CSE17 **Lehigh University** **Fall 2020**

Programming and Data Structures
Programming Project 4: Sorting Algorithms

Project Objectives

At the end of this project, students should be able to:

1. Implement the generic version of the sorting algorithms covered in class
2. Compare the performance of the sorting algorithms for different data sets

Project description

1. Create the class **`Heap`** as defined in the lecture on sorting algorithms. Add a third constructor to class **`Heap`**, that takes a parameter **`l`** of type **`ArrayList<E>`** and adds the elements from **`l`** to **`list`**.
2. Create a class **`Sort`** that contains the definition of static methods for the sorting algorithms covered in class (selection sort, insertion sort, bubble sort, merge sort, quick sort, heap sort). Note that all the methods should be generic and accept an array list of type E, instead of an array. Modify all the methods accordingly. For **`mergeSort`**, define a method **`subList`** with the header below and use it to split the list into two halves **`firstHalf`** an **`secondHalf`**.
   ```
   public static <E> ArrayList<E> subList(ArrayList<E> list, int start, int end)
   ```
   The method returns an array list that contains a deep copy of **`list`** elements from index **`start`** to index **`end-1`**.

3. All sorting methods should calculate the number of iterations and store it in a static variable inside the class Sort as we did in ALA 10.

4. Create a class **`Testing`** for the test program. Create three array lists of type **`Integer`** and size 10,000 for the data sets with the specification below:

a. **randomList**: array list filled with random integers in the range 0 to 9999. Use the method **java.util.Collections.shuffle()** to shuffle **randomList** after each sorting algorithm.

b. **sortedList**: contains the same data as **randomList** in ascending order. To obtain **sortedList**, clone **randomList** and sort the cloned list using the method **java.util.Collections.sort()**.

c. **reversedList()**: contains the same data as **sortedList** in descending order. To obtain **reversedList**, clone **sortedList** and reverse the cloned list using the method **java.util.Collections.reverse()**. Do not forget to recreate **reversedList** after each sorting algorithm.

5. Call each sorting method on the three data sets generated in step 4 and record the number of iterations for each sorting algorithm on each data set. Then display the results in a tabular form similar to the following sample output.

```
Comparing Sorting Algorithms for data sets with 10000 integers

Sorting Algorithm       Random List    Sorted List         Reversed List
Selection Sort          50014998       50014998            50014998
Insertion Sort          25318731       9999                50004490
Bubble Sort             49990972       10000               50004999
Merge Sort              153615         153615              153615
Quick Sort              170893         50014999            48773016
Heap Sort               159605         247334              146693
```

6. Include a multi-line comment inside the class **Testing**. The comment must include a brief discussion of the results. Discuss first the performance of each sorting algorithm on the different data sets. Then compare the performance of the sorting algorithms to each other.

7. Submit the following files on coursesite: **Heap.java**, **Sort.java**, and **Testing.java.**