Project 1: CS/CE/SE 3345: Data Structures and Algorithm Analysis

Purpose: Sorting Techniques Due Date: June 28 at 11:30pm

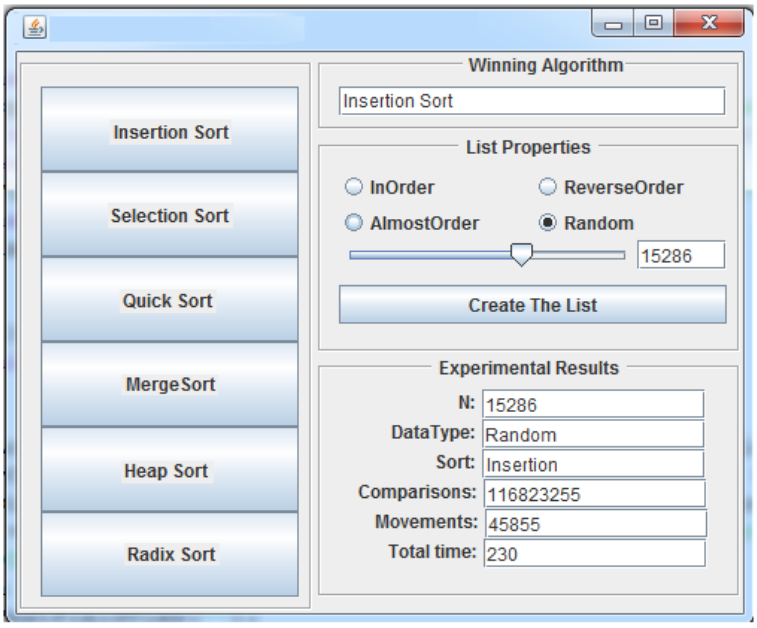
The purpose of this assignment is to implement various data structures and algorithms described in class.

Overview

One of the most important ADTs is the Dictionary and one of the most studied problems is sorting. In this assignment, you will write multiple implementations of sorting algorithms.

Are there techniques you can create or tweaks you can make to introduce a Winning Algorithm for this assignment outside of the techniques listed in the GUI?

Write a GUI or UI based to perform analysis on various sorting algorithms from the Sorting Algorithm Slides. Submit a report discussing the analysis at each iteration. Clearly define your approach, challenge and assessment.



Below is the minimum information your program's output need to provide.

Experimental Results	InOrder	ReverseOrder	AlmostOrder	Random	Array Size	Comparisons	Movements	Total Time
Insetion Sort								
Selection Sort								
Quick Sort								
Merge Sort								
Heap Radix Sort								

LISTING 23.1 InsertionSort.java

```
1  public class InsertionSort {
2    /** The method for sorting the numbers */
3    public static void insertionSort(int[] list) {
4      for (int i = 1; i < list.length; i++) {
5        /** Insert list[i] into a sorted sublist list[0..i-1] so that
6            list[0..i] is sorted. */
7        int currentElement = list[i];
8        int k;
9        for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
10         list[k + 1] = list[k];
11       }
12
13       // Insert the current element into list[k + 1]
14       list[k + 1] = currentElement;
15     }
16   }
17 }
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

LISTING 7.8 SelectionSort.java

```
1  public class SelectionSort {
2    /** The method for sorting the numbers */
3    public static void selectionSort(double[] list) {
4      for (int i = 0; i < list.length - 1; i++) {
5        // Find the minimum in the list[i..list.length-1]
6        double currentMin = list[i];
7        int currentMinIndex = i;
8
9        for (int j = i + 1; j < list.length; j++) {
10         if (currentMin > list[j]) {
11           currentMin = list[j];
12           currentMinIndex = j;
13         }
14       }
15
16       // Swap list[i] with list[currentMinIndex] if necessary
17       if (currentMinIndex != i) {
18         list[currentMinIndex] = list[i];
19         list[i] = currentMin;
20       }
21     }
22   }
23 }
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

LISTING 23.8 QuickSort.java

```
1  public class QuickSort {
2    public static void quickSort(int[] list) {
3      quickSort(list, 0, list.length - 1);
4    }
5
6    public static void quickSort(int[] list, int first, int last) {
7      if (last > first) {
8        int pivotIndex = partition(list, first, last);
9        quickSort(list, first, pivotIndex - 1);
10       quickSort(list, pivotIndex + 1, last);
11     }
12   }
13
14   /** Partition the array list[first..last] */
15   public static int partition(int[] list, int first, int last) {
16     int pivot = list[first]; // Choose the first element as the pivot
17     int low = first + 1; // Index for forward search
18     int high = last; // Index for backward search
19
20     while (high > low) {
21       // Search forward from left
22       while (low <= high && list[low] <= pivot)
23         low++;
24
25       // Search backward from right
26       while (low <= high && list[high] > pivot)
27         high--;
28
29       // Swap two elements in the list
30       if (high > low) {
31         int temp = list[high];
32         list[high] = list[low];
33         list[low] = temp;
34       }
35     }
36
37     while (high > first && list[high] >= pivot)
38       high--;
39
40     // Swap pivot with list[high]
41     if (pivot > list[high]) {
42       list[first] = list[high];
43       list[high] = pivot;
44       return high;
45     }
46     else {
47       return first;
48     }
49   }
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

LISTING 23.9 Heap.java

```
1  public class Heap<E extends Comparable<E>> {
2    private java.util.ArrayList<E> list = new java.util.ArrayList<>();
3
4    /** Create a default heap */
5    public Heap() {
6    }
7
8    /** Create a heap from an array of objects */
9    public Heap(E[] objects) {
10     for (int i = 0; i < objects.length; i++)
11       add(objects[i]);
12   }
13
14   /** Add a new object into the heap */
15   public void add(E newObject) {
16     list.add(newObject); // Append to the heap
17     int currentIndex = list.size() - 1; // The index of the last node
18
19     while (currentIndex > 0) {
20       int parentIndex = (currentIndex - 1) / 2;
21       // Swap if the current object is greater than its parent
22       if (list.get(currentIndex).compareTo(
23           list.get(parentIndex)) > 0) {
24         E temp = list.get(currentIndex);
25         list.set(currentIndex, list.get(parentIndex));
26         list.set(parentIndex, temp);
27       }
28       else
29         break; // The tree is a heap now
30
31       currentIndex = parentIndex;
32     }
33   }
34
35   /** Remove the root from the heap */
36   public E remove() {
37     if (list.size() == 0) return null;
38
39     E removedObject = list.get(0);
40     list.set(0, list.get(list.size() - 1));
41     list.remove(list.size() - 1);
42
43     int currentIndex = 0;
44     while (currentIndex < list.size()) {
45       int leftChildIndex = 2 * currentIndex + 1;
46       int rightChildIndex = 2 * currentIndex + 2;
```

```
47
48        // Find the maximum between two children
49        if (leftChildIndex >= list.size()) break; // The tree is a heap
50        int maxIndex = leftChildIndex;
51        if (rightChildIndex < list.size()) {
52          if (list.get(maxIndex).compareTo(
53              list.get(rightChildIndex)) < 0) {
54            maxIndex = rightChildIndex;
55          }
56        }
57
58        // Swap if the current node is less than the maximum
59        if (list.get(currentIndex).compareTo(
60            list.get(maxIndex)) < 0) {
61          E temp = list.get(maxIndex);
62          list.set(maxIndex, list.get(currentIndex));
63          list.set(currentIndex, temp);
64          currentIndex = maxIndex;
65        }
66        else
67          break; // The tree is a heap
68      }
69
70      return removedObject;
71    }
73    /** Get the number of nodes in the tree */
74    public int getSize() {
75      return list.size();
76    }
77  }
```

LISTING 23.10 HeapSort.java

```
1  public class HeapSort {
2    /** Heap sort method */
3    public static <E extends Comparable<E>> void heapSort(E[] list) {
4      // Create a Heap of integers
5      Heap<E> heap = new Heap<>();
6
7      // Add elements to the heap
8      for (int i = 0; i < list.length; i++)
9        heap.add(list[i]);
10
11     // Remove elements from the heap
12     for (int i = list.length - 1; i >= 0; i--)
13       list[i] = heap.remove();
14   }
15
16   /** A test method */
17   public static void main(String[] args) {
18     Integer[] list = {-44, -5, -3, 3, 3, 1, -4, 0, 1, 2, 4, 5, 53};
19     heapSort(list);
20     for (int i = 0; i < list.length; i++)
21       System.out.print(list[i] + " ");
22   }
23 }
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

LISTING 23.6 MergeSort.java

```
1  public class MergeSort {
2    /** The method for sorting the numbers */
3    public static void mergeSort(int[] list) {
4      if (list.length > 1) {
5        // Merge sort the first half
6        int[] firstHalf = new int[list.length / 2];
7        System.arraycopy(list, 0, firstHalf, 0, list.length / 2);
8        mergeSort(firstHalf);
9
10       // Merge sort the second half
11       int secondHalfLength = list.length - list.length / 2;
12       int[] secondHalf = new int[secondHalfLength];
13       System.arraycopy(list, list.length / 2,
14         secondHalf, 0, secondHalfLength);
15       mergeSort(secondHalf);
16
17       // Merge firstHalf with secondHalf into list
18       merge(firstHalf, secondHalf, list);
19     }
20   }
21
22   /** Merge two sorted lists */
23   public static void merge(int[] list1, int[] list2, int[] temp) {
24     int current1 = 0; // Current index in list1
25     int current2 = 0; // Current index in list2
26     int current3 = 0; // Current index in temp
27
28     while (current1 < list1.length && current2 < list2.length) {
29       if (list1[current1] < list2[current2])
30         temp[current3++] = list1[current1++];
31       else
32         temp[current3++] = list2[current2++];
33     }
34
35     while (current1 < list1.length)
36       temp[current3++] = list1[current1++];
37
38     while (current2 < list2.length)
39       temp[current3++] = list2[current2++];
40   }
41
42   /** A test method */
43   public static void main(String[] args) {
44     int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
45     mergeSort(list);
46     for (int i = 0; i < list.length; i++)
47       System.out.print(list[i] + " ");
48   }
49 }
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

```
void bucketSort(E[] list) {
  E[] bucket = (E[])new java.util.ArrayList[t+1];

  // Distribute the elements from list to buckets
  for (int i = 0; i < list.length; i++) {
    int key = list[i].getKey(); // Assume element has the getKey() method

    if (bucket[key] == null)
      bucket[key] = new java.util.ArrayList<>();

    bucket[key].add(list[i]);
  }
  // Now move the elements from the buckets back to list
  int k = 0; // k is an index for list
  for (int i = 0; i < bucket.length; i++) {
    if (bucket[i] != null) {
      for (int j = 0; j < bucket[i].size(); j++)
        list[k++] = bucket[i].get(j);
    }
  }
}
```

Ref: Intro to Java Programming, Comprehensive Version / Edition 10 by Y. Daniel Liang

```
import java.io.*;
import java.util.*;
class Radix {
        static int getMax(int arr[], int n){
        int mx = arr[0];
        for (int i = 1; i < n; i++)
            if (arr[i] > mx)
                mx = arr[i];
        return mx;
    }
   static void countSort(int arr[], int n, int exp) {
        int output[] = new int[n];
        int i;
        int count[] = new int[10];
        Arrays.fill(count,0);
        for (i = 0; i < n; i++)
            count[ (arr[i]/exp)%10 ]++;
        // Change count[i] so that count[i] now contains
        // actual position of this digit in output[]
        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];
        // Build the output array
        for (i = n - 1; i >= 0; i--){
            output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];
            count[ (arr[i]/exp)%10 ]--;
       }
        for (i = 0; i < n; i++)
            arr[i] = output[i];
    }
   static void radixsort(int arr[], int n)
      { // Find the maximum number to know number of digits
        int m = getMax(arr, n);
        for (int exp = 1; m/exp > 0; exp *= 10)
            countSort(arr, n, exp);
    }
   static void print(int arr[], int n)  {
       for (int i=0; i<n; i++)
            System.out.print(arr[i]+" ");
    }
    public static void main (String[] args)
    {    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
        int n = arr.length;
        radixsort(arr, n);
        print(arr, n);
    }
}
```