Sorting: The process of organizing a collection of data into either ascending or descending order. Sorted lists are often formed from unsorted ones to facilitate the searching process.

Insertion Sort

Methodology: create a sorted array by inserting 1 item at a time into the correct location. Assume that the first element in the array is in the correct place. Compare the second element to the first, and insert either before or after it. After this first pass, the first 2 numbers will be sorted. Then compare the 3rd number to the 2nd. If it’s less, then shift the 2nd number over to the right and compare the 3rd to the first. Continue this process of comparing, shifting, and inserting until the whole list is sorted.

Algorithm:
Insertion_Sort ( array A )

for each array starting from the second element (i.e A[1]) till the last

```
current = A[j]

i ← j - 1
while (i >= 0 and A[i] > current)
    shift A[i] to the right in the array
    i = i-1
endwhile
A[i+1] = current
```

Example:

```
29 10 14 37 13 // current = 10
29 29 14 37 13 // 29 > current so shift 29 to the right
10 29 14 37 13 // insert current at start of list

10 29 14 37 13 // current = 14
10 29 29 37 13 // 29 > current so shift 29 to the right
10 14 29 37 13 // 10 < current so insert current

10 14 29 37 13 // current = 37
10 14 29 37 13 // 29 < current so current stays where it is

10 14 29 37 13 // current = 13
10 14 29 37 37 // 37 > current so shift 37 to the right
10 14 29 29 37 // 29 > current so shift 29 to the right
10 14 14 29 37 // 14 > current so shift 14 to the right
10 13 14 29 37 // 10 < 13 so insert current
```

Analysis: $O(n^2)$ is the worst case growth rate. This happens when the list is in reverse sorted order (largest to smallest). Requires $O(n^2)$ for comparisons and $O(n^2)$ for moves. For arrays in close to sorted order, this algorithm works well: $O(n)$ if already sorted.
Selection Sort
Methodology: use a two-phase process to (1) find the smallest element in the unsorted portion of the list, and (2) switch that element with the one in the first position in the unsorted portion. That element plus anything to its left now constitute the sorted portion. Continue this process until the whole list is sorted.

Note: a symmetric version selects the largest element in the unsorted portion and switches the largest element with the one in the last position of the unsorted portion (T.B.)

Algorithm Summary:
Pick the largest element and put it in place. Then find the largest of the remaining elements.
For an array A with n elements:
Do the following steps n-1 times, starting with count = 0
• Find the smallest item in the unsorted portion of the list, between positions count and n and note its position as ind_min
• Swap the item in position ind_min with the item in position count
• count = count +1

Analysis: Requires fewer moves but more comparisons than Insertion sort. Independent of how well sorted the initial list is.
Comparisons: n-1 + n-2 + ... + 2 + 1 = n*(n-1)/2 = O(n²)

Bubble Sort (Exchange Sort)
Methodology: Compare 2 successive elements and swap them if necessary to put the largest element after the smaller one. Larger elements “bubble” to the end of the list.

Bubble Sort Summary
During each of (n-1) passes through the (unsorted portion of the) array:
• Compare item in position i with item in position i+1.
• If it is larger, swap the two items.

As the result of one pass, the largest element of the unsorted portion is placed on its correct position according to the sorted order, so the unsorted portion of the array “shrinks” by one element on the right hand side.
Example:

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Pass 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 10 14 37 13</td>
<td>10 14 29 13 37</td>
<td>10 14 13 29 37</td>
<td>10 13 14 29 37</td>
</tr>
<tr>
<td>10 29 14 37 13</td>
<td>10 14 29 13 37</td>
<td>10 14 13 29 37</td>
<td>10 13 14 29 37</td>
</tr>
<tr>
<td>10 14 29 37 13</td>
<td>10 14 13 29 37</td>
<td>10 14 13 29 37</td>
<td>10 13 14 29 37</td>
</tr>
<tr>
<td>10 14 29 13 37</td>
<td>10 14 13 29 37</td>
<td>10 14 13 29 37</td>
<td>10 13 14 29 37</td>
</tr>
</tbody>
</table>

Analysis: BubbleSort performs n-1 passes, where n is the length of the array. During its first pass the algorithm performs n-1 comparisons and at most (n-1) exchanges (swaps), during the second pass - n-2 comparisons and at most that many exchanges, etc.

Running time -- worst case: (n-1) + (n-2) + ... + 1 = n*(n-1)/2 = O(n^2)

best case: same

= O(n)