Comp 482 Project #1 (30 pts) Fall 2017

Due: Thursday Sept 26

GENERAL DIRECTIONS: This is an individual project. In this project you will create a Java class to analyze and compare the mergesort and quicksort algorithms. The sort required is always from smallest value to largest value. Your source code file must be named Sorts.java . Neatness counts and so does indented code that is easy to read with helpful variable names. Your Java class methods must match the specifications in the class outline below. In particular, all specified methods must have the same signatures as specified on page 4. Otherwise my test cases will not work. You may add any extra private methods and fields as needed.

1. Create a static Java method, called **merge**, that merges two sorted lists of integers into a single sorted list. Use the merge algorithm from class or our text. This should be a linear algorithm in the sum of the lengths of the arrays.

2. Create a static Java method, called **mergeSort,** that implements a mergesort algorithm that sorts an array of n integers. Use the mergesort algorithm from class or our text.

3. Create a static **partition** method that implements the in-place partition algorithm for quicksort from class or the text. This should be a linear algorithm in the length of the subarray array being partitioned.

4. Create a static method **quickSort** that implements the in-place quick sort from class or the text.

5. Create a java method, called **isSorted**, that tests if an array of integers is sorted in increasing order.

6. Test your merge, mergeSort, partition, quickSort methods thoroughly to make sure that they are correct; ie they can sort arrays of random values of various lengths. Test your isSorted method too, Make sure to print out the results of the sorts for smaller arrays to visually check that the arrays are sorted.

   DIRECTIONS FOR CODING THE EXPERIMENTS. Create a new java file called SortTester.java that contains the code for the experiments. SortTester should use the methods in Sorts.java.

7. Experiment #1: Design an experiment to determine which of mergesort or quicksort is the faster sorting algorithm on a on integer arrays of random numbers? Does your answer depend on the size of the array?

   Use <u>random</u> arrays of size n, for n = 10, 100, 1000, 10000, 100000, 1000000, 2000000. Use a random number generator to fill your array with integers between 1 and 1000000. For each n,

run 20 trials to compare speeds of mergeSort and quickSort. For example, if n = 100, runt mergeSort and quickSort on 20 different arrays of 100 random numbers. For each n, report the number of wins for mergeSort and the number of wins for quickSort. Your program should print out the information to create a table that reports the number of wins for mergeSort and for quicksort as a function of n. In your report, the table should be formatted as follows:

n	nTrials	# mergeSort Wins	# quicksortWins
10	20		
100	20		
1000	20		
10000	20		
100000	20		
1000000	20		
2000000	20		

8. Experiment #2 Show that runtimes of mergeSort and quicksort are $O(n*\log_2(n))$ by timing the runs in Experiment #1 by using the java function System.nanoTime().

   Your program should printout the information needed to create a table with that reports the mean runtime (nanosecs) of mergeSort and quickSort (over 20 trials) for each n. How do the results of the table confirm that mergeSort and quickSort are O(n*log(n)) ? Refer to the definition of big Oh. In your report, the table should be formatted as follows:

n	mergeSort: mean runtime (nanosecs)	mergeSort: mean runtime/ $(n*\log_2(n))$	quickSort: mean runtime (nanosecs)	quickSort: mean runtime / $(n*\log_2(n))$
10				
100				
1000				
10000				
100000				
1000000				
2000000				

Turn in:

1. Report with
    - Cover Page;
    - Sorts.java and SortTester.java source code; (Two files)
    - Computer printout (from Sort.java main program) that shows that quickSort and mergeSort sort the following array.
      { 34, 67, 23, 19, 122, 300, 2, 5, 17, 18, 5, 4 3 19, -40, 23 }

    - Computer printout from SortTester.java that prints all of the information needed for your report in a table format.
    - A 2-3 page typewritten discussion that describes the experiments, presents the results in table form and discusses the meaning of the results in the tables.
    - Due: In class on Sept 26

2. Electronic Copy: **Upload** a single source file called Sorts.java to Canvas. The file should not be a package nor a zip file. Your source file MUST contain your name, date and Project# as a comment at the top of the file. You will lose 5 points from your project if your name, date and project # are not included in your file. (Due Thursday Sept 26 at 12 noon)

```
//Note: you may add any private data fields or methods needed

public class Sorts
{

//returns true only if a is sorted from smallest to largest values

  public static boolean isSorted( int[] a)


/* --------------------MergeSort --------------------*/

  //merges sorted slices a[i.. j] and a[j + 1 ...  k] for 0<=  i <=j < k < a.length
 public static void merge ( int[] a  int i, int j , int k)  {  }

   //sorts  a[ i .. k]  for 0<=i <= k < a.length
  public  static  void mergeSort(int[] a,  int i ,  int k)  {  }

 //Sorts the array a using mergesort
  public static  void  mergeSort(int[] a ) {  }

 /* ----------   QuickSort ---------------------------------------------- */

//Implements in-place partition from text. Partitions subarray s[a..b] into s[a..l-1] and s[l+1..b]
// so that all elements of  s[a..l-1] are less than each element in s[l+1..b]

public static int partition ( int[] s,  int a , int b  )  {  }

//quick sorts subarray a[i..j]
public static void quickSort ( int[] a, int i, int j) {  }

//quick sorts array a
public static void quickSort( int[] a) { }

}
```