipedia.org/wiki/Timsort) which is a combination of merge and insertion sort.

Sample output 1 (With printed array content):

```
=====================================================
Lab Exercise 6, Q1
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: ........!
```

```
=====================================================
Testing execution time of different sorting methods for sorting 5 random numbers:
The unsorted list: [39, 25, 14, 65, 83]
Collections' Sorting Time: 0.14 milliseconds
The sorted list using Collections' sort method: [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Selection-Sort Time: 0.01 milliseconds
The sorted list using selection-sort: [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Bubble-Sort Time: 0.00 milliseconds
The sorted list with Bubble-sort: [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Insertion-Sort Time: 0.00 milliseconds
The sorted list with Insertion-sort): [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Merge-Sort Time: 0.01 milliseconds
The sorted list with Merge-sort): [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Quick-Sort Time: 0.00 milliseconds
The sorted list with Quick sort): [14, 25, 39, 65, 83]

The unsorted list: [39, 25, 14, 65, 83]
My Bucket-Sort Time: 0.26 milliseconds
The sorted list with Bucket-sort): [14, 25, 39, 65, 83]
=====================================================
Completion of Lab Exercise 6, Q1 is successful!
Signing off - Quazi
=====================================================
```

Sample output 2 (Without printing the arrays):

```
=====================================================
Lab Exercise 6, Q1
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: ........!
=====================================================
Testing execution time of different sorting methods for sorting 50000 random numbers:
Collections' Sorting Time: 17.39 milliseconds
My Selection-Sort Time: 1565.16 milliseconds
My Bubble-Sort Time: 5997.24 milliseconds
My Insertion-Sort Time: 1870.57 milliseconds
My Merge-Sort Time: 33.28 milliseconds
My Quick-Sort Time: 114.83 milliseconds
My Bucket-Sort Time: 10.81 milliseconds
=====================================================
```

Completion of Lab Exercise 6, Q1 is successful!
Signing off - Quazi
==========

2. Question 2 [10 Marks]

(a) Create a class Called StudentGrade that implements Comparable<> interface with the following specs:

- The class header:
  public class StudentGrade implements Comparable<StudentGrade>
- Three private data fields: firstName, lastName, grade.
- A Constructor with no parameters (empty body); it is optional
- A Constructor with all the parameters (firstName, lastName, grade).
- All the Getter and setter methods for all three data fields, even if you do not need to use all.
- Implement the CompareTo() method to compare the grades.
- Override toString() method so that it can print the list as shown in the sample output for the unsorted list, and the sorted list based on the grades.
- The other two sored list based on the last names and first names, need to be printed separately; in this case, you need to create a public static void method called printArray() with appropriate parameter list in the driver class, and accomplish this task.

(b) Create a driver class YourFirstName_SortNameAndGrade. Add the public static sort() method, the printArray() method, header method, footer method, and the given driver method (see below) in your code and complete the rest of the code based on the commented instructions below:

```
public static void main(String[] args) {

//call your header method

//Three arrays with 8 first names, 8 last names and 8 randomly
generated grades between 60 and 85 inclusive have been created below
for your use
    String[] fnArray = {"Hermione", "Ron", "Harry", "Luna", "Ginny",
"Draco", "Dean", "Fred"};
    String[] lnArray = {"Granger", "Weasley", "Potter", "Lovegood",
"Weasley", "Malfoy", "Thomas", "Weasley"};
    Integer[] grd = {(int)(60 + Math.random()*26),(int)(60 +
Math.random()*26),(int)(60 + Math.random()*26),(int)(60 +
Math.random()*26),(int)(60 + Math.random()*26),(int)(60 +
Math.random()*26),(int)(60 + Math.random()*26),(int)(60 +
Math.random()*26)};
//create a Vector<> class instance 'sg' with StudentGrade tag.


//Add each StudentGrade object to the Vector class with its add() method (Slide #36, Unit 2 Part 2)
```

//print the unsorted ‘sg’ contents just by using toString() method (see the sample output)

//sort ‘sg’ using Collections’ sort method, which will sort the list based on the grade

//print the sorted content by using toString() method (see the sample output)

//Create an array of StudentGrade type with the length of the fnArray that has been created above.

//with the help of Vector’s copyInto() method (Slide #36, Unit 2 Part 2), copy ‘sg’ to StudentGrade array you just created above

//At this point tweak the stable sorting method (your created in Q1), *insertion sort* that will accept a StudentGrade array and an integer key (alternatively, you can tweak *merge sort*). Define this **public static void** method outside the driver method below. The integer key in the method header will decide whether you would like to sort the StudentGrade array according to the first name (key = 1) or the last name (key = 2). Once done, come back to this place inside the driver and call your sort-method with the argument of StudentGrade array (you created above) and key = 1 (first names).

//print the sorted array (see the sample output) with the aid of your printArray() method

// Call your sort-method again with the argument of StudentGrade array (you created above) and key = 2.

//print the sorted array (see the sample output) with the aid of your printArray() method

//call your footer method here

}

//Your sort-method() will go here.

//Add your printArray() method here.

Note: Since the grades are generated using Math.random() method, the grades will vary in every run.

Sample output:

```
----------------------
        ======================================================
        Lab Exercise 6, Q2
        Prepared By: Quazi Rahman
        Student Number: 999999999
        Goal of this Exercise: ........!
        ======================================================

        The Unsorted Array ..................
          Hermione Granger    :         84
```

```
        Ron Weasley   :    79
       Harry Potter   :    84
     Luna Lovegood    :    77
     Ginny Weasley    :    71
      Draco Malfoy    :    69
      Dean Thomas     :    75
      Fred Weasley    :    80

Sorted by Grades.....................
      Draco Malfoy    :    69
     Ginny Weasley    :    71
      Dean Thomas     :    75
     Luna Lovegood    :    77
       Ron Weasley    :    79
      Fred Weasley    :    80
Hermione Granger     :    84
       Harry Potter   :    84

Sorted by First Names ..............
      Dean Thomas     :    75
      Draco Malfoy    :    69
      Fred Weasley    :    80
     Ginny Weasley    :    71
       Harry Potter   :    84
Hermione Granger     :    84
     Luna Lovegood    :    77
       Ron Weasley    :    79

Sorted by Last Names...............
Hermione Granger     :    84
     Luna Lovegood    :    77
      Draco Malfoy    :    69
       Harry Potter   :    84
      Dean Thomas     :    75
      Fred Weasley    :    80
    Ginny Weasley     :    71
       Ron Weasley    :    79

=====================================================
Completion of Lab Exercise 6, Q2 is successful!
Signing off - Quazi
=====================================================
```

Here are some sample questions for your demo-prep. The TAs can ask any kind of question if those are related to this lab.

1. *What is 'natural order'.*
2. *What is 'order by comparator'.*
3. *What do you need to do if you would like to sort the instantiated objects of a class called 'MyClass' using natural order?*
4. *What is stability of sorting algorithm.*
5. *How the stability of a sorting algorithm is ensured.*
6. *Can any sorting algorithm be made stable?*
7. *What are the inherently stable sorting algorithms?*
8. *Can Radix sort be used to sort the instantiated objects of a class called 'MyClass'. What is the reasoning of your response?*
9. *If you sort an array with same dataset, with the same algorithm, in the same machine twice, will the execution time for both runs will be the same? What is the basis of your response?*
10. *How have you picked the pivot in the quick sort algorithm? What is the justification of this choice?*
11. *If you call the quick-sort method to sort an already sorted array based on the implementation you presented what would happen in terms of time and space complexity. What is the basis of your response?*
12. *Modify any algorithm to do the sorting in descending order.*
13. *Explain how you generate a random number between min and max inclusive using Math.random() method.*
14. *If you have a smaller dataset of 50 elements, which algorithm would you choose to sort that set, and why?*
15. *If you create a class MyClass that implements Comparable<T> interface, would you prefer choosing primitive datatype (when needed) over Wrapper class object reference. What is the basis of your response?*
16. *If you would like to do Bucket-search Strings that use alphabetical characters, how many buckets would you need to accomplish this? Why?*
17. *What would be your preference between Merge and quick sort? Why?*
18. *If you are sorting integer data set, what would be your sorting algorithm choice? Why?*
19. *Point out in the code, how the best-case time complexity of Insertion sort is O(n). Also, do the same for Bubble sort.*