① Insertion Sort

```
Void insertionSort ( int A[] , int n)
{
    for ( int i = 1 ; i < n ; i++)        ⟶ O(n)
    {
        j = i - 1 ;
        Key = A[i]
        while ( j > -1 && A[j] > Key)     ⟶ O(n)
        {
            A[j+1] = A[j] ;
            j--
        }
        A[j+1] = Key ;
    }
}
```


★ Time Complex. (Avg. Case) = $O(n^2)$

★ Time Complex. for Best Case

Suppose the array is already sorted

for this array $j = i-1$, the condition $A[j] > A[i]$ will be false

∴ The Condition of while loop will never true

for n element, NO. of comparision = $O(n)$

NO. of swap (while loop) = $O(1)$

∴ Time Complex. of best case is $O(n)$

=> Binary search is used to reduce the no. of comparision in insertion sort. This modification is known as Binary Insertion sort.

=> Binary Insertion sort use binary serach to find the proper location to insert the selected item at each iteration. In insertion sort, it takes $O(i)$ at ith iteration, in worst case. Using binary search, it reduces to $O(\log i)$

Time Complexity

* Worst case - $O(n \log n)$
* Avg Case - $O(n \log n)$
* Best Case - $O(n)$

(2) Bubble sort

In this sorting tech. every element is compared with every other adjacent element. if it is in wrong order then it is swaped

Eg. | 5 | 3 | 2 |

Pass I Comp. element 5 with adj. element.

5 > 3 → swap

5 > 2 → swap

| 2 | 3 | 5 |

| 3 | 4 | 5 |

Pass II 3 > 4 → NO

4 > 5 → NO

⇒ Time Comp → $O(n^2)$

every elem. is Comp. to adjacent element for n times.

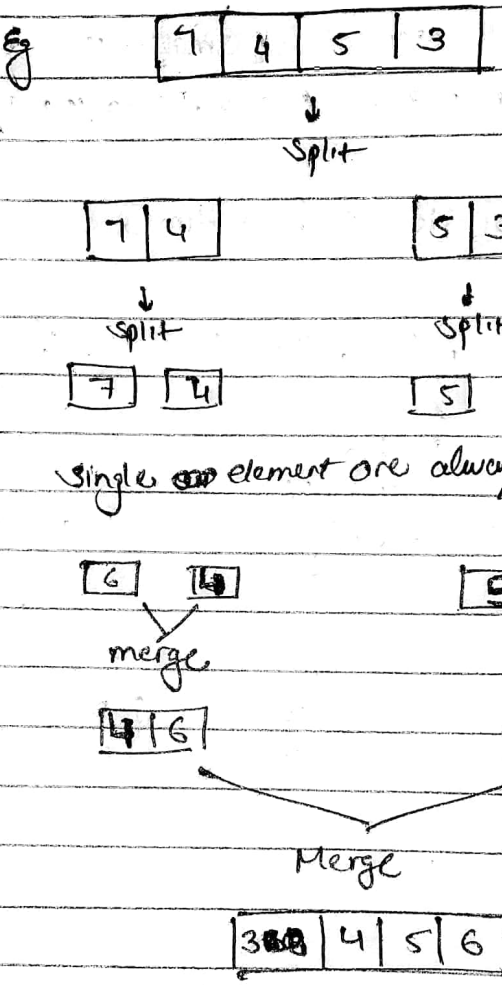
⇒ Space Comp → O(n)

Not required any extra space for sorting.

∴ It is In place algorithm

Merge sort

In this sort we take two list or array having sorted element and merge them accordingly

Eg 

Single element are always sorted

Time Comp → nlogn

Space Comp → Since merge sort require an extra array for sorting and the recursive merge sort is also requires a stack having height of logn

Total space comp = 2n + logn

Since an extra space is required.

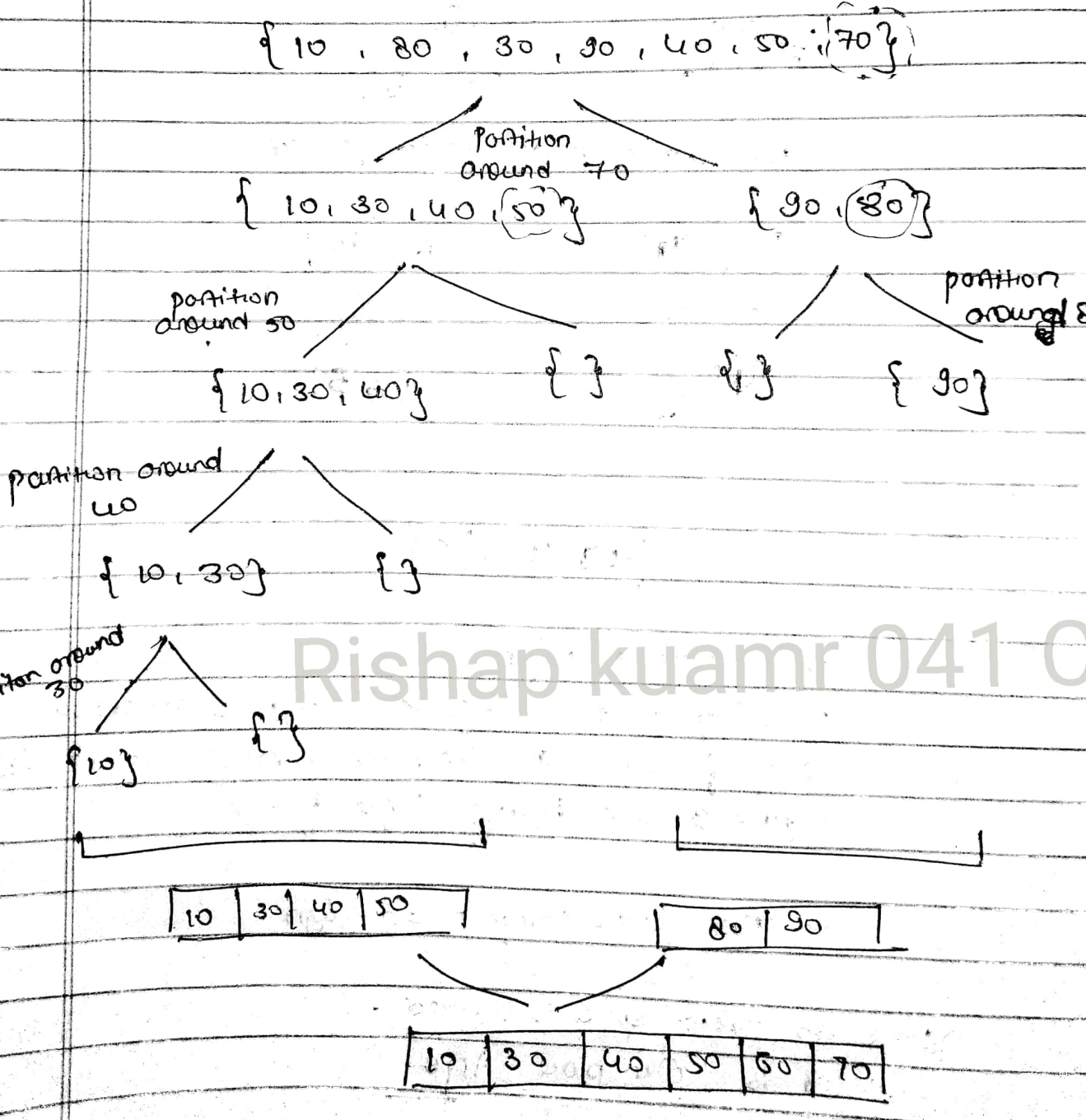
∴ It is Out place Algo.

Date:
Page:

Quick sort

The key process in quick sort is partition. Target of partition is, given an array and an element x of array as pivit, put x at its correct position in sorted array and put all smaller element (smaller than x) befor x. and put all grater element (greater than x) after x.

All this should be done in linear time.



∴ Time comp (Avg Case)
splitting the array in half and compare n elements
∴ (nlogn)

→ Although the avg time complex. is nlogn but time time complex. in worst case is $O(n^2)$

→ In worst case the partition of array takes place at end of array

∴ Time Complex = $O(n^2)$

(3)
- String is Converted into lower Case
- Comparasion is done on the bases of ASCII values
- Sort using merge SoA.