Project 2: A Bunch of 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...

Here is a complete Java program, consisting of four files, to sort, using Insertion sort, a list of items of the Contact type.

1.1 An interface

As we discussed in class, there could be multiple ways to sort things out. It might be convenient to come up with an interface, which can then be implemented in various ways.

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

1.2 What is Comparable?

A sorting mechanism can sort a collection of anything that can be compared, e.g., a list of names.

//*******************************************************************************
// 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 Insertion sort

We went through in class the details of InsertionSort, which is included in the Java Package that comes with the text book by Cormen et al [1], with most of the comments taken off.

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

        for (int j = 1; j < n; j++) {
            // Insert element into its place in the sorted portion of array
            int i = j - 1;
            Comparable current = array[j];
            while (i >= 0 && array[i].compareTo(current) > 0)
            {
                array[i+1] = array[i];
                i--;
            }
            array[i+1] = current;
        }
    }
}
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;
}
}

1.4 A driver

Finally, we need a driver to complete the program.

//********************************************************************
// 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");
//Let’s try insertion sort first...
InsertionSort sorter = new InsertionSort();
//SelectionSort sorter = new SelectionSort();
//MergeSort sorter = new MergeSort();
//BubbleSort sorter = new BubbleSort();

sorter.sort(friends);

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

1.5 The output

When you run the above program, you should get the following back:

Barnes, Sarah 215-555-3827
Getz, Laura 663-555-3984
Grant, Marsha 243-555-2837
Phelps, Frank 322-555-2284
Riley, Mark 733-555-2969
Smith, John 610-555-7384
Smith, Larry 464-555-3489
Press any key to continue...

2 Assignments

1. Study and compare the code of the InsertionSort as shown in Sec. 1.3, with the algorithm as presented on page 18 of the textbook.

2. Collect the four files into a program, as shown in Section 1. Run the program, and make sure it brings back the result as shown in Section 1.5.

3. Write and test another class, SelectionSort algorithm, that implements 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...

¹We discussed it in a class in the setting of Divide’n Conquer.
²For an alternative implementation of the Merge procedure, check out Exercise 2.3-2.
5. Implement yet another sorting mechanism, the *BubbleSort* algorithm, as discussed on page 40 (Problem 2.2). You might also 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 program*, in `.java`, including the `InsertionSort`, `SelectionSort`, `BubbleSort`, and the `MergeSort` classes, together with the interface file, the `Comparable` implementation, and a driver.

Also send in a lab report, showing your testing results of the program.

### 4 A general grading guideline

≥ 1: A syntactically correct and executable program is implemented for the insertion sorting algorithm.

≤ 4: Each of the three sorting algorithms, i.e., selection sort, mergesort and bubble sort, is correctly implemented. Each counts one point.

≤ 5: A lab report containing all the testing results is sent in.

### References