10.5 Bubble Sort

Our next (but not our last) sorting algorithm, called a bubble sort, is an example of a class of sorts known as exchange sorts. With the bubble sort, the basic idea is to compare adjacent values and exchange them if they are not in order.

Example 1

Suppose that we want to use a bubble sort to arrange the names

```
               Phil        Ivan        Sara        Jack        Gina
```

in alphabetical order. We start by comparing Phil to Ivan. Since they are

not in order, we exchange them to give

```
               Ivan        Phil        Sara        Jack        Gina
```

Next we compare Phil to Sara. Since they are in order, we leave them that way giving

```
               Ivan        Phil        Sara        Jack        Gina
```

Now, comparing Sara to Jack, we see that they must be exchanged. Doing this gives

```
               Ivan        Phil        Jack        Sara        Gina
```

We then compare Sara to Gina. Again, we must perform an exchange to obtain

```
               Ivan        Phil       Jack        Gina        Sara
```

The result of all this comparing and exchanging is that, after one pass, the largest value (Sara, in our example) will be at the upper end of the list. Like a bubble rising in a liquid, the largest value has risen to the top of the
list.

As with the selection sort shown in the previous section, we now repeat the process used in the first pass on all but the last element. As we proceed, the passes deal with shorter and shorter subsequences of the original list until all values are in their correct positions.

Example 2

The following tables show the actions of a bubblesort in ordering the names

```
Phil      Ivan      Sara      Jack      Gina
```

The colons indicate the values currently being compared. The vertical bars indicate the division points between the unsorted data and the sorted data.

```
                         Phil   :  Ivan      Sara      Jack      Gina
                         Ivan      Phil   :  Sara      Jack      Gina
    First Pass           Ivan      Phil      Sara   :  Jack      Gina
                         Ivan      Phil      Jack      Sara   :  Gina
                         Ivan      Phil      Jack      Gina   |  Sara

                         Phil   :  Ivan      Sara      Jack   |  Gina
                         Ivan      Phil   :  Sara      Jack   |  Gina
    Second Pass          Ivan      Phil      Sara   :  Jack   |  Gina
                         Ivan      Phil      Jack   |  Sara      Gina


                         Phil   :  Ivan      Sara   |  Jack      Gina
    Third Pass           Ivan      Phil   :  Sara   |  Jack      Gina
                         Ivan      Phil   |  Sara      Jack      Gina



                         Phil   :  Ivan      Sara   |  Jack      Gina
     Forth Pass          Ivan      Phil      Sara      Jack      Gina
```

As we saw with selection sort, the number of passes needed to sort the items is one less than the number of items. After all but one of the items have been ordered, the remaining one must also be in its correct position. To create a Java method for a bubble sort is fairly easy if we use some of our previous techniques. To perform all the passes we need a loop like that of the selection sort.

```java
for (int top = list.length-1; top> 0; top--)
 // compare adjacent values of the unsorted sublist
 // from 0 to top, exchanging as necessary
```

Within each pass, the code for comparing and exchanging values in a sublist is easily written.

```java
for (int i = 0; i < top; i++)
if (list[i] .compareTo(list[i+1]) > 0)
{
temp = list [i];
list[i] = list[i+1];
list[i+1] = temp;
}
```

Combining these fragments of code and inserting them into a method gives us a basic bubble sort. We can improve this sort's performance by noting that, on each pass, we often do more than simply place the greatest value at the upper end of a sublist; the comparisons and exchanges tend to push all values toward their final positions in the list. It may happen that this shuffling of values puts all the values in their final positions before all passes have been completed. In such a situation, we should stop working as soon as possible, avoiding any unnecessary passes. As it turns out, this is not too difficult to do. Before we incorporate this modification, we must first answer the following question: how do we know if there is no more work to be done? The answer is: if no exchanges are performed in an entire pass, then the items must be fully sorted. The reasoning here is that, if no exchanges are needed, then each value must be correctly ordered between its immediate neighbours. It follows that the entire set of values must, therefore, be completely ordered. Thus we need to perform another pass only if the previous one carried out one or more exchanges. To encode this idea in Java, the for statement that controls the number of passes should now incorporate a test to see if another pass is necessary. We do this with a boolean variable called sorted. Initially, sorted is set to false; we do not start a pass unless sorted is false. Once we have started a pass, we (optimistically) set sorted to true because no exchanges have yet been performed during that pass. If any exchanges are required during a pass, we reset sorted to false. Processing continues until sorted remains true after a full pass or we have completed the maximum number of passes.

Example 3

This method uses a bubble sort to place an array of strings in ascending

order. It uses a boolean flag, sorted, to make the bubble sort more efficient

.. by stopping the sort after a full pass in which there are no exchanges.

```
      public static void bubbleSort(String[] list) {
              boolean sorted = false;
              for (int top = list.length - 1; top > 0 && !sorted; top--) {
                      sorted = true;
                      for (int i = 0; i < top; i++)
                              if (list[i].compareTo(list[i + 1]) > 0) {
                                      sorted = false;
                                      String temp = list[i];
                                      list[i] = list[i + 1];
                                      list[i + 1] = temp;
                              }
              }
        }
```

Although bubble sort uses an interesting technique and has a cute name, we do not recommend it. It is almost always slower than either of the sorts that we have already examined because it usually involves far more data movement than they require.

Exercise 10.5

1. Make tables like those shown in Example 2 to show the comparisons and exchanges that would take place in using a bubble sort to put the following data in ascending

order.

3 8 3 2 7 5

First Pass	3,3,8,2,7,5 3,3,2,8,7,5 3.3.2.7.8.5 3.3.2.7.5.8
Second Pass	3,2,3,7,5,8 3.2.3.7.5.8 3.2.3.5.7.8
Third Pass	2,3,3,5,7,8

Code:

```
public class Question1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {3,8,3,2,7,5};
		bubbleSort (list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");

	}

	public static void bubbleSort(double [] list) {
		boolean sorted = false;
		for (int top = list.length - 1; !sorted && top > 0; top--) {
			sorted = true;
			for (int i = 0; i < top; i++) {
				if (list[i] > list[i+1]) {
					sorted = false;
					double temp = list[i];
					list[i] = list[i+1];
					list[i+1] = temp;
				}
			}
		}
	}
}
```

2. What changes would have to be made to the bubbleSort method in order to make it sort values in descending order?

   To make the method sort values in descending order you simply change the sign in the `if` statement from `if (list[i] > list[i+1])` to `if (list[i] < list[i+1])`. Hence, in every pass, the smallest value will be stored at the very end of the array.

   Code:

```
public class Question2 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {3,8,3,2,7,5};
		bubbleSort (list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");

	}

	public static void bubbleSort(double [] list) {
		boolean sorted = false;
		for (int top = list.length - 1; !sorted && top > 0; top--) {
			sorted = true;
			for (int i = 0; i < top; i++) {
				if (list[i] < list[i+1]) {
					sorted = false;
					double temp = list[i];
					list[i] = list[i+1];
					list[i+1] = temp;
				}
			}
		}
	}
}
```

3. A modification of the bubble sort is the cocktail shaker sort in which, on odd-numbered passes, large values are carried to the top of the list while, on even-numbered passes, small values are carried to the bottom of the list. Make tables like those shown in Example 2 to show the first two passes of a cocktail-shaker sort on the following data.

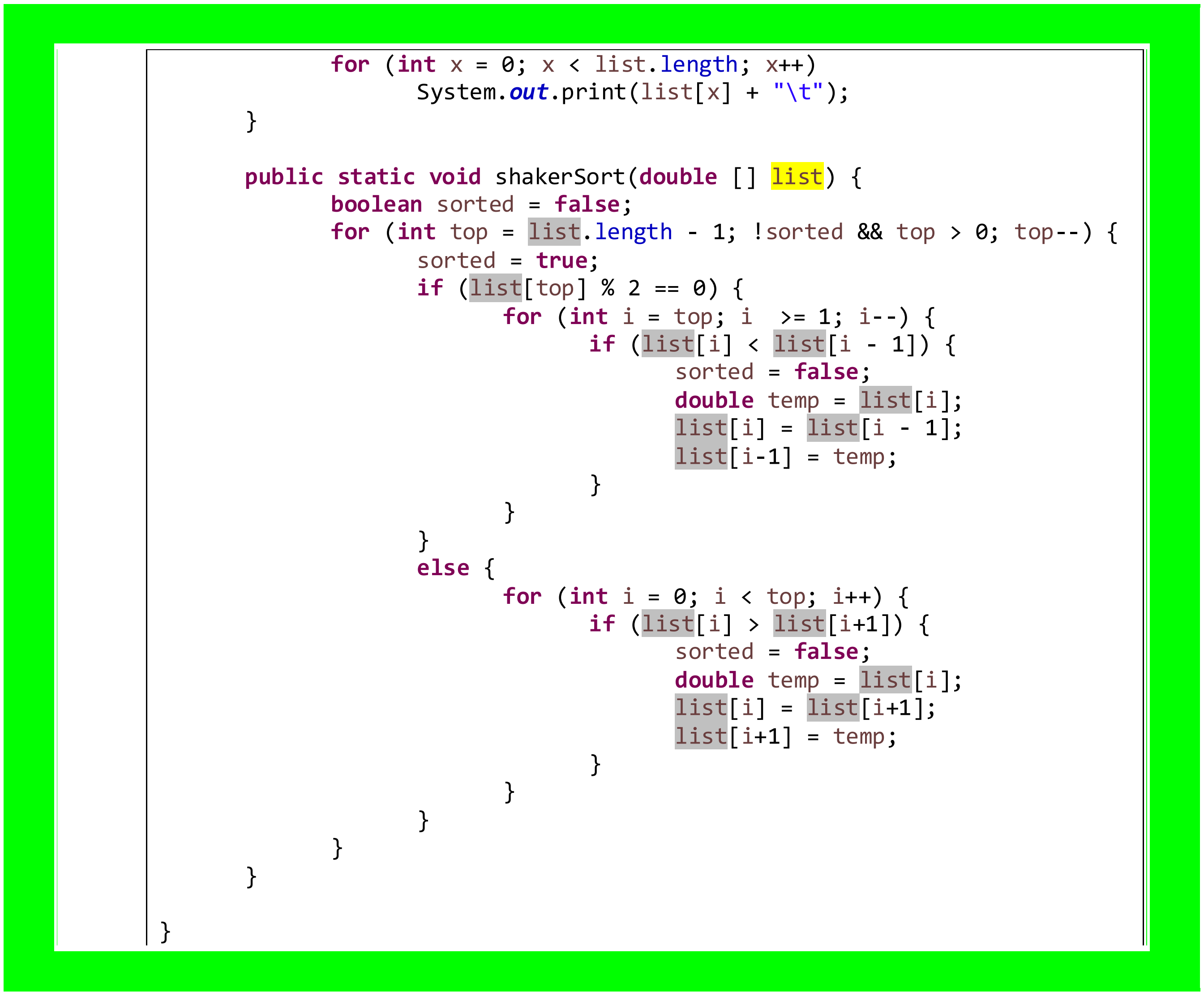
   2 9 4 6 1 7

Code:

```
public class Question3 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {2,9,4,6,1,7};
		bubbleSort (list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");
	}

	public static void bubbleSort(double [] list) {
		boolean sorted = false;
		for (int top = list.length - 1; !sorted && top > 0; top--) {
			sorted = true;
			if (list[top] % 2 == 0) {
				for (int i = top; i  >= 1; i--) {
					if (list[i] < list[i - 1]) {
						sorted = false;
						double temp = list[i];
						list[i] = list[i - 1];
						list[i-1] = temp;
					}
				}
			}
			else {
				for (int i = 0; i < top; i++) {
					if (list[i] > list[i+1]) {
						sorted = false;
						double temp = list[i];
						list[i] = list[i+1];
						list[i+1] = temp;
					}
				}
			}
		}
	}
}
```

4. Write a method shakerSort to implement the cocktail shaker sort algorithm to arrange an array of double values in ascending order. Use a boolean flag to stop processing once the items have been completely sorted.

```
public class Question4 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {2,9,4,6,1,7};
		shakerSort (list);
```



```java
        for (int x = 0; x < list.length; x++)
            System.out.print(list[x] + "\t");
    }

    public static void shakerSort(double [] list) {
        boolean sorted = false;
        for (int top = list.length - 1; !sorted && top > 0; top--) {
            sorted = true;
            if (list[top] % 2 == 0) {
                for (int i = top; i  >= 1; i--) {
                    if (list[i] < list[i - 1]) {
                        sorted = false;
                        double temp = list[i];
                        list[i] = list[i - 1];
                        list[i-1] = temp;
                    }
                }
            }
            else {
                for (int i = 0; i < top; i++) {
                    if (list[i] > list[i+1]) {
                        sorted = false;
                        double temp = list[i];
                        list[i] = list[i+1];
                        list[i+1] = temp;
                    }
                }
            }
        }
    }

}
```