Pseudocode for 3 Elementary Sort Algorithms

If we want to sort an array, we have a wide variety of algorithms we can use to do the job. Three of the simplest algorithms are Selection Sort, Insertion Sort and Bubble Sort. None of these is especially efficient, but they are relatively easy to understand and to use.

In each of the three methods, we traverse all or part of an array repeatedly. There is a sorted part, which starts out empty and keeps growing, and there is an unsorted part, which initially is the whole array and keeps shrinking. We continue until we are done.

Selection Sort

Suppose A is an array of N values. We want to sort A in ascending order. That is, A[1] should be the smallest and A[N] should be the largest.

The idea of Selection Sort is that we repeatedly find the smallest element in the unsorted part of the array and swap it with the first element in the unsorted part of the array.

```
For I = 1 to N-1 do:
  Smallsub = I
  For J = I + 1 to N-1 do:
    If A(J) < A(Smallsub)
      Smallsub = J
    End-If
  End-For
  Temp = A(I)
  A(I) = A(Smallsub)
  A(Smallsub) = Temp
End-For
```

A refinement of the above pseudocode would be to avoid swapping an element with itself.

An alternate way to sort in ascending order is to find the largest value and swap with the last element in the unsorted part of the array.

Selection Sort does roughly N**2 / 2 comparisons and does N swaps.

Insertion Sort

Suppose A is an array of N values. We want to sort A in ascending order.

Insertion Sort is an algorithm to do this as follows: We traverse the array and insert each element into the sorted part of the list where it belongs. This usually involves pushing down the larger elements in the sorted part.

```
For I = 2 to N
  J = I
  Do while (J > 1) and (A(J) < A(J - 1)
    Temp = A(J)
    A(J) = A(J - 1)
    A(J - 1) = Temp
    J = J - 1
  End-Do
End-For
```

Insertion Sort does roughly N**2 / 2 comparisons and does up to N - 1 swaps.

Bubble Sort

Suppose A is an array of N values. We want to sort A in ascending order.

Bubble Sort is a simple-minded algorithm based on the idea that we look at the list, and wherever we find two consecutive elements out of order, we swap them. We do this as follows: We repeatedly traverse the unsorted part of the array, comparing consecutive elements, and we interchange them when they are out of order. The name of the algorithm refers to the fact that the largest element "sinks" to the bottom and the smaller elements "float" to the top.

```
For I = 1 to N - 1
  For J = 1 to N - 1
    If (A(J) > A(J + 1)
      Temp = A(J)
      A(J) = A(J + 1)
      A(J + 1) = Temp
    End-If
  End-For
End-For
```

Bubble Sort does roughly N**2 / 2 comparisons and does up to N**2 / 2 swaps.

Notes

To sort in descending order instead, we would use > instead of < when we compare array elements.

It may be possible to improve the above algorithms slightly.

None of these is an especially efficient sorting algorithm. Each of them requires a number of steps proportional to N**2. They vary in their efficiency, depending on whether the array is filled with random values or is already partly sorted or is already sorted in one direction or the other.

Bubble Sort is the best to use for an array which is already mostly in order, but it is the slowest to use for an array initially filled with random values. Insertion Sort is very fast if the array is already in ascending order. Selection Sort (unless we avoid self-swapping) is always about equally fast or slow.