Assignment # 2
Due Date: October 10, 2024

Introduction:

Sorting routines are among the most widely used and studied algorithms. Every student should know how to implement several different kinds of sorts, and should have idea about how they perform, both theoretically and practically. This programming project is designed to give the student practice in implementing and observing the behavior of two sorts: Insertion Sort, Merge Sort.

Resources:

The algorithms for Insertion Sort and Merge Sort are given with the pseudo code.

Programs must be written in Java.

You will find links to 12 data files on Blackboard for this assignment. These files all contain shuffled lists of integers, with one integer listed per line. The files are:

Filename	# items	lowest	highest	Description
Num8.txt	2^3	1	8	no omissions, no duplicates
Num16.txt	2^4	1	16	no omissions, no duplicates
Num32.txt	2^5	1	32	no omissions, no duplicates
Num64.txt	2^6	1	64	no omissions, no duplicates
Num128.txt	2^7	1	128	omissions/duplicates possible
Num256.txt	2^8	1	256	omissions/duplicates possible
Num512.txt	2^9	1	512	omissions/duplicates possible
Num1024.txt	2^{10}	1	1024	omissions/duplicates possible
Num2048.txt	2^{11}	1	2048	omissions/duplicates possible
Num4096.txt	2^{12}	1	4096	omissions/duplicates possible
Num8192.txt	2^{13}	1	8192	omissions/duplicates possible
Num16284.txt	2^{14}	1	16384	omissions/duplicates possible

Description:

You will write Java code that implements the algorithms for the sorting routines mentioned above. As part of your code, you will include counters that iterate whenever a specific line of the algorithm is executed.

Some lines in an algorithm may have a higher *cost* than other lines. For example, the function call in line 5 in the Merge Sort algorithm is executed only 7 times for an array with 8 elements, but the body of the Merge function which is being called has many lines, some of which are executed more than once. So the cost of line 5 in the Merge sort algorithm is higher than the other 4 lines. We can use the cost of the highest-cost line as an *indicator* of the cost of the algorithm as a whole.

Insertion Sort:

Here is the pseudocode for Insertion Sort, modified to include a counter:

```
count ← 0
Insertion_Sort (A)
1     for j ← 2 to length(A) do
2           key ← A[j]
3           // Insert A[j] into the sorted sequence A[1..j - 1]
4           i ← j – 1
5           while i > 0 and A[i] > key do
5.5               count ← count + 1
6                 A[i + 1] ← A[i]
7                 i ← i – 1
8           A[i + 1] ← key
```

Your code for Insertion Sort should have a line in it that is equivalent to line 5.5 in the Insertion_Sort pseudocode above. The global variable *count* will keep a running total of the number of times this line is executed. When you exit from the call to the Insertion Sort function, you should print out the values of *n* (the length of the array) and *count* as an indicator of the cost of the function.

Merge Sort:

Here is the pseudocode for Merge Sort, modified to include a counter:

```
count ← 0
Merge_Sort(A, p, r)
1     if p < r
2           then  q ← ⌊(p + r)/2⌋
3                 Merge-Sort (A, p, q)
4                 Merge-Sort (A, q+1, r)
5                 Merge (A, p, q, r)
```

And here is the modified algorithm for the Merge function used by Merge Sort:

```
Merge (A, p, q, r)
1     n1 ← (q – p) + 1
2     n2 ← (r – q)
3     create arrays L[1..n1+1] and R[1..n2+1]
4     for i ← 1 to n1 do
5           L[i] ← A[(p + i) -1]
6     for j ← 1 to n2 do
7           R[j] ← A[q + j]
8     L[n1 + 1] ← ∞
9     R[n2 + 1] ← ∞
10    i ← 1
11    j ← 1
12    for k ← p to r do
12.5        count ← count + 1
13          if L[I] <= R[j]
14                then  A[k] ← L[i]
15                      i ←i + 1
16                else A[k] ← R[j]
17                      j ← j + 1
```

Your code for Merge Sort should have a line of code in it that is equivalent to line 12.5 in the Merge pseudocode above. The global variable *count* will keep a running total of the

number of times this line is executed. When you exit from the call to the Insertion Sort function, you should print out the values of *n* (the length of the array) and *count* as an indicator of the cost of the function.

Program Execution:

Your program should read in data from Num8.txt, run Insertion Sort on it, and store its results; read in data from Num16.txt, run Insertion Sort on it, and store its results, etc., up through file Num16284.

Next it should repeat the process, using Merge Sort as the sorting routines.

When your program terminates, you should have 24 sets of results. Each set of results should contain:

(1) the value of *count* immediately prior the termination of the algorithm,
(2) the array after having been sorted by your sort routine

Deliverables:

By the due date, submit your lab report and source code as well as an instruction how to compile and run your code to Blackboard.

If you are submitting multiple files, compress into and send a single zip file.

About Lab report:

This lab report should consist of:

1) Source code: include source code files.

2) Test case output: include a print out of the results of running your program with your test cases. Include the *whole* sorted file for files Num8.txt, Num16.txt, Num32.txt, and Num64.txt. For each of the other files include a printout of the sorted file consisting of array items with index values of 51 through 100 *only*. sectionthe.

3) Test case summary: The test case summary should be presented in the form of a table. Each entry in the table should be the value of *count* for one of the sorts after sorting one of the data sets. The table should have the following form:

Test Case Summary

	Insertion sort	*Merge sort*
Num8.txt		
Num16.txt		
Num32.txt		
Num64.txt		
Num128.txt		
Num256.txt		
Num512.txt		
Num1024.txt		
Num2048.txt		
Num4096.txt		
Num8192.txt		
Num16284.txt		

4) Analysis: Analyze and discuss the results of your experiment. You may want to plot your results using Excel or a graphing program to help you visualize the results better. At the very least answer the following questions:

a) Discuss what your results mean regarding the theoretical run-time of the different algorithms.
b) Do the sorts really take $O(n^2)$ and O(n lg n) steps to run?
c) Explain how you got your answer to this question.

Academic Honesty:
Please do your own work on this assignment. No collaboration in coding or writing the laboratory report is allowed. Downloading code from the web is also NOT permitted for this assignment.