* Insertion sort

```
for(i=2; i<n-1; i++)
{
    curr = arr[i];
    j = i-1;
    while(arr[j] > curr && j>=0)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = curr;
}
```

Insertion time complexity

Best = O(n)

avg ~~worst~~ = O(n²)

Worst = O(n²)

* Merge sort ← time complexity in all cases nlogn

```
Merge_sort(A,p,r)
If p<r then
    q = (p+r)/2;
    Merge_sort(A,p,q,r A,p,q)
    Merge_sort(A,q+1,r)
    Merge(A,p,q,r)
```

Merge (A, p, q, r)

Step 1 :- [Initialization]

M [r-p+1]

I1 <— p

I2 <— q+1

X <— 0

step 2 :- Repeate step 3 while i1<=q && i2 <= r

step 3 :-

If A[i1] ≤ A[i2] Then

M[x] <— A[i1];

x <— x+1

else

M[x] <— A[i2];

x <— x+1;

step 4 :- Repeate step 5 while i1 ≤ q

step 5 :- merged[x] <— A(i1)

step 6 :- Repeate step 7 while i2 ≤ r

step 7 :- m[x] <— A[i2]

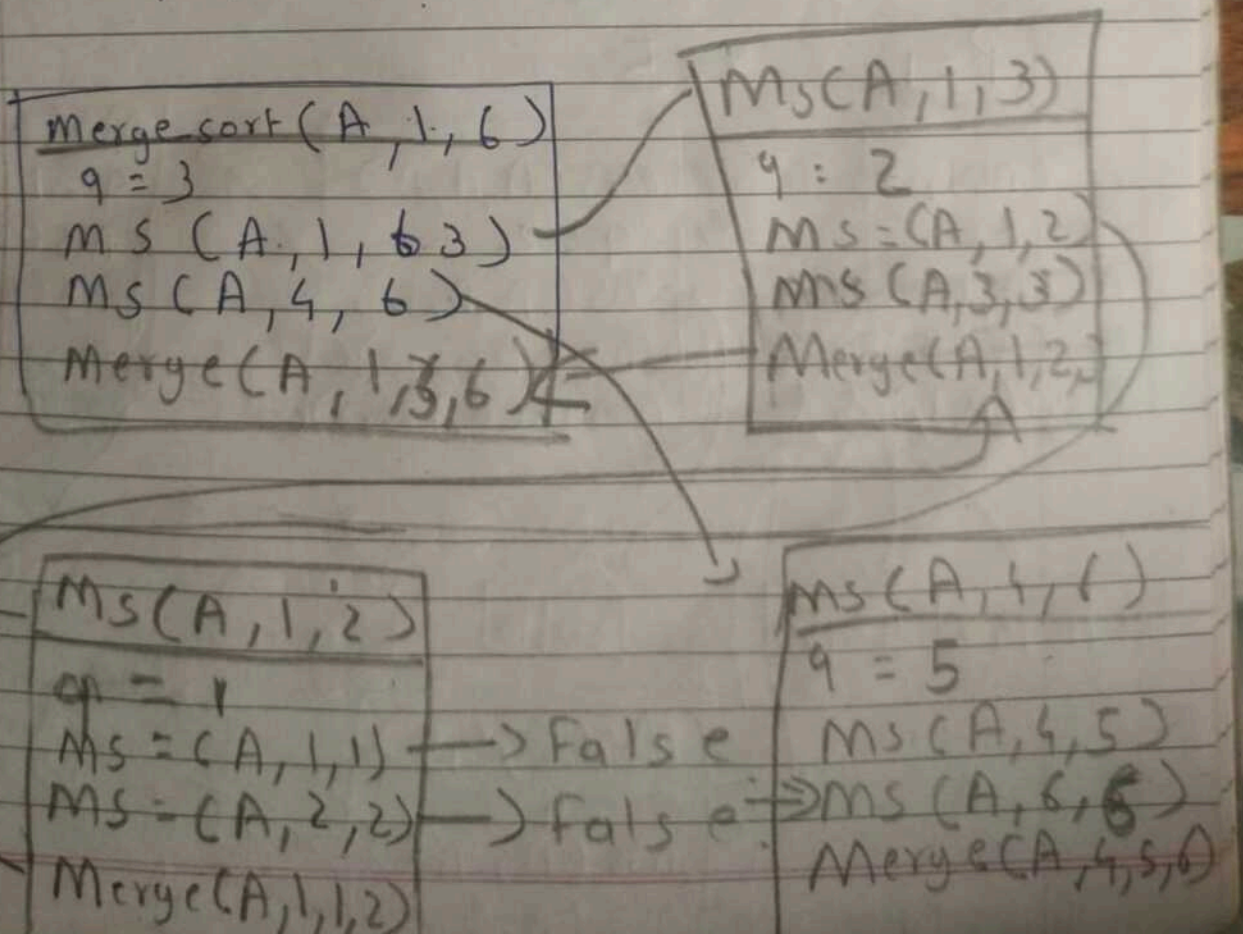
Step 8:- [copy element to original array]

Repeat ~~for~~ while i<-1 ; j <- p to r do

A[j] = Merged[i];
j <- i + 1 ;
j <- j + 1 ;

Tracing

2, 5, 7, 3, 9, 8



MS(A,4,5)
q = 4
MS(A,4,4) ← False
MS(A,5,5) ← False
Merge(A,4,4,5)

30, 10, 40, 60, 50
1 2 3 4 5

P=1 , r=5

MS(A,1,5)
q = 3
MS(A,1,3)
MS(A,4,5)
Merge(A,1,3,5)

MS(A,1,3)
q = 2
MS(A,1,2)
MS(A,3,3) ← False
Merge(A,1,2,3)

MS(A,1,2)
q = 1
MS(A,1,1) → False
MS(A,2,2) → False
Merge(A,1,2)

30 | 10 | 40 | 60 | 50

30 | 10 | 40

30

MS(A,4,5)
q = 4
MS(A,4,4) → False
MS(A,5,5) → False

60

50

Merge(A,4,4,5)