**HW8 CSCD300**

**Rules**

**If you have skipped one or more lectures relevant to this homework assignment without an acceptable excuse, the instructor will refuse to answer ANY questions about this homework. An acceptable excuse is usually documented or related to an emergency. Your instructor has checked the class attendance everyday. Please refer to the syllabus for more information regarding this rule.**

**To turn in: please wrap up all your java source files (.java files) into a zip file, then** turn in the single zip file on the **EWU Canvas** by going to CSCD300-01 course page on Canvas, then clicking Assignments🡪HW8->submit. Please name your zip file with your last name, followed by the first initial of your first name, followed by hw8. For example, if you are John Smith, name you file as smithjhw8.zip

**Specifics**

1. You have to implement your own **LinkedList** class in this homework (either Doubly or Singly is acceptable). And, you have to implement your own **Queue** class. Your LinkedList class and Queue class must be separate.
2. Please implement the MergeSort() method in your **LinkedList** class, whichisdesignedfor sorting a LinkedList with the merge sort algorithm, as we learned in the classroom. The logic idea is shown below, which is entirely **different from** the logic idea of the MergeSort on an array. If you fail to follow the logic idea as described below, you could get a zero in this project.

1, Create a Queue object **q**, then enqueue each data object in the linked list to be sorted, as a **single-node** LinkedList. The following piece of pseudo code illustrates this step.

{

Queue q = new Queue()

For each **data object** **D in the linked list to be sorted**

first create a new linked list object, newList = new LinkedList(),

perform newList.addFirst(**D**),

q.enqueue(newList)

End For

}

2, while there is more than one items in the Queue **q** {

a. dequeue and assign to a **LinkedList reference** sublist1, (already **sorted**)

b. dequeue again and assign to another **LinkedList reference** sublist2, (already **sorted**)

c. walk through **sorted** sublist1 and **sorted** sublist2 and **merge** them into a larger sorted list **tempList**, with tempList = merge(sublist1, sublist2 )

d. q.enqueue( **tempList )**

}

3, dequeue your sorted linked list. Here think about how to make this list sorted, because the dequeued list and this list are different object. **Hint**: we learned this technique when we used addOrdered() method to sort an linked list.

1. The logic idea of the merge() method in the step 2.c above is also provided in the below.

Algorithm merge(A, B)

Input sorted SubList A and sorted SubList B

Output sorted sequence C as Union A and B

S <--create an empty list with size equal to size(A) + size(B)

while (A is not Empty **and** B is not Empty ) {

if A's first data Element **fa** < B's first data Element **fb**

remove **fa** from **A //**A becomes shorter

**S**.addLast(**fa**) // supposed to have O(1) time complexity

else

remove **fb** from **B //**B becomes shorter

**S**.addLast(**fb**) // supposed to have O(1) time complexity

}

while A is not Empty {

S.addLast( first data element in A))// supposed to have O(1) time complexity

A.removeFirst();

}

while B is not Empty() {

S.addLast(first data element in B) // supposed to have O(1) time complexity

B.removeFirst();

}

return S;

1. Please implement the InsertionSort() method in your **LinkedList class** that isdesigned for sorting a LinkedList object with InsertionSort Algorithm, as we learned in the classroom.
2. Please write a method **boolean** **isSorted()** in your LinkedList class to verify whether this LinkedList is correctly sorted in an ascending order or not.

***boolean isSorted()***

Note: **isSorted(**) returns true, if this list has been sorted in ascending order. Otherwise it returns false.

Note: your isSorted() **has to** run in the time complexity of **O(n**), where **n** is the size of the input LinkedList. In other words, you scan this input list **once**, you should know whether it is sorted or not.

1. In your main() method in a source file named **Tester.java**, please create an empty LinkedList **A**, then please use the Java Random class to generate a sequence of 20,000 random integers, which range between 0 and 3,000,000. And you have to save all the random numbers into the LinkedList **A**, with each list node of **A** holding one integer.

NOTE PLEASE READ: **For those who have an old and slow machine, sorting two 20,000 integers maybe too time-consuming. You can generate 2000 integers and sort them, instead of 20,000.**

1. In your main() method of the **Tester.java** file, please create another separate LinkedList object **A2**, which contains the same set of random integers as **A** does.
2. Please invoke your MergeSort() method on the input list **A** in the java file **Tester.java**. Then call A.isSorted() to verify whether A is correctly sorted. You should expect that A.isSorted() returns true.
3. Please invoke your InsertionSort() method on the input list **A2** in the java file **Tester.java**. Then call A2.isSorted() to verify whether A2 is correctly sorted. You should expect that A2.isSorted() returns true.
4. Your **main() method** of your **Tester** class should perform the operations **described** in the below. The method calls in your main() may differs from mine **syntactically**, with a different method interface. But, your main() should generally performs the operations that are listed below.

import java.util.Random;

public class Tester {

public static void main(String[] args) {

LinkedList A = new LinkedList();

LinkedList A2 = new LinkedList();

int ramdomListSize = 20000;

for(int i = 0; i < ramdomListSize; i++) {

int randomInt = (int)(Math.random() \* 3000000);

A.addLast(randomInt);

A2.addLast(randomInt);

}

//measure the time cost of merge sort

double then = System.currentTimeMillis();

A.MergeSort();

double now = System.currentTimeMillis();

System.out.println("Time cost in milliseconds for mergesort " + (now - then));

System.out.println(A.isSorted()); //verify that your merge sort implementation works.

System.out.println("Size of list A is: " + A.getSize());

//measure the time cost of insertion sort

then = System.currentTimeMillis();

A2.InsertionSort();

now = System.currentTimeMillis();

System.out.println("Time cost in milliseconds for insertionsort " + (now - then));

System.out.println(A2.isSorted() ); //verify that your insertion sort works.

System.out.println("Size of list A2 is: " + A2.getSize());

}//end of main

}//end of tester class

1. In the below is the correct output of your program. Note that the time cost may varies on different machines, however, generally mergesort should be much quicker than insertionsort.

Time cost in milliseconds for mergesort 37.0

true

Size of list A is: 20000

Time cost in milliseconds for insertionsort 660.0

true

Size of list A2 is: 20000

1. Please organize your source code so that I can compile **all** your source files in **one** folder using command, **javac \*.java**, and run your program using command on command line, **java Tester.**
2. In this project, you have to implement three **separate** java classes of your own: **Queue**, **LinkedList**, **Tester**.
3. **You can have your own design for any details that have NOT been specified in this document.**