Project 2: A Few other Sorting Algorithms

The purpose of this project is to understand the syntax of pseudo code, and further sharpen your Java programming skills.

1 Something to work with...

The following is a Java program to sort, in Insertion sort, a list of items of the Contact type.

1.1 A Java implementation of the insertion sort algorithm

Below is an interface, Sorter, and an implementation, InsertionSort, included in the Java Package that comes with the text book by Cormen et al [1], with most of the comments taken off.

//This has to go into a separate file
public interface Sorter{
   public void sort(Comparable[] array);
}

//A specific sorting mechanism
public class InsertionSort implements Sorter
{
   public void sort(Comparable[] array){
      int n = array.length;

      for (int j = 1; j < n; j++) {
         Comparable k = array[j];
         // Insert array[j] into the sorted sequence array[0..j-1].
         int i = j-1;

         while (i >= 0 && array[i].compareTo(k) > 0) {
            array[i+1] = array[i];
            i--;
         }
         array[i+1] = k;
      }
   }
}
1.2 A general definition of a comparable type

//****************************************************************************
// Contact.java Author: Lewis/Loftus
//
// Represents a phone contact.
//****************************************************************************

//A specific type that could be sorted
public class Contact implements Comparable
{
    private String firstName, lastName, phone;

    //-----------------------------------------------------------------
    // Sets up this contact with the specified information.
    //-----------------------------------------------------------------
    public Contact(String first, String last, String telephone)
    {
        firstName = first;
        lastName = last;
        phone = telephone;
    }

    //-----------------------------------------------------------------
    // Returns a description of this contact as a string.
    //-----------------------------------------------------------------
    public String toString()
    {
        return lastName + ", " + firstName + "\t" + phone;
    }

    //-----------------------------------------------------------------
    // Uses both last and first names to determine lexical ordering.
    //-----------------------------------------------------------------
    public int compareTo(Object other)
    {
        int result;

        if (lastName.equals(((Contact)other).lastName))
            result = firstName.compareTo(((Contact)other).firstName);
        else
            result = lastName.compareTo(((Contact)other).lastName);

        return result;
    }
}
1.3 A driver

The following driver should work for the above two classes.

```java
//*****************************************************
// SortPhoneList.java
//
// Driver for testing an object sort.
//********************************************************

//package com.mhhe.clrs2e;

public class SortPhoneList
{
    //-------------------------------------------------------------
    // Creates an array of Contact objects, sorts them, then prints
    // them.
    //-------------------------------------------------------------
    public static void main (String[] args)
    {
        Contact[] friends = new Contact[7];

        friends[0] = new Contact("John", "Smith", "610-555-7384");
        friends[2] = new Contact("Mark", "Riley", "733-555-2969");

        InsertionSort sorter = new InsertionSort();

        sorter.sort(friends);

        for (int index = 0; index < friends.length; index++)
            System.out.println(friends[index]);
    }
}
```


2 Assignments

1. Study and compare the code of the InsertionSort as shown in Sec. 1.1, with the algorithm as presented in pp. 18 of the textbook.

2. Run the driver, as shown in Sec. 1.3 to test the InsertionSort code.

3. Write and test another class, SelectionSort algorithm implementing the selection sort algorithm \(^1\), as discussed in Exercise 2.2-2. Play with it with some sample data yourself, until you understand its mechanism.

4. Implement the MergeSort algorithm, as discussed in Sec. 2.3 of the textbook \(^2\). You might want to check the video demonstration of this algorithm on the course page, and also play with it with some sample data yourself, until...

5. Implement yet another sorting mechanism, the BubbleSort algorithm, as discussed in pp. 40 (Problem 2.2). You might want to check the video demonstration of this algorithm, as shown on the course page, and also play with it with some sample data yourself, until...

6. Thoroughly test out all of the works that you have done.

3 What should be sent in?

Email me, by the deadline, the whole Java program, in .java, including the InsertionSort, SelectionSort, BubbleSort, and the MergeSort classes, together with a driver. Also send in a lab report, showing your testing results of the program.

References


---

\(^1\)We discussed it in a class in the setting of Divide’n Conquer.

\(^2\)For an alternative implementation of the Merge procedure, check out Exercise 2.3-2.