

Q1 Time Complexity Analysis of Insertion Sort  
Ans In this sorting first we assume we have two part in our array sorted part and unsorted part. we will take element from unsorted part and put it into its correct position in sorted part.  
algorithm

```
for (i = 1; i <= n; i++):
```

```
    j = i - 1
```

```
    temp = a[j + 1]
```

```
    while (j >= 0 && a[j] > temp)
```

```
    {
```

```
        a[j + 1] = a[j]
```

```
        j = j - 1
```

```
    }
```

```
    a[j + 1] = temp
```

```
return a
```

Let us take example

let list  $[a_0 | a_1 | a_2 | a_3 | a_4]$

$[a_0 | a_1 | a_2 | a_3 | a_4 | - | - | a_n]$

sorted  
portion

unsorted  
portion

Step 1 compare  $a_0$  with sorted portion

$[a_0 | a_1 | a_2 | a_3 | a_4 | - | - | a_n]$

smaller  
of  $a_0$  &  $a_1$

larger of  $a_0$  &  $a_1$

unsorted  
list

Step 2 Now we will compare  $a_2$  with sorted

$[a_0 | a_1 | a_2 | a_3 | a_4 | - | - | a_n]$

sorted list      unsorted portion

so go on

✓



In best case there will be condition that whenever we compare element from unsorted position to the sorted position then element in unsorted position is always greater than all elements of sorted position it means list is ascending then every time there will be only 1 call

for  $n$  times =  $n-1$  call

Time complexity =  $O(n-1) = O(n)$

## 02 Bubble sorting Time Complexity

In this Method of sorting we start from 0th Index to last index if  $arr[j] > arr[j+1]$  then we will swap them 2nd time we start from 0th index to 2nd last index and do the same thing. we will keep doing it  $n-1$  time





$$\text{Total calls} = (n-1) + (n-2) + \dots + 1$$

By using A.P. sum formula.

$$\text{Total calls} = \frac{(n-1) \times n}{2}$$

$$\text{Time complexity} = O(n^2)$$

## Merge Sort

In this sorting we divide list into two half the again apply merge sort on two half after it we will merge the two sorted half

In Normal condition Merge sort is faster than selection, insertion & Bubble sort

Let us take example

$$[a_1, a_2, \dots, a_n] = [ \quad ] \quad [ \quad ]$$

we break it into 2 half

$$\text{Let total time} = T(n)$$

$$\text{then for each half time} = T\left(\frac{n}{2}\right)$$

$$T\left(\frac{n}{2}\right)$$

$$T\left(\frac{n}{2}\right)$$

(2)

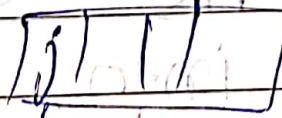
apply Merge  
Sort

apply Merge  
Sort

Now we have applied Merge  
Sort on both halves

2x 2x

(3)

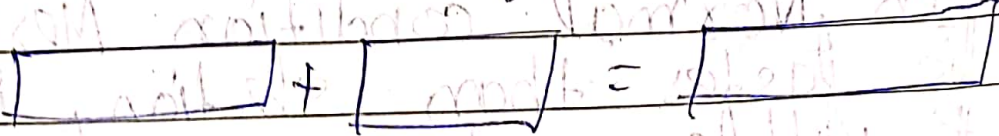


compare

(takes  $K_1 N$  time)

Now we will compare  
the two half

(4)



Now we will Merge them  
into 1

It will take  $K_2 N$  time

Here Recurrence Relation is



$$T(n) = 2T\left(\frac{n}{2}\right) + K_1 N + K_2 N$$

Let  $K_1 + K_2 = K$ .

$$T(n) = 2T\left(\frac{n}{2}\right) + Kn$$

$$2 \times \left[ T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + K\frac{n}{2} \right]$$

$$4 \times \left[ T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + K\frac{n}{4} \right]$$

⋮

$$T(1) = K$$

Now we will add them.

$$T(n) = \underbrace{Kn + Kn + Kn + \dots}_{\log n \text{ times.}}$$

$$T(n) = Kn \log n$$

$$T(n) = O(n \log n)$$