Laboratory Assignment № 10

Dr. Duncan, CSC 1350, Louisiana State University 11/26/2019

Profiling the Bubble Sort Algorithm

Learning Objectives

- Using an Elementary Sorting Algorithm and
- Implementing and Modifying the Bubble Sort Algorithm

In this laboratory exercise you will implement several variants of the bubble sort algorithm.

The *Sorter* Class

Define a class called *Sorter* that consists of one method, the standard implementation of the bubble sort algorithm below. Provide Javadoc documentation for both the class and the method.

Listing 1: An Implementation of the Selection Sort Algorithm

```java
/**
 * Sorts an array of integers in non-decreasing order
 * @param data an array of integers
 * @version 1
 */
public static void bubbleSort(int[] data)
{
   boolean notFinished ;
   int partitionIndex = data.length-1, temp;
   do
   {
      notFinished = false;
      for (int i = 0; i < partitionIndex; i++)
      {
         if (data[i] > data[i+1])
         {
            temp = data[i];
            data[i] = data[i+1];
            data[i+1] = temp;
            notFinished = true;
         }
      }
      partitionIndex--;
   }while(notFinished);
}
```

The standard implementation of the bubble sort algorithm initially puts all the elements of the array in the unsorted region. Observe that during each pass, iteration of the outer loop, the algorithm partitions the array into an unsorted region and a sorted region. The unsorted region is left region and the sorted region is the right region. After each pass, the size of the unsorted region shrinks by 1 element and the size of the sorted region expands by 1 element. The algorithm places the next largest element at the head of the sorted region after each pass.

Whenever you modify the *bubbleSort* method, use the *@version* tag in your javadoc to indicate the version of the method. For example, the initial version of the code will have the " **@version 1**" included as the last line in its Javadoc along with the other lines of the Javadoc of the method.

The *BubbleSortTester* Class

Define the *BubbleSortTester* class that consists of only one method, the *main*. In the main method, define an integer array data = [2, 4, 5, 6, 4, 5, 3, 5, 3], print initial contents of the array using the *Arrays.toString* method in the standard Java API, then sort the array using the *bubbleSort* method defined in the *Sorter* class and print the sorted array. If you have done things properly, the output of the program should be:

Listing 2: Sample Run

```
Initial Data: [2, 4, 5, 6, 4, 5, 3, 5, 3]
Sorted Data: [2, 3, 3, 4, 4, 5, 5, 5, 6]
```

First Version

Copy the *bubbleSort* method above and paste it in the *Sorter* class as a second method, version 2, in the *Sorter* class. Comment out version 1 of the method. Modify version 2 of the method that you copied so that it prints the array after each pass as shown in Listing 3.

Listing 3: Sample Run

```
Initial Data: [2, 4, 5, 6, 4, 5, 3, 5, 3]

[2, 4, 5, 4, 5, 3, 5, 3, 6]
[2, 4, 4, 5, 3, 5, 3, 5, 6]
[2, 4, 4, 3, 5, 3, 5, 5, 6]
[2, 4, 3, 4, 3, 5, 5, 5, 6]
[2, 3, 4, 3, 4, 5, 5, 5, 6]
[2, 3, 3, 4, 4, 5, 5, 5, 6]
[2, 3, 3, 4, 4, 5, 5, 5, 6]

Sorted Data: [2, 3, 3, 4, 4, 5, 5, 5, 6]
```

Second Version

Again, copy version 2 of the bubbleSort method and paste it as the third method, version 3, in the *Sorter* class. Comment out version 2 of the *bubbleSort* method. Modify version 3 so that it prints the pairs of elements that are compared during each pass. After the modification, the output of the program will be as shown in Listing 4:

Listing 4: Sample Run

```
Initial Data: [2, 4, 5, 6, 4, 5, 3, 5, 3]

(2,4) (4,5) (5,6) (6,4) (6,5) (6,3) (6,5) (6,3)
[2, 4, 5, 4, 5, 3, 5, 3, 6]
(2,4) (4,5) (5,4) (5,5) (5,3) (5,5) (5,3)
[2, 4, 4, 5, 3, 5, 3, 5, 6]
(2,4) (4,4) (4,5) (5,3) (5,5) (5,3)
[2, 4, 4, 3, 5, 3, 5, 5, 6]
(2,4) (4,4) (4,3) (4,5) (5,3)
[2, 4, 3, 4, 3, 5, 5, 5, 6]
(2,4) (4,3) (4,4) (4,3)
[2, 3, 4, 3, 4, 5, 5, 5, 6]
(2,3) (3,4) (4,3)
[2, 3, 3, 4, 4, 5, 5, 5, 6]
(2,3) (3,3)
[2, 3, 3, 4, 4, 5, 5, 5, 6]

Sorted Data: [2, 3, 3, 4, 4, 5, 5, 5, 6]
```

Third Version

Copy version 3 of the *bubbleSort* method and paste it as the fourth method, version 4, in the *Sorter* class. Comment out version 3 of the *bubbleSort* method. Modify version 4 so that it sorts the array in non-increasing order and prints the pairs of elements that are compared during each pass. After these modifications, the output of the program will be as shown in Listing 5.

Listing 5: Sample Run

```
Initial Data: [2, 4, 5, 6, 4, 5, 3, 5, 3]

(2,4) (2,5) (2,6) (2,4) (2,5) (2,3) (2,5) (2,3)
[4, 5, 6, 4, 5, 3, 5, 3, 2]
(4,5) (4,6) (4,4) (4,5) (4,3) (3,5) (3,3)
[5, 6, 4, 5, 4, 5, 3, 3, 2]
(5,6) (5,4) (4,5) (4,4) (4,5) (4,3)
[6, 5, 5, 4, 5, 4, 3, 3, 2]
(6,5) (5,5) (5,4) (4,5) (4,4)
[6, 5, 5, 5, 4, 4, 3, 3, 2]
(6,5) (5,5) (5,5) (5,4)
[6, 5, 5, 5, 4, 4, 3, 3, 2]

Sorted Data: [6, 5, 5, 5, 4, 4, 3, 3, 2]
```

Final Version

Finally, copy version 4 of the *bubbleSort* method and paste it as the fifth method, version 5, in the *Sorter* class. Comment out version 4 of the *bubbleSort* method. Modify version 5 so that in addition to sorting the array in non-increasing order, it scans the array leftward. The sorted region of the array will be on the left and the unsorted region will be on the right. After each pass, the largest number in the unsorted region is moved to the tail of the sorter region. It should also print the pairs of elements that are compared during each pass. After these modifications, the output of the program will be as shown in Listing 6.

Listing 6: Sample Run

```
Initial Data: [2, 4, 5, 6, 4, 5, 3, 5, 3]

(5,3) (3,5) (5,5) (4,5) (6,5) (5,6) (4,6) (2,6)
[6, 2, 4, 5, 5, 4, 5, 3, 3]
(3,3) (5,3) (4,5) (5,5) (5,5) (4,5) (2,5)
[6, 5, 2, 4, 5, 5, 4, 3, 3]
(3,3) (4,3) (5,4) (5,5) (4,5) (2,5)
[6, 5, 5, 2, 4, 5, 4, 3, 3]
(3,3) (4,3) (5,4) (4,5) (2,5)
[6, 5, 5, 5, 2, 4, 4, 3, 3]
(3,3) (4,3) (4,4) (2,4)
[6, 5, 5, 5, 4, 2, 4, 3, 3]
(3,3) (4,3) (2,4)
[6, 5, 5, 5, 4, 4, 2, 3, 3]
(3,3) (2,3)
[6, 5, 5, 5, 4, 4, 3, 2, 3]
(2,3)
[6, 5, 5, 5, 4, 4, 3, 3, 2]

[6, 5, 5, 5, 4, 4, 3, 3, 2]

Sorted Data: [6, 5, 5, 5, 4, 4, 3, 3, 2]
```

Additional Requirements

In addition to the initial array shown in the sample run for the final version, test the program using: list = [1, 2, 3, 4, 5, 6, 7, 8, 9] and list = [9, 8, 7, 6, 5, 4, 3, 2, 1]. Verify that the results generated by the program are accurate. Your program should have declarations for three arrays: the one used in the sample run and the two additional ones. Comment out the two additional ones when submitting your code. Exhaustively, test your program to ensure that it works. Make sure both classes and the *bubbleSort* methods are documented. You do not need to provide Javadoc for the main method.

Some Netbeans Keyboard Shortcuts

To execute these keyboard sequences, hold down the first key and then press the second key.

1. $\boxed{\text{CTRL}} + \boxed{\text{C}}$: To copy a code block to the clipboard, highlight the code block and then press this sequence.
2. $\boxed{\text{CTRL}} + \boxed{\text{V}}$: To paste the last copied code block from the clipboard, move the cursor to where you want to paste the code block and then press this sequence. The contents of the clipboard is then copied beginning at the location of the cursor.
3. $\boxed{\text{CTRL}} + \boxed{\text{/}}$: To comment out a code block, highlight the code block and then press this sequence. This is a toggle key sequence: if the code block is already commented out, highlighting the code block and pressing this key sequence will *uncomment* it.

Submitting Your Work for Grading

Navigate your way to the ...\NetBeansProjects\BubbleSortTester\src\bubblesorttester folder using Windows file explorer. You should find *Sorter.java* and *BubbleSortTester.java*, files containing your source code for the program. Click one of the files and then hold down the shift key and click the other so that both files are selected. Right-click the selected files and create a compressed (zipped) folder containing a copy of each file. Rename the zip file *PAWSID_lab10.zip*, where *PAWSID* is the prefix of your LSU/Tiger email address - the characters left of the @ sign. Double-click the zip file to verify that both *Sorter.java* and *BubbleSortTester.java* are included the zip file. If the zip file does not contain both files, delete the zip file and repeat the steps. Upload the zip file to the digital drop box on Moodle.