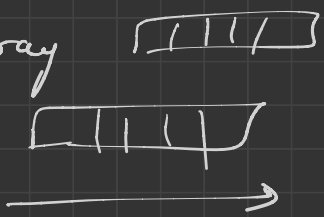
Recursion {Day-6}

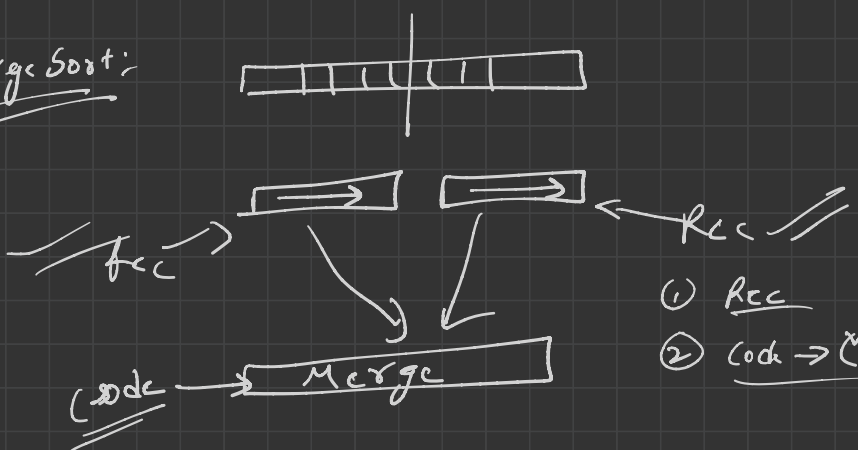
→ Quick Sort

i/p → array

o/p →



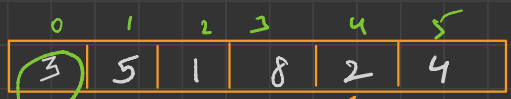
Merge Sort:



① Rec

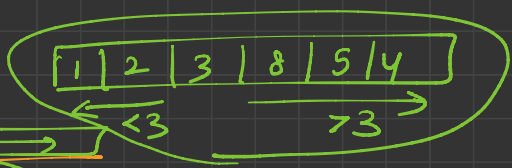
② Code → Merge

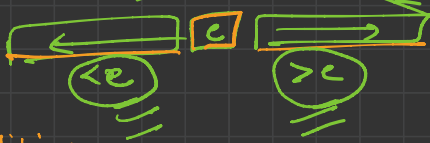
→ Quick sort



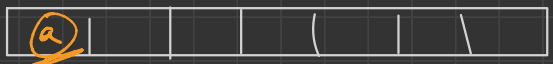
1st step

↳ element → Right place



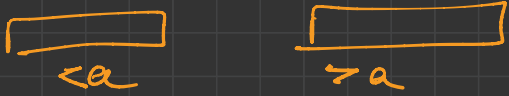


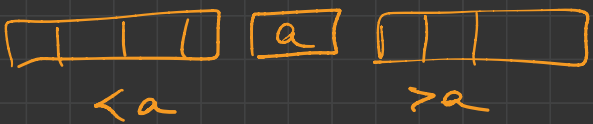
partition

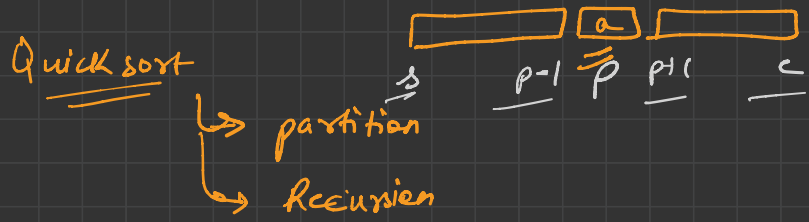


position pr daalo → right place

partition





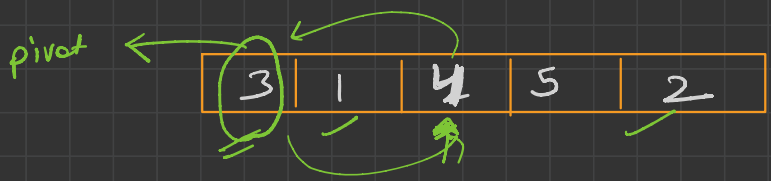



```
void quickSort (int arr[], int s, int e)
{
    // base case
    if (s >= e)
        return;

    //① partition
    p = partition (arr, s, e)

    // Rec C
    quickSort (arr, s, p-1);
    quickSort (arr, p+1, e);
}
```

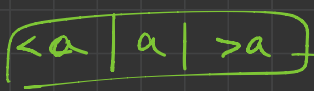
→ Partition



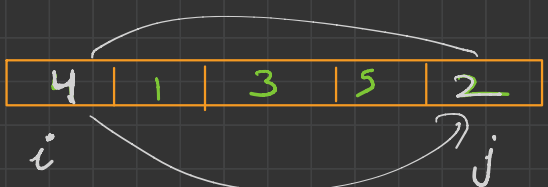
↳ take a pivot

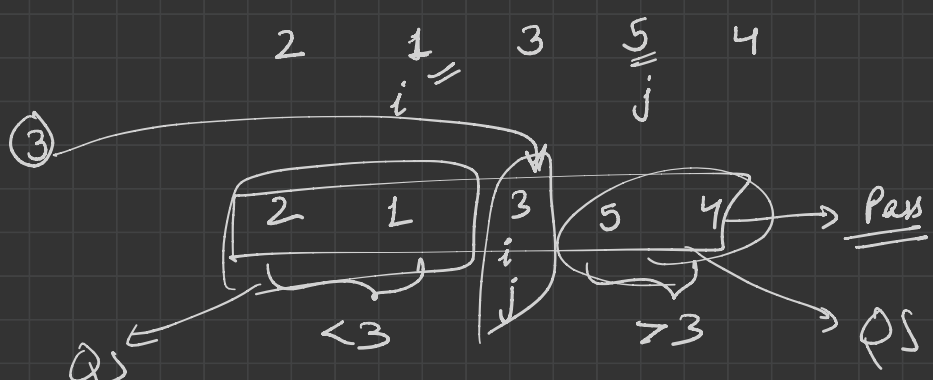
↳ count all element < pivot →

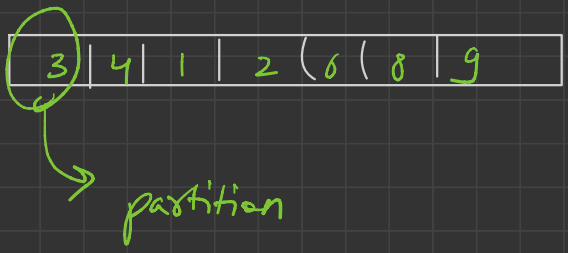
↳ pivot → s + count

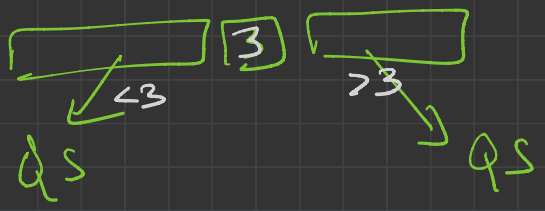
↳ 

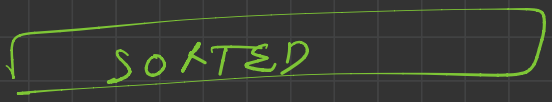
→ condn

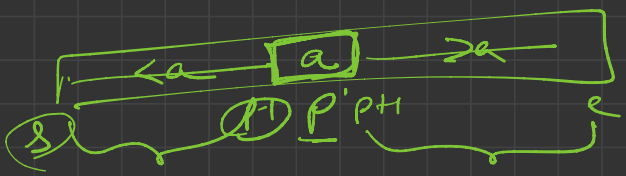


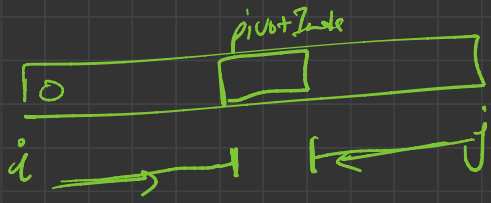


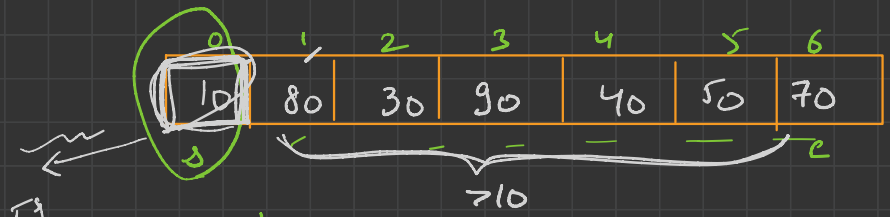

3
4
1
2
6
8
9
partition


3
<3
>3
QS
QS


SORTED


a
p-1
p
p+1
e


pivotIndex
0
i
j



pivot = 10

cnt = 0

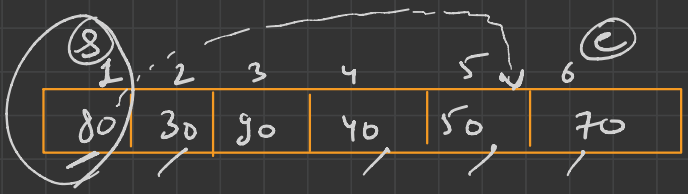
pivotIndex = s + cnt

= 0 + 0 = 0

i = 0 pi

0 < 0

return 0;



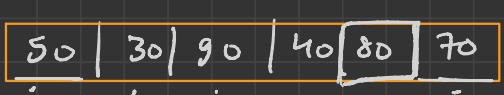
pivot = 80

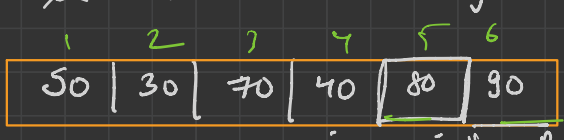
cnt = 4

pivotIndex = s + cnt

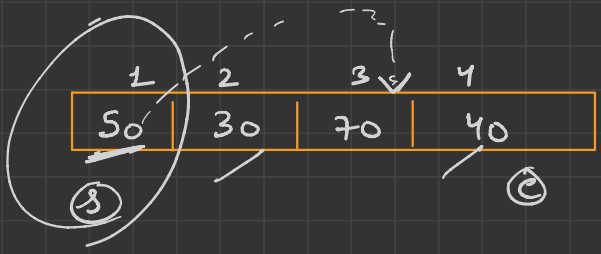
= 1 + 4 = 5

after swap →





left part →



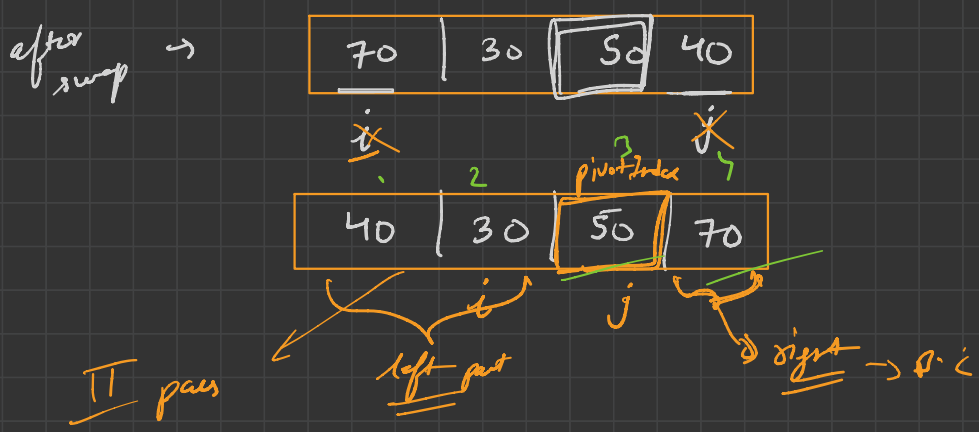
pivot = 50

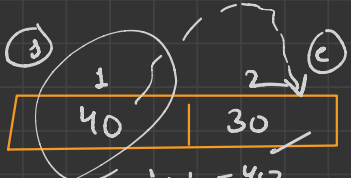
cnt = 0 → (2)

pivotIndex = s + cnt

= 1 + 2

= (3)

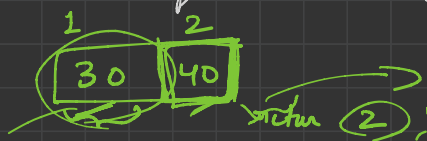


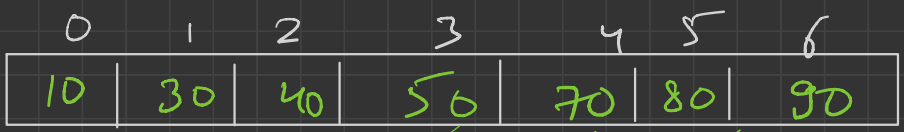


pivot = 40

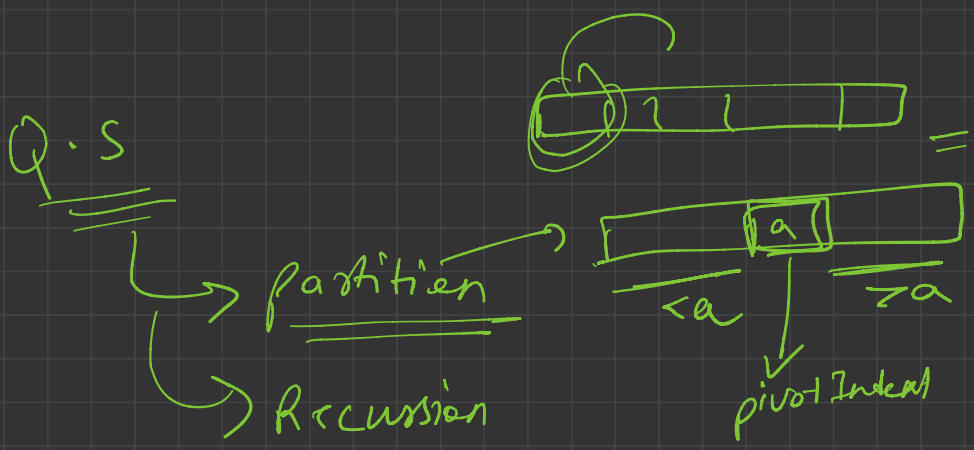
cnt = 1

pivotIndex = s + cnt = 1 + 1 = (2)





End result



→ H/w → in-place ? S.C → O(n)

→ stable → ?

T.C → O(nlogn)

Worst Case → $O(n^2)$

Q.S (T.C) S.C ↓ O(n)

Avg Case - $O(n \log n)$

Best Case - $O(n \log n)$

Worst Case - $O(n^2)$

n → size of array

HW

in-place → ✓ ✗

stable → ✓ ✗

KHUD code kro with DRY RUN