Selection Sort

Selection Sort

- Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted. So with n elements,n-1 passes, one element will be left which will already be at its sorted position.

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
- 1) The subarray which is already sorted.
  2) Remaining subarray which is unsorted.

- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.
- It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.
- Selection sort is an inplace sorting(no extra memory is required) but not a stable sorting(as relative order of elements is not maintained at each swap).

Eg: Given an array A contains 8 elements as follows: 77,33,44,11,88,22,66,55

Suppose an array A contains 8 elements as follows:

77, 33, 44, 11, 88, 22, 66, 55

Pass	A[l]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = l, LOC = 4	**77**	33	44	**11**	88	22	66	55
K = 2, LOC = 6	11	**33**	44	77	88	**22**	66	55
K = 3, LOC = 6	11	22	**44**	77	88	**33**	66	55
K = 4, LOC = 6	11	22	33	**77**	88	**44**	66	55
K = 5,LOC = 8	11	22	33	44	**88**	77	66	**55**
K = 6, LCjC = 7	11	22	33	44	55	**77**	**66**	88
K = 7, LOC = 7	11	22	33	44	55	66	**77**	88
Sorted:	11	22	33	44	55	66	77	88

Algorithm

- There remains the problem of finding ,during the Kth pass, the location LOC of the smallest among the elements A[K], A[K+1], ...A[N]
- This may be accomplished by using a variable MIN to hold the current value while scanning the subarray from A[K] to A[N].
- Specifically, first set MIN=A[K] and LOC=K, and then traverse the list, comparing MIN with each other element A[J] as follows:

 If MIN<=A[J] , then simply move to the next element.

 If MIN>A[J], then update MIN and LOC by setting MIN=A[J] and LOC=J.

Procedure: MIN(A,K,N,LOC)

- An array A is in the memory. This procedure finds the location LOC of the smallest element among A[K],A[K+1],.....A[N].

```
1.  Set MIN=A[K] and LOC=K.
2.  Repeat for J=K+1,K+2,.....,N
      If MIN>A[J],then: Set MIN= A[J] and
    LOC=A[J] and LOC= J.
     [End of loop]
 3. Return
```

(Selection Sort)SELECTION (A,N)

- This algorithm sorts the array A with N elements.

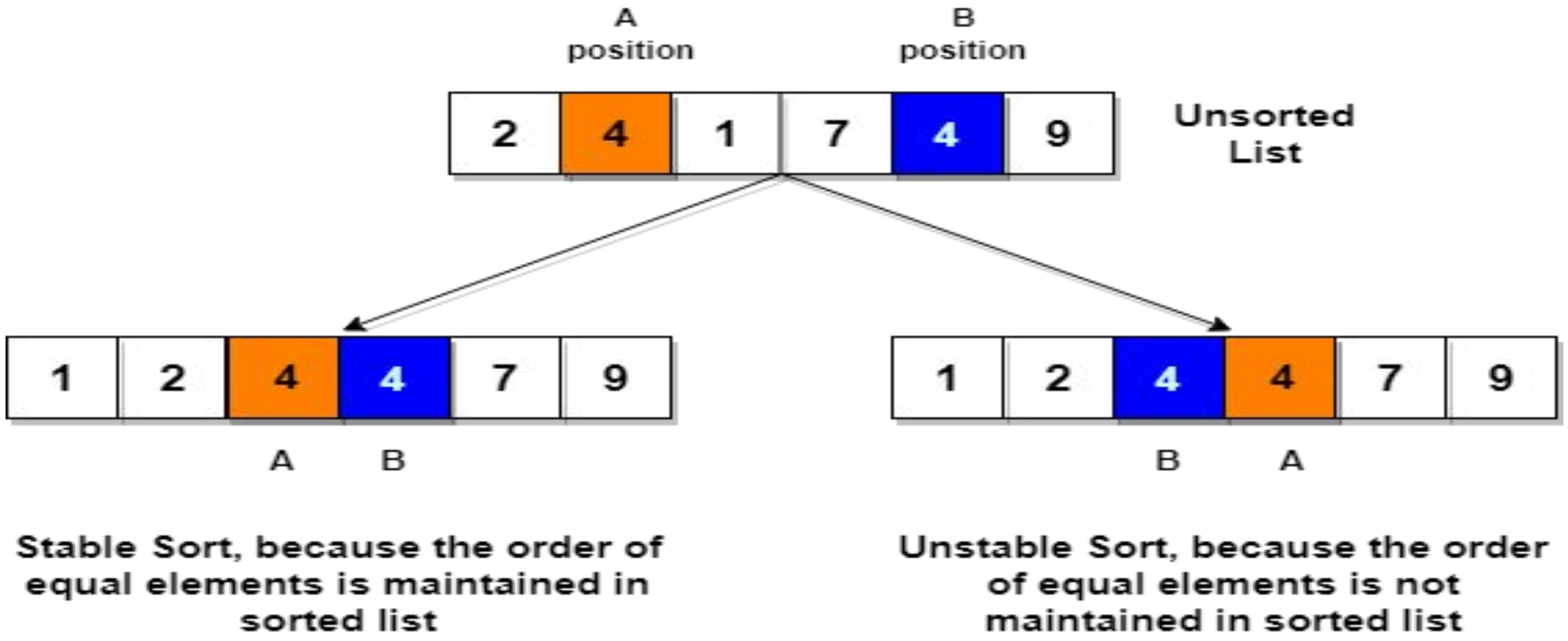
```
1.  Repeat Steps 2 and 3 for K=1,2,....,N-1
2.  Call MIN(A,K,N,LOC).
3.  [Interchange A[K] and A[LOC]]
      Set TEMP:= A[K],A[K]:=LOC and
    A[LOC]:=TEMP
    [End of Step 1 Loop]
4. Exit
```

Complexity Analysis of Selection Sort

- Worst case complexity:- When the list is in descending order and we have to arrange it in ascending order by placing smallest element at first position.
- Selecting the smallest element requires scanning all n elements. This takes (n-1) comparisons. Then swap it with the number at first position.
- Selecting the second smallest element requires scanning all n-1 elements. This takes (n-2) comparisons. Then swap it with the number at second position and so on till we reach the last element where we have to do 0 comparisons.
- $(n-1)+(n-2)+(n-3)+.....+2+1=n*(n-1)/2=O(n^2)$.

Insertion Sort

- It is efficient for smaller data sets, but very inefficient for larger lists.
- Insertion Sort is adaptive, that means it reduces its total number of steps if a partially sorted array is provided as input, making it efficient.
- It is better than Selection Sort and Bubble Sort algorithms.
- It is a **stable** sorting technique, as it does not change the relative order of elements which are equal.


A position
B position
2 4 1 7 4 9
Unsorted List
1 2 4 4 7 9
A B
1 2 4 4 7 9
B A
Stable Sort, because the order of equal elements is maintained in sorted list
Unstable Sort, because the order of equal elements is not maintained in sorted list

Given an array A contains 8 elements as follows: 44,77,22,88,33,66,11,55

Algorithm

Insertion_Sort (a, n)

```
1.   For(i=1;i<=(n-1);i++)
2.   {
3.   If(a[i]>a[i+1])
4.     {
5.       temp=a[i+1]
6.       loc=i+1
7.     while(loc !=1 && a[loc-1]>temp)  // free space is loc ,the data at loc-1 will
8.       {                                                    come down
9.         a[loc]=a[loc-1] // At free space previous data will come down
10.          loc=loc-1 // free space is updated
11.  }
12.      a[loc]=temp // Insert element at its proper position
13.  }
14.  }
15.  }
```

Complexity of Insertion Sort

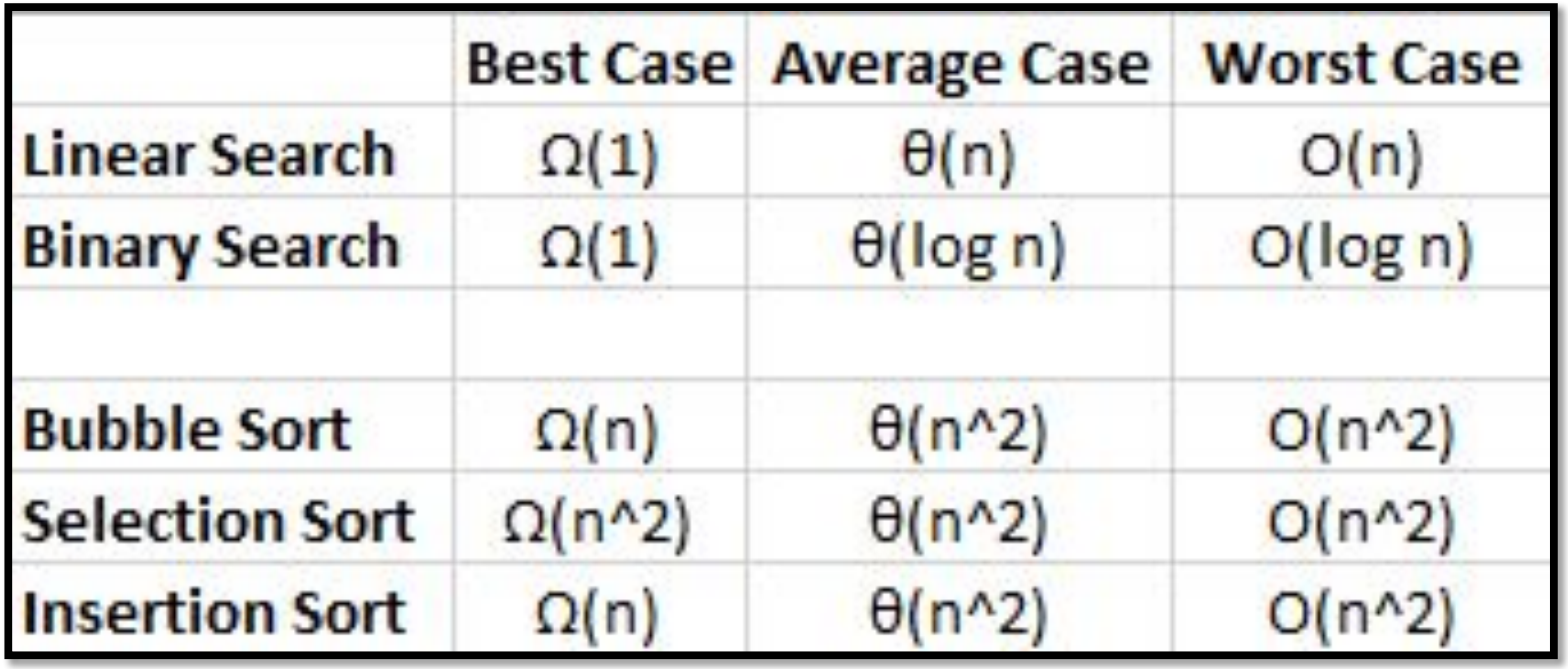
- Worst Case: In case of insertion sort worst case occurs when array A is in reverse order. And inner loop must use maximum number of (k-1) comparisons.

f(n)=1+2+3+........+(n-1)= n*(n-1)/2= O(n^2).

- Average Case: For average case, maximum of (k-1)/2 comparisons takes place ie

f(n)= 1/2+2/2+............+(n-1)/2= n*(n-1)/4= O(n^2).

Summary



	Best Case	Average Case	Worst Case
Linear Search	Ω(1)	θ(n)	O(n)
Binary Search	Ω(1)	θ(log n)	O(log n)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)