

# Quadratic Sorting algorithms

Computer Science AP-A

Assume all functions are sorting a record whose field identifier 'key' is sorted from least to greatest. Whenever the variable ' $n$ ' is used it is assumed to be the size of the array, vector, or file.

```
struct Item
{
    int key;
    data : // struct, this is the weight
}

apvector<Item> a(n); // n is size of array
```

Order is the amount of work the algorithm does to sort the data in terms of  $n$ . This is also called the Big-Oh.

The *order* of all Straight sorts (Quadratic) is  $O(n^2)$ .

*Min*, *Avg* and *Max* refer to the *state* of the array before we begin sorting the data. *Min* indicates the data is already ordered, *Avg* indicates the *data* is in random order and *Max* indicates the data is in inverted order.

The *weight* of a record is the amount data that is stored in the record. The heavier the record, the longer each move will take.

The three factors to consider when choosing a sort are: Size the of the data to sort, the order of the existing data, and the weight of the of Items.

## Straight Insertion:

```
void StraightInsertion(apvector<Item> & a)
{
    for (int i = 1; i < n; i++)
    {
        Item newItem = a[i];
        int insertPos = i;
        while (insertPos != 0 && newItem.key < a[insertPos-1].key)
        {
            a[insertPos] = a[insertPos-1];
            insertPos--;
        }
        a[insertPos] = newItem;
    }
}
```

### Object of Insertion Sort:

1. Starting at the  $A_i$  element compare it with  $A_{i-1}$ ,  $A_{i-2}$ , ... elements until you find  $A_k$  such that  $A_k < A_i < A_{k+1}$  (a linear search with a key). That is, you take the first element of the unsorted field and place it into the correct position of the sorted field.

### Example:

	44 55 12 42 94 18 06 67
i=1	44 55 12 42 94 18 06 67
i=2	12 44 55 42 94 18 06 67
i=3	12 42 44 55 94 18 06 67
i=4	12 42 44 55 94 18 06 67
i=5	12 18 42 44 55 94 06 67
i=6	06 12 18 42 44 55 94 67
i=7	06 12 18 42 44 55 67 94

The number of comparisons in the  $i$ th shift is at most  $i-1$  (at least 1), where  $\frac{i}{2}$  is the average.

The total number of comparisons and moves are:

$$C_{\min} = n - 1$$

$$M_{\min} = 2(n - 1)$$

$$C_{\text{ave}} = \frac{1}{4}(n^2 + n - 2)$$

$$M_{\text{ave}} = \frac{1}{4}(n^2 + 9n - 10)$$

$$C_{\max} = \frac{1}{2}(n^2 - n)$$

$$M_{\max} = \frac{1}{2}(n^2 + 3n - 4)$$

The least number of moves occurs if the Items are originally in order; the worst case occurs if the Items are originally in reverse order.

## Straight Selection:

```
void SelectionSort(apvector<Item> & a)
{
    int j, k, minIndex;

    for (k = 0; k < a.length() - 1; k++)
    {
        minIndex = k;          // smallest Item from k to the end of a
        for (j = k+1; j < a.length(); j++)
        {
            if (a[j].key < a[minIndex].key)
                minIndex = j;    // new smallest Item
        }
        Item temp = a[k];      // swap smallest Item and kth Item
        a[k] = a[minIndex];
        a[minIndex] = temp;
    }
}
```

Method:

1. Select the Item with the smallest key from the unsorted field (a[k] to a[last]).
2. Exchange it with the first Item of the unsorted field a[k].

Example:

```
44 55 12 42 94 18 06 67
06 55 12 42 94 18 44 67
06 12 55 42 94 18 44 67
06 12 18 42 94 55 44 67
06 12 18 42 94 55 44 67
06 12 18 42 44 55 94 67
06 12 18 42 44 55 94 67
06 12 18 42 44 55 67 94
```

Comparisons =        All cases =  $\frac{1}{2}(n^2 - n)$   
Moves                All cases =  $3(n-1)$

## Straight Exchange (Bubble Sort):

```
void BubbleSort(apvector<Item> & a)
{
    for (int i = a.length()-1; i > 0; i--)
        for (int pos = 0; pos < i; pos++)
            if (a[pos].key > a[pos+1].key)
                {
                    Item temp = a[pos];
                    a[pos] = a[pos+1];
                    a[pos+1] = temp;
                }
}
```

Method:

1. Look at neighboring elements in the unsorted field (a[0] up to a[i]), switching if out of order.
2. After the first pass the largest element is moved into the last location.
3. Repeat step 1 until there is only one element left in the unsorted field.

Example:

```
44 55 12 42 94 18 06 67
44 12 42 55 18 06 67 94
12 42 44 18 06 55 67 94
12 42 18 06 44 55 67 94
12 18 06 42 44 55 67 94
12 06 18 42 44 55 67 94
06 12 18 42 44 55 67 94
06 12 18 42 44 55 67 94
```

Notice that the last three inner loops do nothing!

$$\text{Comparisons} = \frac{1}{2}(n^2 - n)$$

$$\text{Mmin} = 0$$

$$\text{Mave} = \frac{3}{4}(n^2 - n)$$

$$\text{Mmax} = \frac{3}{2}(n^2 - n)$$

Improvement can be made by checking to see if there was an exchange or not in the inner loop (Bubble sort with a flag). If no swaps were made then the sort algorithm terminates. Which, if any, of the comparisons and moves will be different?