Searching & Sorting in Java – Bubble Sort

With the bubble sort, the basic idea is to compare adjacent values and exchange them if they are not in order. Consider the following example which shows the first pass through the algorithm.

1. Compare 6 and 3	6	3	5	8	2
2. 6 is higher, so swap 6 and 3	3	6	5	8	2
3. Compare 6 and 5	3	6	5	8	2
4. 6 is higher, so swap 6 and 5	3	5	6	8	2
5. Compare 6 and 8	3	5	6	8	2
6. 8 is higher, so no swap is performed	3	5	6	8	2
7. Compare 8 and 2	3	5	6	8	2
8. 8 is higher, so swap 8 and 2	3	5	6	2	8
9. The largest value, 8, is at the end of the array	3	5	6	2	8

The result of this comparing and exchanging is that , after one pass, the largest value will be at the upper end of the list. Like a bubble, it has risen to the top. On our next pass, we can ignore this position and work with a shortened list, allowing the largest remaining value to rise to the (slightly lower) top.

The complete bubble sort is shown below in a more condensed format.

First Pass	6	3	5	8	2
	3	6	5	8	2
	3	5	6	8	2
	3	5	6	8	2
	3	5	6	2	8
Second Pass	3	5	6	2	8
	3	5	6	2	8
	3	5	6	2	8
	3	5	2	6	8
Third Pass	3	5	2	6	8
	3	5	2	6	8
	3	2	5	6	8
Fourth Pass	3	2	5	6	8
	2	3	5	6	8

Searching & Sorting in Java – Bubble Sort

To code this algorithm in Java, we start with a loop very similar to that used for the selection sort.

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

It is worth noting that, on each pass, the exchanges tend to move all values toward their final positions in the list. Thus it is possible for the list to be sorted before the final pass is completed. For the sake of efficiency, we should stop working as soon as possible to avoid any unnecessary passes.

In a sorted list, the bubble sort algorithm would not perform any exchanges, so we use a `boolean` flag to monitor this condition. The `boolean` value `sorted` is initialized to `false` so the initial pass will begin. At the beginning of each pass, the value is set to `true`, but if a single swap is required, it is set back to `false`.

Processing continues until `sorted` stays `true`, or we have completed the maximum number of passes.

The full method, for an array of strings, is as follows.

```
public static void bubbleSort (String[] list)
{
      boolean sorted = false;

      for (int top = list.length – 1; top > 0; top--)
      {
            sorted = true;
            for (int i = 1; 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;
                  }
            }
      }
}
```

In general, bubble sort is not recommended for any serious applications. It is almost always slower than insertion and selection sorts because it usually involves far more data movement than they require.

Exercises

1. 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

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

3. A variation 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. On even-numbered passes, small values are carried to the bottom of the list. Show the first two passes on the following data.

   2 9 4 6 1 7

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.