DS Assignment

(1)

Time complexity Analysis → Insertion Sort.

* For best case: list should be in Ascending order

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

Example → { 1, 2, 3, 4, 5 }

① x=1

temp=2

→ y=0

arr 2<1 ✗ false

↳ break

Similarly all conditions will workout false & break;

Loop1	Loop2	Call
x=1	y=0	1
2	1	1
3	2	1
4	3	1
5	4	1
		1

Total calls: n-1 → Time complexity = O(n-1) ≈ O(n)

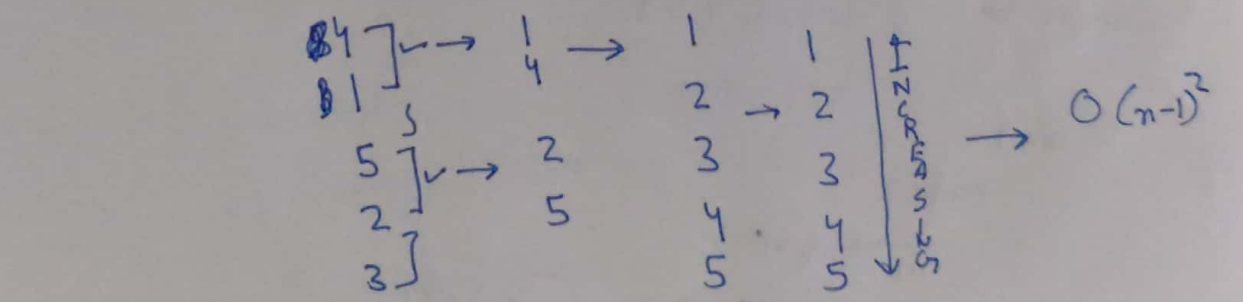
(2) Bubble Sort →

```
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;
      }
   }
}
```

① (n-1) times

② (n-1) times. } → Time complexity = $O(n^2)$
Space complexity $O(1)$ = (const.)

Eg → For n=5 →



* Merge Sort

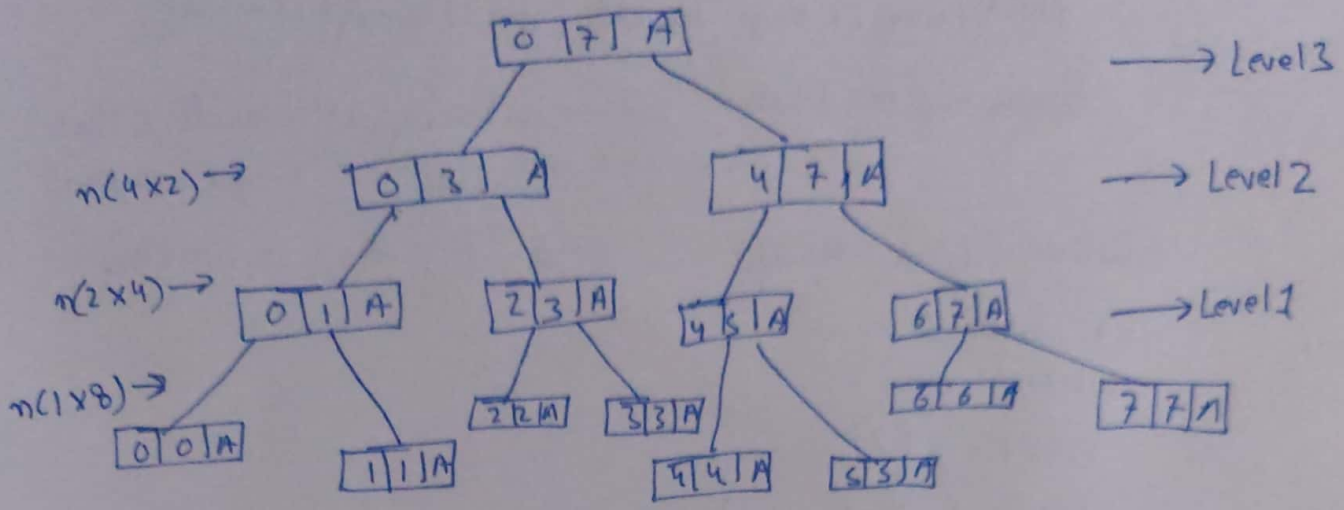
```
Void ms(int l, int r, int *a)
 If (l < r)
  { int m = (l+r)/2;
    ms(l, m, a);
    ms(m+1, r, a);
    m(l, m, r, a);
  }
```

→

```
Void m(int l, int m, int r, int *p)
{ int n1 = m-l+1;
  int n2 = r-m;
  { array[n1];
    array2[n2];
  }
```

Scanned with CamScanner

Taking an Example →



Levels = 3 ($\log_2(8)$)

Time Complexity [T(n)] → n log n

Merge Sort → O(n)
space complexity

* 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] = arr[b+1]
      array[b+1] = temp;
    }
  else
    break;
  }
```

Time Complexity Worst case → O(n²)
Best case → O(n)

☆ Quick Sort → Pivat declaration is imp.
Pivat may be any variable from its original array.

2 Subarrays are created 1st → containing all elements < Pivot
2nd → . . . . . > Pivot

```
Partition (l, m, array)      // (l = 0; m = n-1) initially
 int start = l
 int end = m
  Pivot = a[l]
  while (start < end)
     { while (a[start] <= pivat) && (start <= m)
       { start++;
       }
       while (a[end] > pivat)    && (end >= l)
        { end--;
        }
       if (start < end)
          { swaping (array[start] array[m])
          }
       }
       swapping (array[l] array[end])
      return end;
```