Selection Sort

The algorithm to sort n numbers is as follows:

For the ith element (as i ranges from n-1 down to 1)

1) Determine the largest element in the array to from index 0 to index i.
2) Swap the current ith element with the element identified in step 1.

In essence, the algorithm first picks the largest element and swaps it into the last location, then it picks the next largest element and swaps it into the next location, etc.

In order to do #1, iterate through the array keeping track of the index that is currently storing the minimum value.

Let's look at an example of selection sort on the list:

8, 3, 1, 9, 5, 2

We iterate through this list starting at the first element all the way through until we determine that the largest element is stored in index 3. We then swap this with the element at index 5 to yield:

8, 3, 1, 2, 5, 9. Similarly, we find the last element and swap it:
5, 3, 1, 2, 8, 9. Then,
2, 3, 1, 5, 8, 9. Followed by
2, 1, 3, 5, 8, 9, and finally
1, 2, 3, 5, 8, 9

Insertion Sort

In this sort, we take each element one by one, starting with the second, and "insert" it into a sorted list. The way we insert the element is by continually swapping it with the previous element until it has found its correct spot in the already sorted list. Here is the algorithm for sorting n elements:

```
For the ith element (as i ranges from 1 to n-1)
    1) As long as the current element is greater than the
         previous one, swap the two elements. Stop if there's no
       previous element.
```

Consider using this algorithm to sort the following list:

3, 7, 2, 1, 5

Here are all the passes of the algorithm:

3, 7, 2, 1, 5, since 7 > 3, so the list 3, 7 is sorted.

3, 2, 7, 1, 5, since 2 < 7
2, 3, 7, 1, 5, since 2 < 3, so the list 2, 3, 7 is sorted.

2, 3, 1, 7, 5, since 1 < 7,
2, 1, 3, 7, 5, since 1 < 3,
1, 2, 3, 7, 5, since 1 < 2, so the list 1, 2, 3, 7 is sorted.

1, 2, 3, 5, 7, since 5 < 7 and now we can stop since 5 > 3.

You'll notice that the number of steps in this algorithm VARIES depending on the input...

Bubble Sort

The basic idea behind bubble sort is that you always compare consecutive elements, going left to right. Whenever two elements are out of place, swap them. At the end of a single iteration, the maximum element will be in the last spot. Now, just repeat this n times, where n is the number of elements being sorted. On each pass, one more maximal element will be put in place.

Consider the following trace through:

Original list:

6, 2, 5, 7, 3, 8, 4, 1

On a single pass of the algorithm, here is the state of the array:

2, 6, 5, 7, 3, 8, 4, 1 (The swapped elements are underlined)
2, 5, 6, 7, 3, 8, 4, 1
2, 5, 6, 7, 3, 8, 4, 1 (No swap occurs here, since the are in order)
2, 5, 6, 3, 7, 8, 4, 1
2, 5, 6, 3, 7, 8, 4, 1 (No swap)
2, 5, 6, 3, 7, 4, 8, 1
2, 5, 6, 3, 7, 4, 1, 8 (8 is now in place!)

On the next iteration, we can stop before we get to 8. Each subsequent iteration can stop one spot earlier on the list.

Note that after each iteration, we are guaranteed that the "next maximum element" is in place, much like Selection Sort.

Here is what the array looks like after each subsequent iteration.

2, 5, 3, 6, 4, 1, 7, 8 (end of 2nd iteration, 7 is in place now)
2, 3, 5, 4, 1, 6, 7, 8 (end of 3rd iteration, 6 is in place now)
2, 3, 4, 1, 5, 6, 7, 8 (end of 4th iteration, 5 is in place now)
2, 3, 1, 4, 5, 6, 7, 8 (end of 5th iteration, 4 is in place now)
2, 1, 3, 4, 5, 6, 7, 8 (end of 6th iteration, 3 is in place now)
1, 2, 3, 4, 5, 6, 7, 8 (end of 7th iteration both 2 and 1 are in place now)