**Selection Sort**

In selection sort the list of the numbers is divided into two sub lists sorted and unsorted which are divided by an imaginary wall.

Once we find the smallest element in the unsorted list we swap it with the first element of the unsorted list of numbers.

Once the selection and swapping is done the imaginary wall moves ahead increasing the number of sorted elements and decreasing the number of unsorted elements.

A list of n elements requires n-1 passes to completely arrange the data.

The inner loop in a selection sort executes the size of the unsorted part minus 1 and in each iteration we make one key comparison. The no of key comparisons n\*(n-1)/2. Therefore the complexity of Selection sort is O(n^2).

The best case, worst case and the average case of selection sort are same. That is for each case in Selection Sort the complexity of Selection Sort is O(n^2).

We can see that in the graphs plotted for all cases the time required for selection sort to sort a given list of number increases in proportion with the increase in the number of numbers.

So we can conclude that selection sort is effective only when the number of elements in a list is low.

**Insertion Sort**

In Insertion Sort also the list is divided in two parts sorted and unsorted. In this in each pass, the first element of the unsorted part is picked up and transferred to the sorted sub list and inserted at the appropriate place.

A list of n elements will take at the most n-1 passes to sort the data.

Complexity of Insertion Sort

Best Case:

Array is already sorted in ascending order.

Therefore inner loop will not be executed.

The number of moves will be 2\*(n-1)

The number of key comparisons is (n-1)

Therefore the best case complexity of Insertion sort is O(n).

Worst Case:

Array is in reverse order.

Inner loop will be executed i-1 times, for i=1,2,3,…,n

The number of moves will be 2\*(n-1)+ (1+2+3+…+n-1)

The number of key comparisons will be n\*(n-1)/2

Therefore the worst case scenario of Insertion Sort is O(n^2).

Average Case:

We have to look at all possible initial data organizations.

Complexity of average case will be O(n^2).

**Merge Sort**

Mergesort algorithm is one of the two important divide and conquer sorting algorithms.

Mergesort is a recursive algorithm in which it divides the list into halves and sorted each half separately and then merge the sorted halves into one sorted array.

Complexity of Merge Sort:

Best Case:

All the elements in the first array are smaller than all the elements in the second array.

The number of moves will be 2k+2k.

The number of key comparison will be k.

Therefore for mergsort in best case the complexity will be O(n log(n))

Worst Case:

The number of moves will be 2k+2k

And the number of key comparisons will be 2k-1

Worst Case Complexity for Merge Sort will be O(n log(n)).

Average Case:

We have to look at all possible initial data organizations.

Complexity of average case will be O(n log(n)).

**Bubble Sort**

Even in Bubble Sort the list of numbers is divided in two sub lists that is sorted and unsorted. The smallest element is bubbled from the unsorted list and moved to the sorted sub list.

Then the wall moves one element ahead, increasing the number of sorted elements and decreasing the number of unsorted elements.

Given a list of n elements, bubble sort requires up to n-1 passes to sort the data.

Complexity for Bubble Sort

Best case:

Array is already in ascending order.

The number of moves required will be 0 that is O(1)

The number of key Comparisons will be (n-1)

Therefore the complexity of Bubble Sort will be O(n)

Worst Case:

Array is in reverse order.

That’s why the outer loop will be executed n-1 times.

The number of moves will be 3\*(1+2+…+n-1) = 3\*n\*(n-1)/2

The number of key comparisons will be (1+2+3+…+n-1) = n\*(n-1)/2

Therefore the complexity for worst case will be O(n^2).

Average Case:

We have to look at all possible initial data organizations.

Complexity of average case will be O(n^2).

|  |  |  |  |
| --- | --- | --- | --- |
| Sort /Case | Worst Case | Best Case | Average Case |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) |
| Insertion Sort | O(n^2) | O(n) | O(n^2) |
| Merge Sort | O(n log(n)) | O(n log(n)) | O(n log(n)) |
| Bubble Sort | O(n^2) | O(n) | O(n^2) |