```
{
    Swapping (array[start], array[i]);
    }
}
    Swapping (array[l], array[end]);
return end.
```

Analysis of the complexity of any list in insertion sort
for best case list should be in ascending order as we know algorithm of insertion sort

```
for (int n=1; n<n; n++)
    temp = arr[n]
    for (int y=n-1; y>=0; y--)
{
if (temp < arr[y])
    {
      temp [y+1] = temp[y]
     temp [y] = temp;
    }
    Else
    break;
    }
}
```

Consider list is {7, 9, 11, 13, 18}

Loop 1→ temp=9

Loop 2→ (9<7) Else

false break;

means at n=1 loop-2 has been called 1 time

similarly

(ii) for n=2

temp=11

1)

Loop 2 [y=1] ✓ y>=0 true

if (11 < arr[1] then break

false

means at n=2 Loop has been called once again

So we can say - [In ascending order]

Loop 1	Loop 2	no. of call
n=1	y=0	1
n=2	y=1	1
n=3	y=2	1
n=4	y=3	1
n=5	y=4	1
		1
n=n-1	y=n-2	

Total loop call = n-1

Time Complexity = O(n-1) ≈ O(n)

Ques (2)

Bubble Sort

Algorithm

```
for (int a=0; a<n-1; a++)
{
  for (int b=0 b<n; a++)
  {
    if (a[b] > a[b+1])
    {
      temp = a[b];
```

```
        a[b] = a[b+1];
        a[b+1] = temp;
      }
    }
  }
```

Time complexity :→

$T(n) = O(n-1) \times O(n-1)$

$= [O(n-1)]^2 \simeq O(n^2)$

Loop 1

Loop-2 operates $(n-1)$ times

Loop-2

1 Loop-2 operates $(n-1)$ times

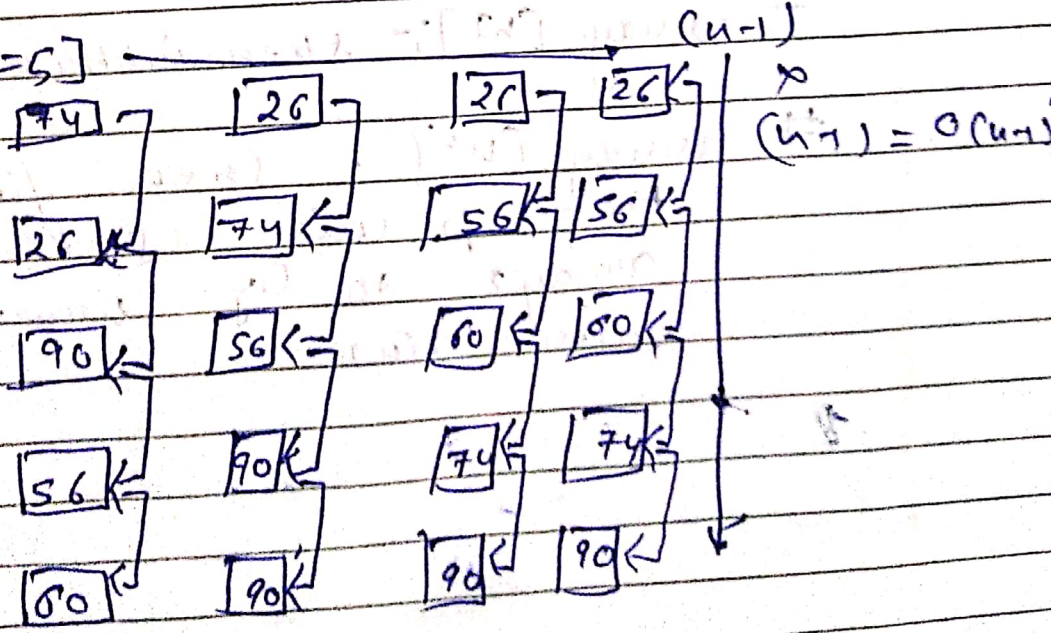
means that Loop 1 operates $(n-1)$

——— loop 2 — — — $(n-1)$

ne $T(n) = O(n-1)^2$

$\simeq O(n^2)$

for [n=5]



Space Complexity $O(1) =$ const

time Complexity at worst case = O(n²)
best case = O(n)

Space complexity = O(1)

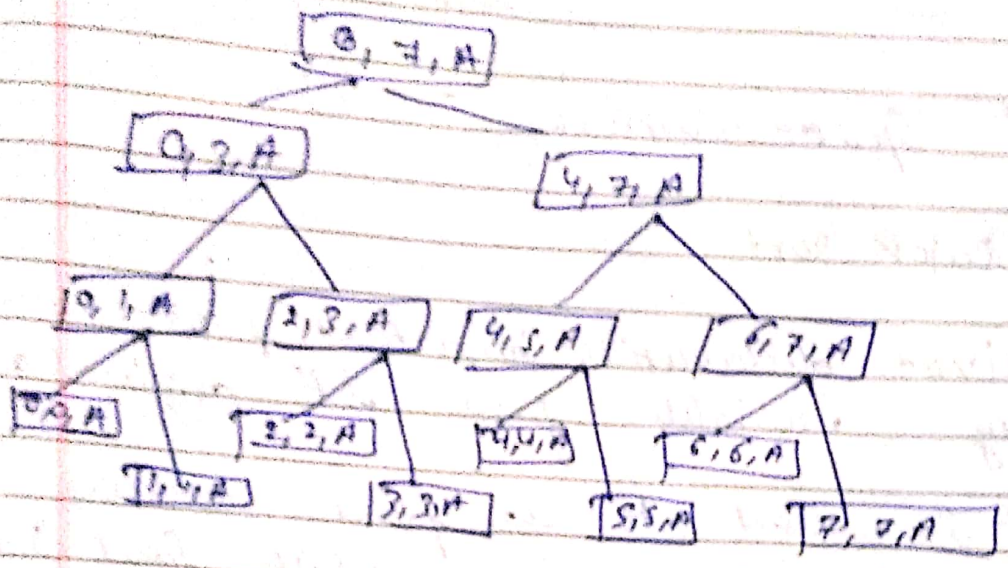
Quick Sort

Pivot declaration important pivot may
any variable from its original

It is used to a part array into two array in which first sub array of contain all values than pivot & other sub array will contain all values greater than pivot

```
partition (a array, l=0, u=n)
{
    int start = l
    int End = u
    pivot = a[l]
    while (start < end)
    {
        while (a[start] <= pivot && (
        {
            Start ++;
        }
        while (a[end] > pivot) && end >=
        {
            End a--;
        }
        if (Start < end)
```

Let's consider 8 element



No. of level $O(n) = \log_2(n) = \log_2(8) = 3$

Time complexity $[T(n) \rightarrow n \log n]$

Merge sort space complexity $= O(n)$

→ Insertion sort Algorithm

```
for (int a=1 ; a<n ; a++)
{
for (int b = a-1 ; b>=0 ; b--)
{
    If (array[b] > array[b+1])
    {
        temp = array[b];
        array[b] = [b+1]
        array[b+1] = temp;
    }
    else
    break;
```

Merge Sort Algorithm

```
Void mergesort (int l, int r, int *a)
  {
    if (l<r)
      {
        int m = (l+r)/2;

        mergesort (l, m, a);
        mergesort (m+1, r, a);
        merge (l, m, r, a);
      }
  }
void merge (int l, int m, int r, int *p)
  { int n2 = m-l+1;
    int n2 = r-m;
```

array [n1]; stores initial Data

array [n2]; stores final from other than p will store in array's array2 to by wanting with O(n).