Sorting – Insertion Sort

To develop a solution to the problem of sorting data, let us look at how we might proceed if we were playing a card game and we wanted to put our hand in order, from smallest to largest. A common way of sorting such a hand is to examine cards from left to right, rearranging them if necessary by placing cards where they belong in the sequence of cards on their left. The next example illustrates the process.

Suppose that we have the five cards shown below (ignoring suits).
6, 3, 5, 8, 2

We could begin to sort these cards by looking at the 3 (the second card from the left). Since 3 < 6, the 3 should be inserted to the left of the 6. This will produce the following arrangement in which the two values on the left are in order.
3, 6, 5, 8, 2

The next card from the left, the 5, should be inserted between the 3 and the 6. Doing this gives us the next arrangement in which the first three cards are guaranteed to be in order.
3, 5, 6, 8, 2

Now we examine the 8. Because it is greater than any of the values to its left, it should stay where it is and still give us the four leftmost values in order.
3, 5, 6, 8, 2

Finally, looking at the 2, we can see that it must be inserted before any of
the other values. Doing this gives us our final, ordered arrangement.
2, 3, 5, 6, 8

This sorting algorithm is called **insertion sort**. To implement this algorithm for an array of values, we must examine and correctly insert each of the values in the array from the second position up to the last one. This requires a loop like the following, for an array called list containing listLength items.

```
for (int current = 1; current < listLength; current++)
{
    // insert element at current into its correct position
    // among the elements from 0 to top - 1
}
```

At each stage or pass of the sort, we first copy the element that we want to insert into a temporary location. We then move from right to left through the (already sorted) items on the left of the value that we are inserting. If a value in the list is larger than the new one, it is moved one space to the right (to provide room for the new item). Once the correct location of the new value is found, it is then inserted back into the array from the temporary location in which it had been saved.

Look at these web sites for a visual demo of the insertion sort.

http://math.hws.edu/TMCM/java/xSortLab/

http://www.youtube.com/watch?v=ROalU379l3U

Sorting – Insertion Sort

The method insertionSort uses an insertion sort to arrange an array of
double values in ascending order.

```
public static void insertionSort (double[] list)
{
    int top;
    int i;

    // start from the second item in the array
    for (top = 1; top < list.length; top++)
    {
        // copy the next item to a temporary location
        double item = list[top];
        i = top;

        // shift items to the right if next item is smaller
        // than previous item
        while (i > 0 && item < list[i-1])
        {
            list[i] = list[i-1];
            i--;
        }
        list[i] = item;
    }
}
```

<u>Exercises</u>

1) Use insertion sort to put the values 6, 2, 8, 3, 1, 7, 4 in ascending (lowest to highest) order. **Modify the code above to show the values** as they would appear after each pass of the sort.

2) Write a program that will sort (using insertion sort) 20 random numbers with values between 1 and 60. The program will graphically output to the screen as a series of * for the value of array element. Show the * as the data is being sorted after each pass of the sort.
You can use the **Console** to print the *'s. You will have to add a delay and a clear screen between each pass so that the sort is slow enough to see.

Ie , suppose the data is 10, 5, 3, 6. The initial screen would be:

```
**********
*****
***
******
```

After the first pass:

```
*****
**********
***
******
```

After the second pass:

```
***
*****
**********
******
```

After the third pass:

```
***
*****
******
**********
```