Comp 496 ALG Project #1 (30 pts) Spring 2017

Due: Tuesday Feb 28

**GENERAL DIRECTIONS:** This is an individual project. In this project you will create a Java class to create, analyze and compare two sort algorithms. The sort requires is always from smallest value to largest value. Your source code 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 (Page 3) . In particular, all specified methods must have the same signatures as specified. Otherwise my test cases will not work. You may add any extra private methods needed.

1. Create a java function, 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. It should return as a long int the number of comparisons of array elements made.
2. Create a java function, called **mergesort,** that implements a mergesort algorithm that sorts an array of n integers. Use the mergesort algorithm from class or our text. Your merge sort algorithm should return the exact number of comparisons made between elements of the input array. It should include the count of the number of comparisons made in the merge function. This type of the return value should be a long int.
3. Create a java function, called **isSorted**, that tests if an array is sorted.
4. Experiment #1: Test your mergesort code on random arrays of size n, for n = 10, 100, 1000, 10000, 100000, 1000000, 5000000. Use a random number generator to fill your array with integers between 1 and 1000000. Test each n five times. For example, if n = 100, test five different arrays of 100 random numbers. Use isSorted to test that your algorithm actually does sort your array. Track the number of comparisons made. Time mergesort using the java function System.nanoTime().

Print out a table with that shows the average number of comparisons and the average runtime for each n. All arithmetic divisions in the table should use double number types. See required format of table on next page.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | C(n) = Mean number comparisons | C(n) / (n\*log2(n) ) | T(n) = Mean runtime (nanosecs) | T(n)/ (n\*log2(n) ) |
| 10 |  |  |  |  |
| 100 |  |  |  |  |
| 1000 |  |  |  |  |
| 10000 |  |  |  |  |
| 100000 |  |  |  |  |
| 1000000 |  |  |  |  |
| 5000000 |  |  |  |  |

1. Experiment #2. Repeat experiment #1 using an insertion sort algorithm. You will need to implement an insertion sort to do this. Also change the n\*log2(n) in the table to n2. Why? **ADDED: You do not have to test Insertion Sort at n = 1000000 and n = 5000000 if it takes TOO LONG. Try 200000 and 300000.**

**Turn in:**

1. Report with
   * Cover Page;
   * Sorts.java source code;
   * Java driver program that shows all of code that created the data for Experiment #1 and Experiment #2; THIS DRIVER CODE IS A VERY IMPORTANT PART OF YOUR PROJECT.
   * A 3 page typewritten discussion of that describes the experiment, presents the tabled results and discusses the meaning of the results in the table.
   * Due: In class on Feb 28.
2. Electronic Copy: **Upload** a single source file called Sorts.java to Moodle. The file should not be a package nor a zip file. Your source file should contain your name, date and Project# as a comment. (Due Tuesday Feb 28 at 8:00 am)

public class Sorts  
{  
  
/\*--------------Insertion Sort -----------------------\*/  
 public static long insertionsort( int[] a)  
 {  
   
 }  
  
  
  
/\* --------------------Merge Sort --------------------\*/

//merges sorted slices a[i.. j] and a[j + 1 … k] for 0<= i <=j < k < a.length

public static long merge ( int[] a int i, int j , int k)  
 {

}

//sorts a[ i .. k] for 0<=i <= k < a.length  
 private static long mergesort(int[] a, int i , int k)  
 {  
   
 }

//Sorts the array using mergesort

public static long mergesort(int[] a )  
 {  
   
 }  
  
  
 public static boolean isSorted( int[] a)  
 {  
 }

}