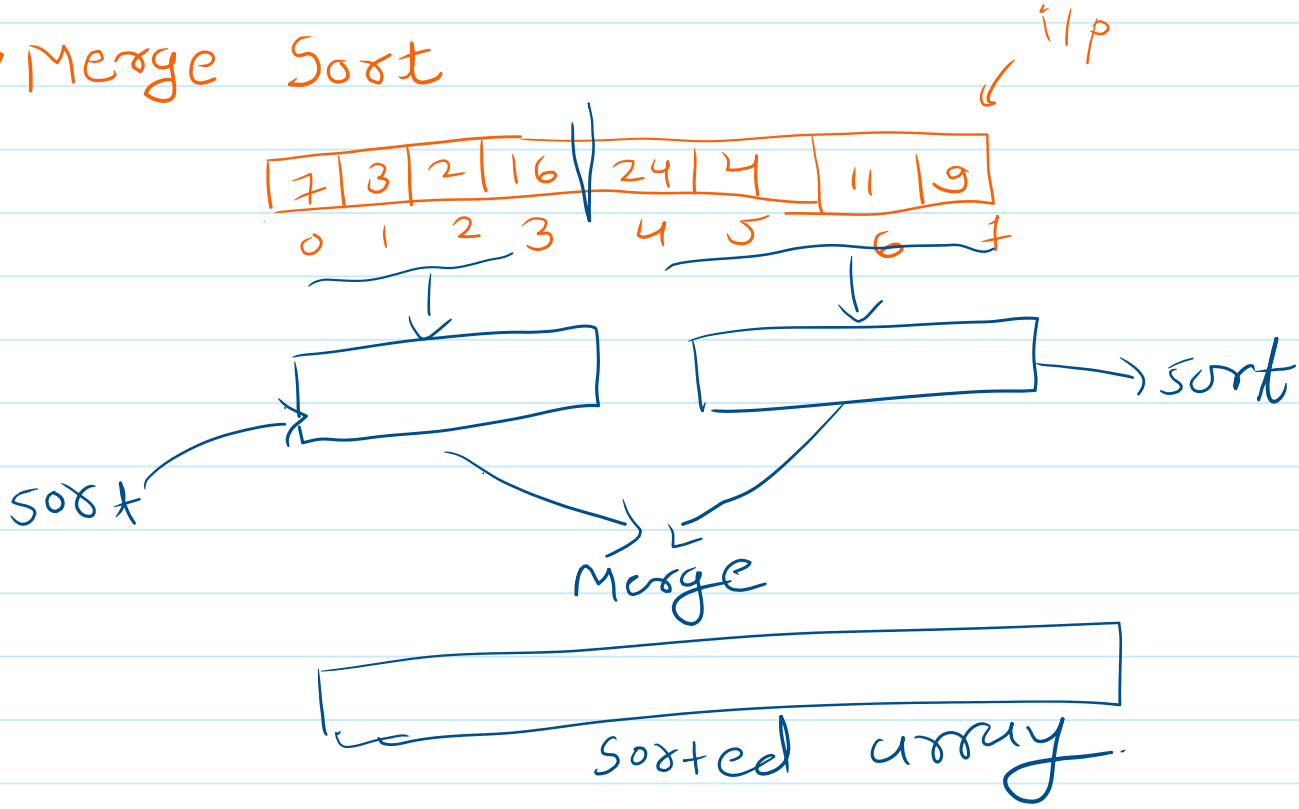
DnC

→ Merge Sort



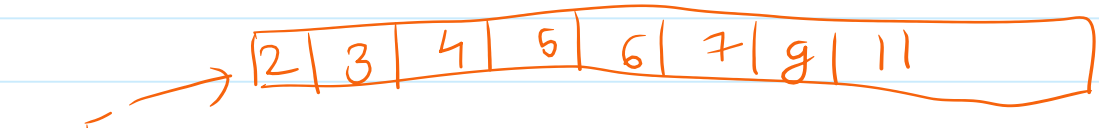
A mid find
B break 2 half
C Remaining 2half sort.
D 2 half → merge

10 April 2024 14:56

Merge 2 sorted arrays :- two pointer approach

ip → Array 1 → 2 | 4 | 6

Array 2 → 3 | 5 | 7 | 9 | 11



ans

array 1 end or array 2 end

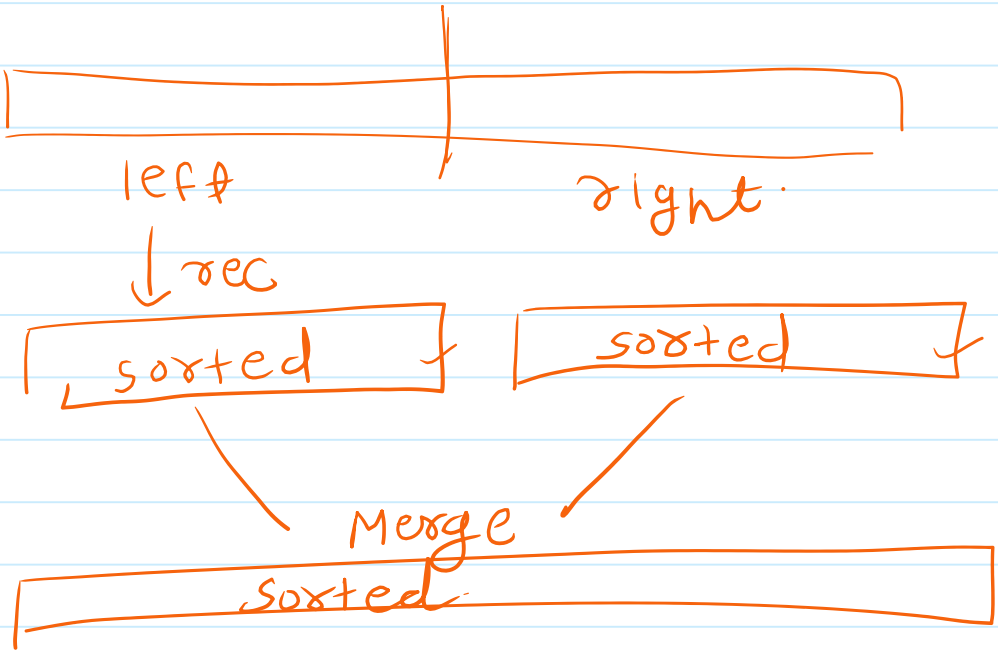
↳ while (i_{rj})

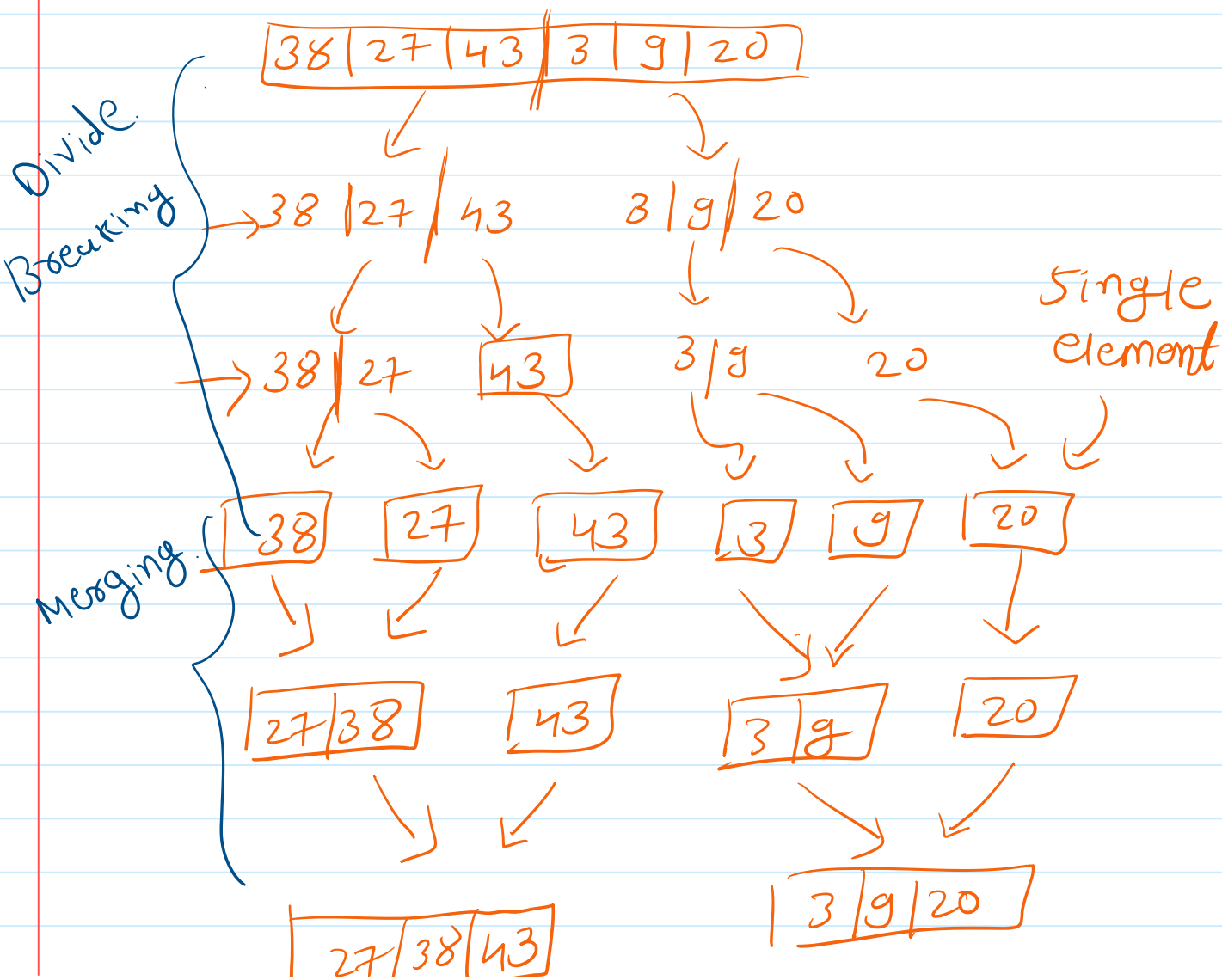
{

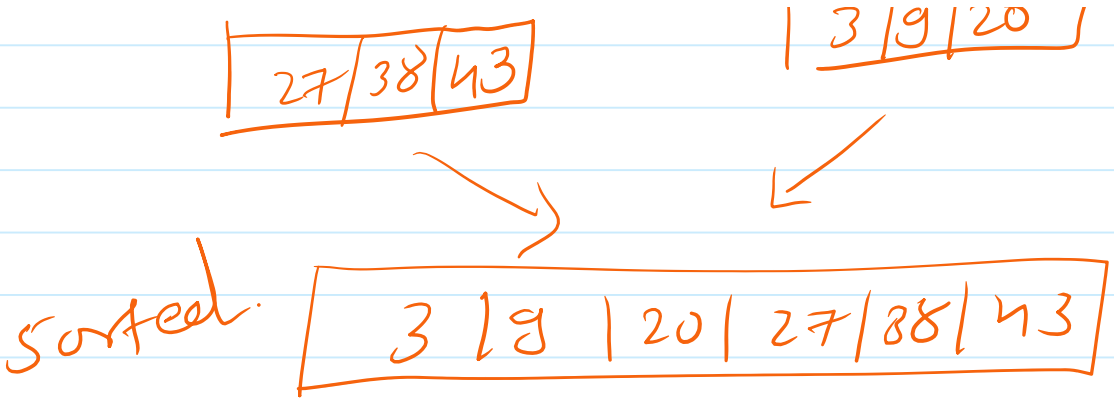
}

array 1 → remaining ele copy

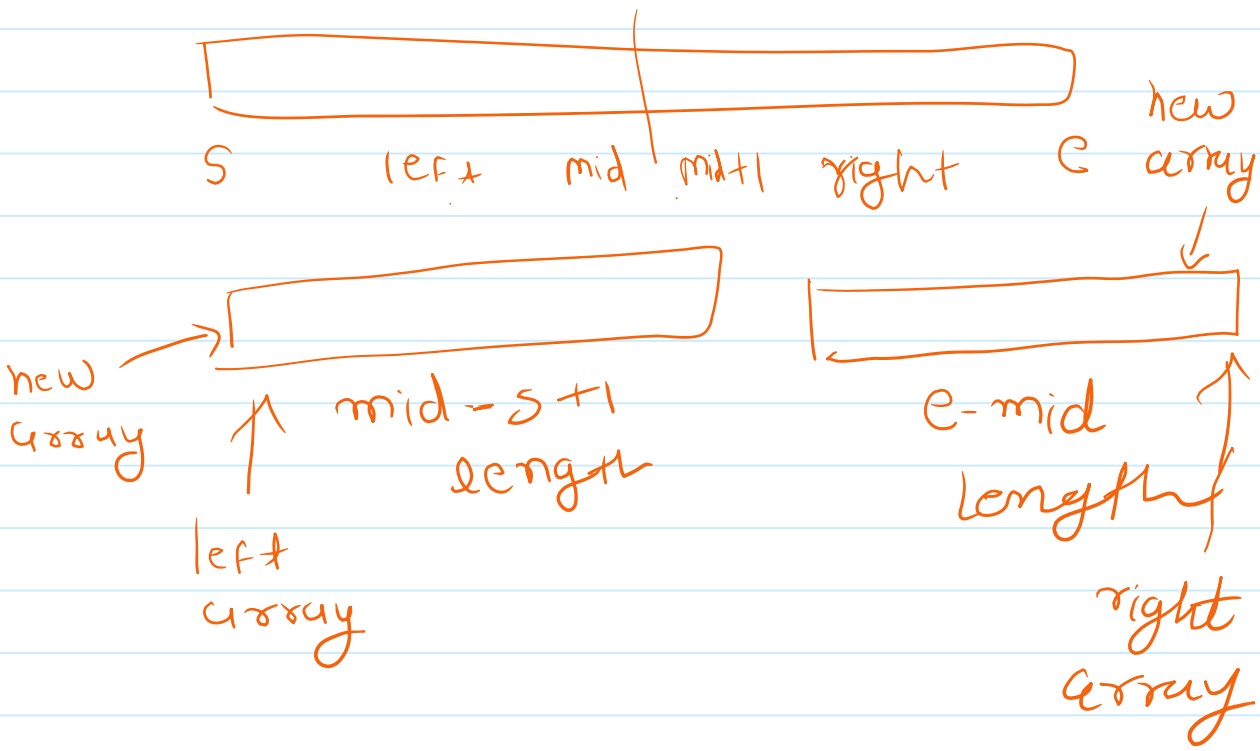
a2 → remaining ele copy


left
right
rec
sorted
sorted
Merge
sorted


38 27 43 3 9 20
Divide
Breaking
38 27 43
3 9 20
38 27
43
3 9
20
Single element
Merging
38
27
43
3
9
20
27 38
43
3 9
20
27 38 43
3 9 20


27 38 43
3 9 20
sorted.
3 9 20 27 38 43

10 April 2024 14:56



Merge Sort

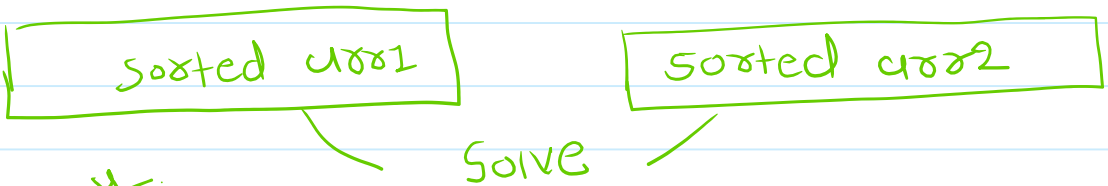
```
MS (arr, s, e)
{
    // BC

    int mid = (s+e)/2 <- (A)   Break

    // left

    ms(arr, s, mid) <- (B) left sort

    // right

    ms(arr, mid+1, e) <- (C) right sort
```

ms(arr, mid+1, e) ← (c) right sort

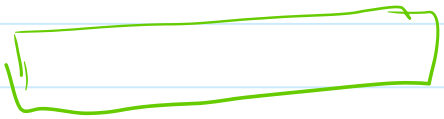
// Merge

merge(arr, s, e) ← (d) Merge both part.

}



new array



① length = mid - st + 1

② copy all value for s → mid.

① length = e - mid.

② copy all value for mid+1 → e

T.C = O(n log n)

S.C = O(left + right)

not S.C used → in-place merge sort

10 April 2024 14:56

```
Merge sort()
{
  // B.C
  // left call
  // right call
  // merge
}
```

left call, right call

$$T(n) = K_1 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n*K$$

$$= K_1 + 2T\left(\frac{n}{2}\right) + n*K$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n*K$$

$$2T\left(\frac{n}{2}\right) = 4T\left(\frac{n}{4}\right) + \frac{n}{2}*K \quad *2$$

$$4T\left(\frac{n}{4}\right) = 8T\left(\frac{n}{8}\right) + \frac{n}{4}*K \quad *4$$

$$\vdots \qquad \vdots$$

$$T(1) = K$$

(log n - 1) times

$$T(n) = n*K + n*K$$

$$= (\log n - 1)^* n*K$$

$$= (\log n - 1)\, n*K + K$$

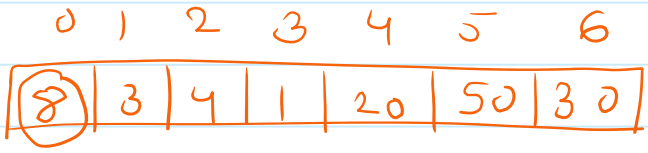
$$= n*K\log n$$

$$n \to n/2 \to n/4 \to \cdots \to 1$$

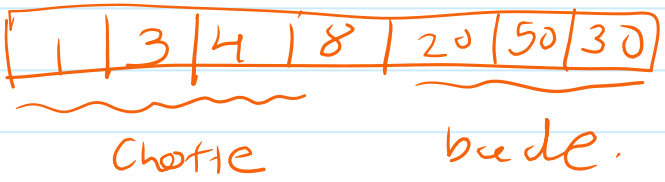
$$\boxed{T(n) = n\log n}$$

10 April 2024 14:56

Quick Sort



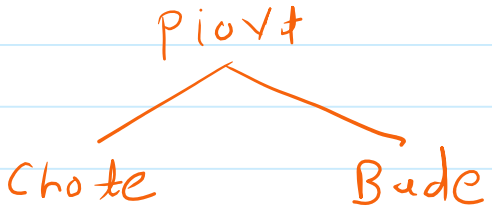
Qs → 1 number place in write position



Partition

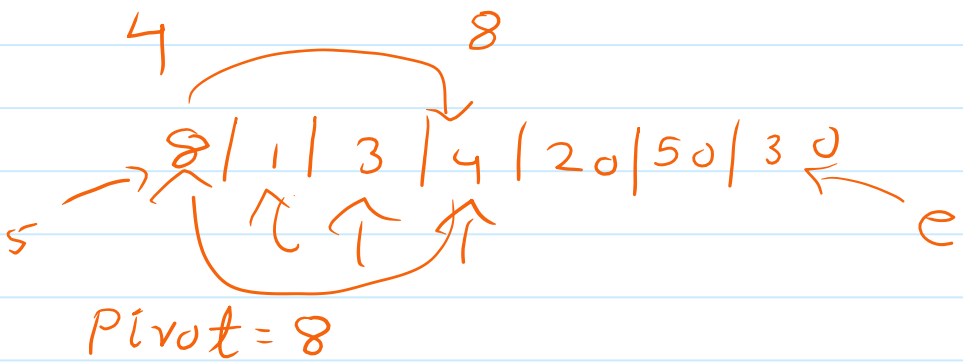
↳ element → pivot
choose

Pivot element in place in right position



Qs → Partition logic

↳ Recursion call

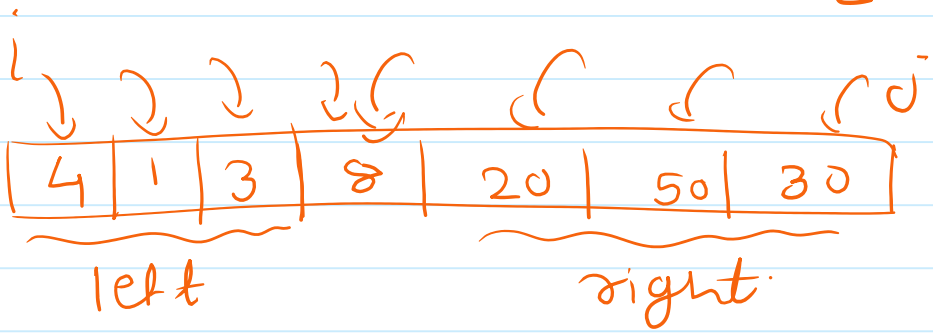


Pivot → right place → Pivot → count small

Count = 3

Swap (arr[pivotIndex], arr[s+count])

③



④ Recursive call

Q.5 → Partition

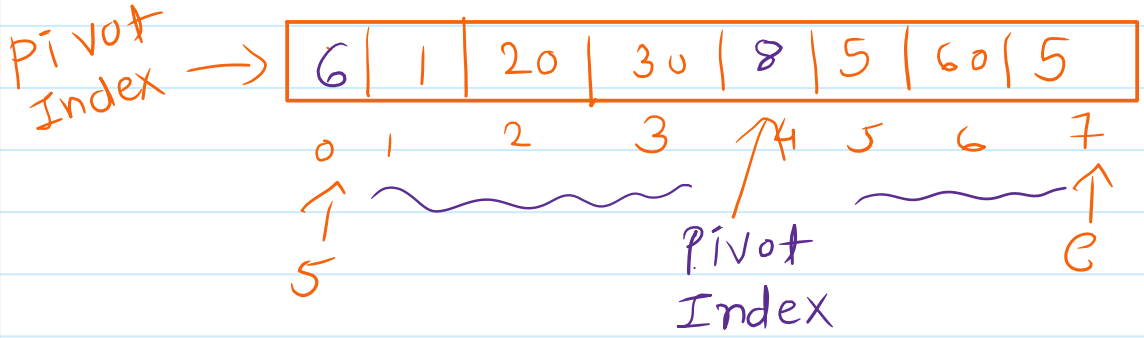
↓

(A) pivot choose

(B) Pivot → right position place

(C) left ← chute && right → bade.

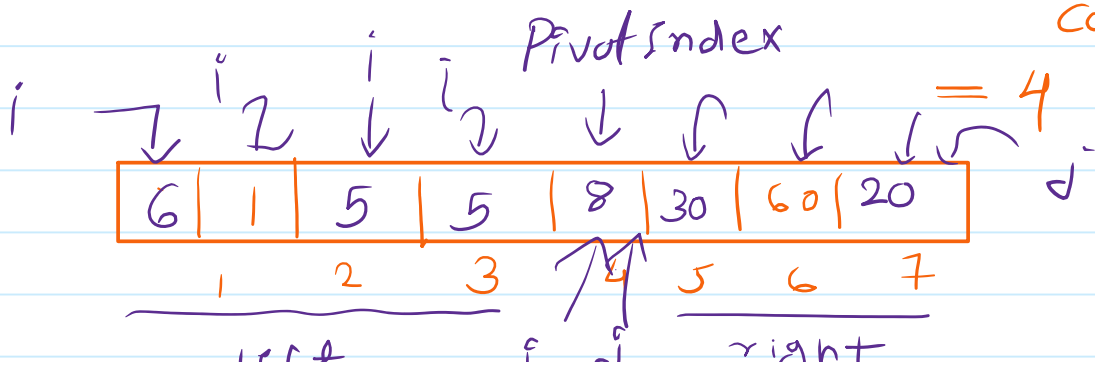
(D) return pivotIndex

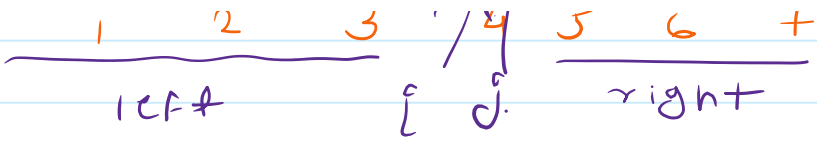


(1) pivotIndex = S = 0

(2) Count <= pivotIndex count = 4

↳ rightIndex = S + count





(3) wrong element

↳ left element

↳ element > pivot.

Right part

element < pivot.

i=2
j=7

Swap (arr[i], arr[j])

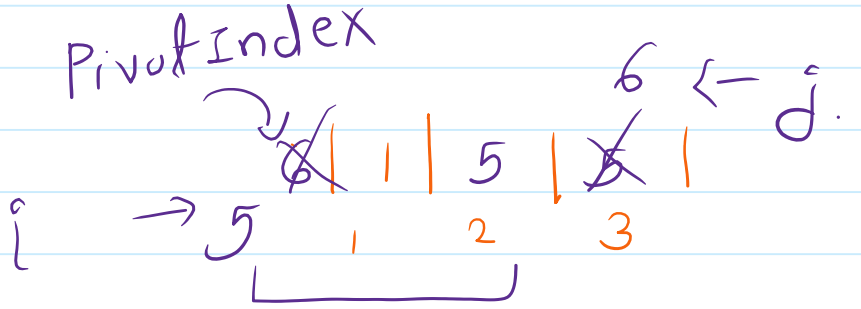
i=3
j=5

swap

i=j = pivotIndex

↓

Rukana (stop)

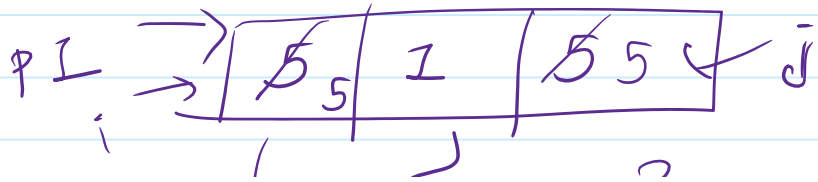


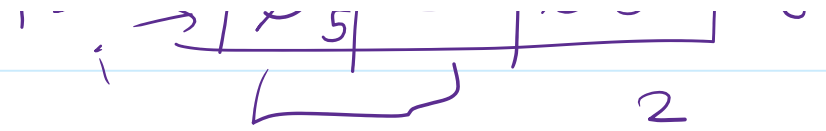
(1) pivotIndex = 0

(2) count = 3

swap (arr[PI], arr[stcount])

(3) ∝





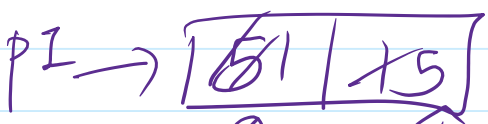
2

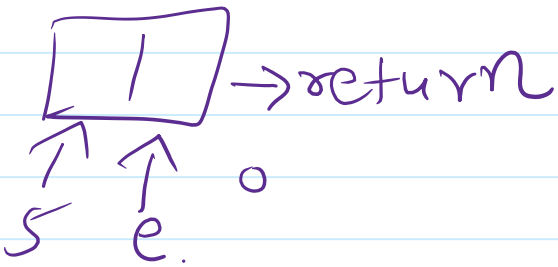
① PI = 0

② count = 2

swap(arr[PI], arr[s+count])

③ x



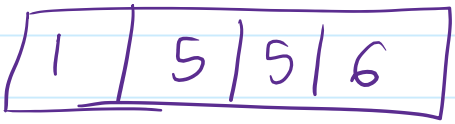


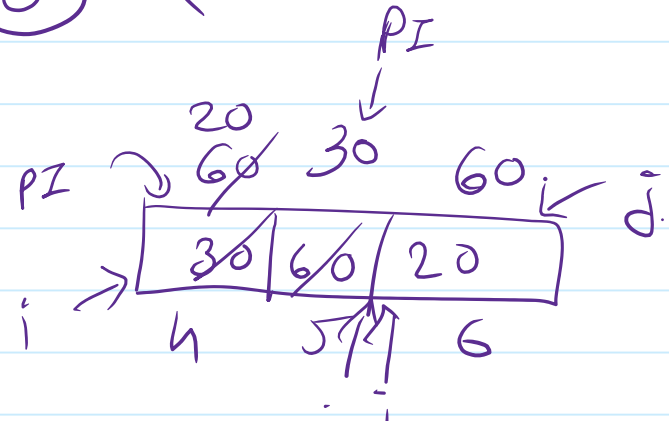
① PI = 0

② count = 1

swap

③ x





Ⓐ PI = 4

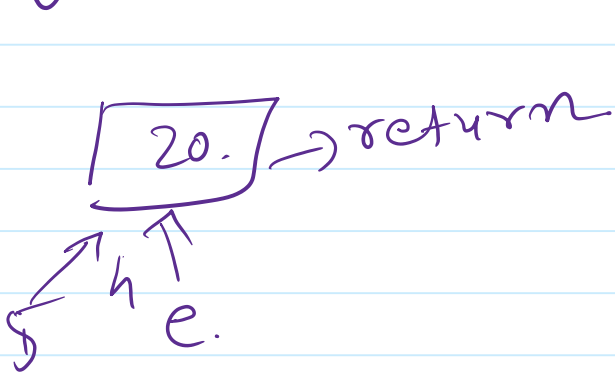
Ⓑ count = 1

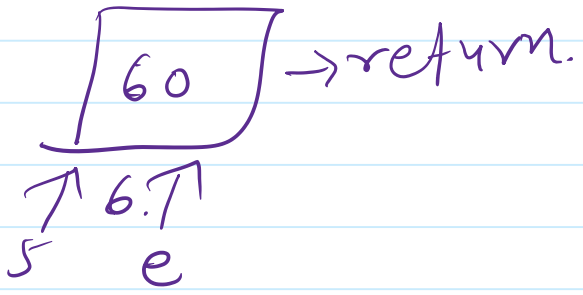
swap

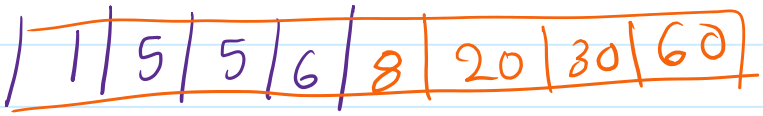
Ⓒ i = 4 & swap.

PI = s+count = 4+1 = 5

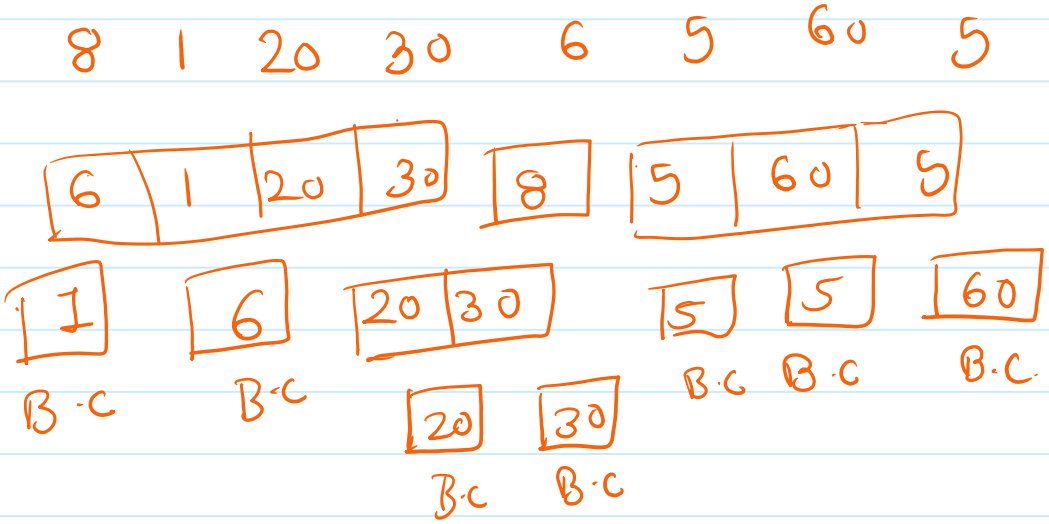
i = 4
j = 6 } swap.





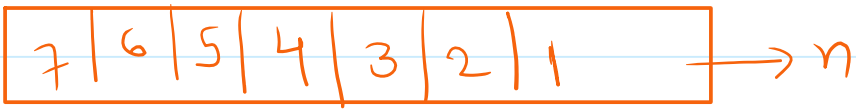


T.C →

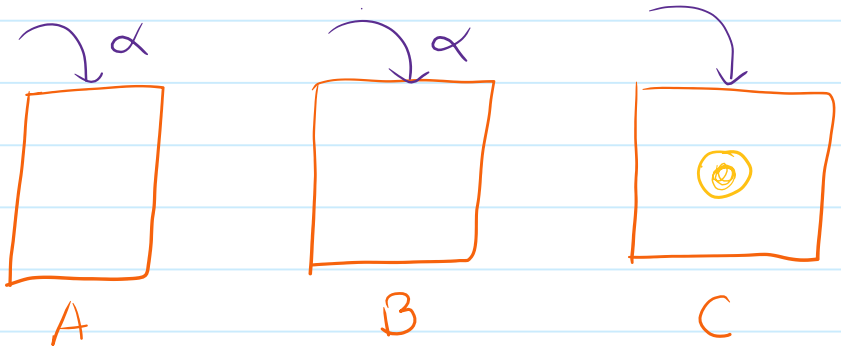


T. C → O(n log n)

worst case → $O(n^2)$

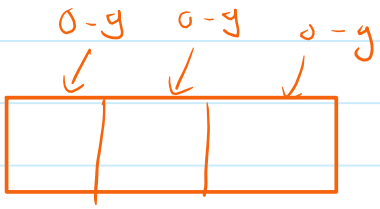


→ Back tracking : → (specific form of recursion)



→ explore all possible solution

discard any solution that not repeate that solution.



000

001

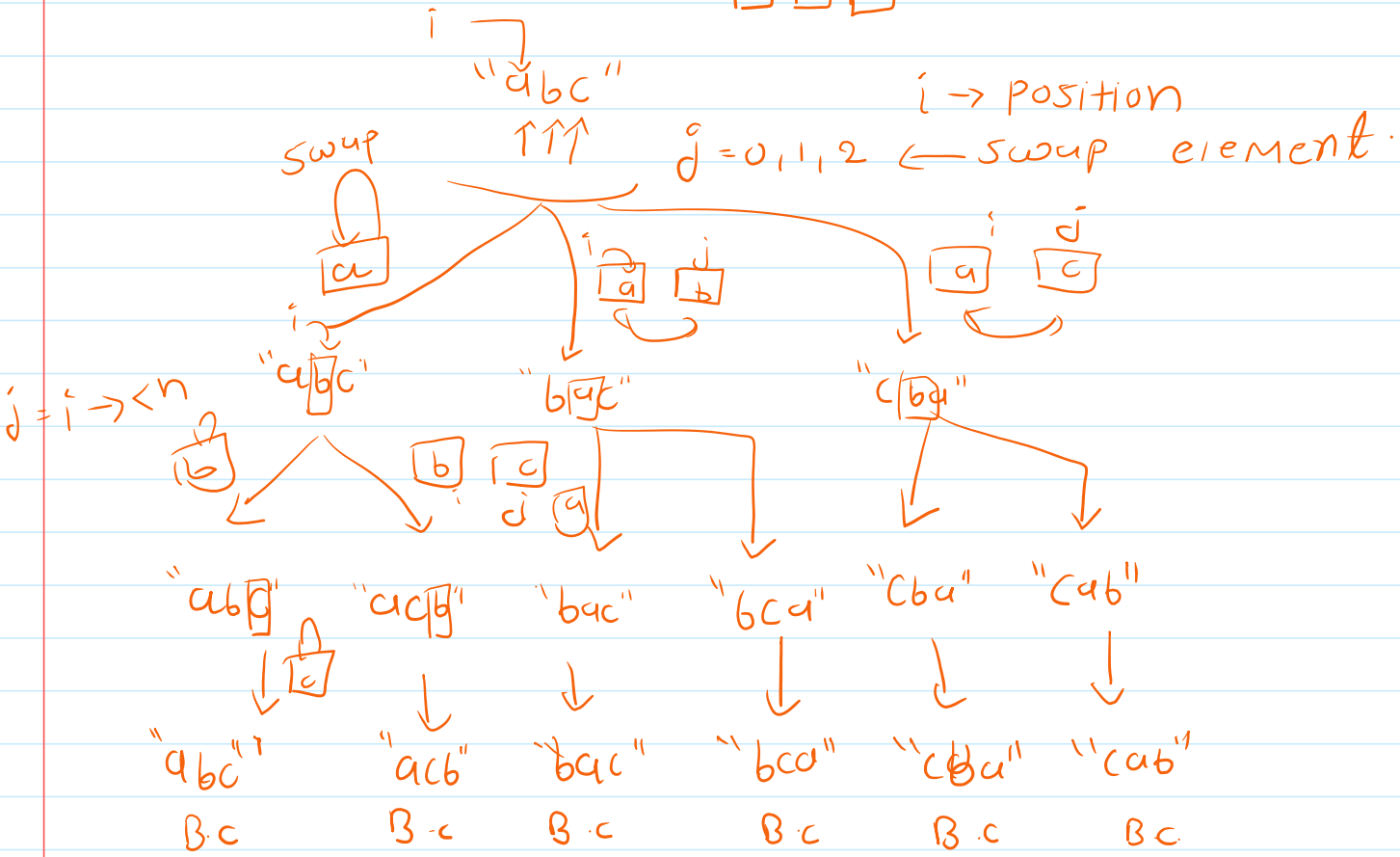
↓

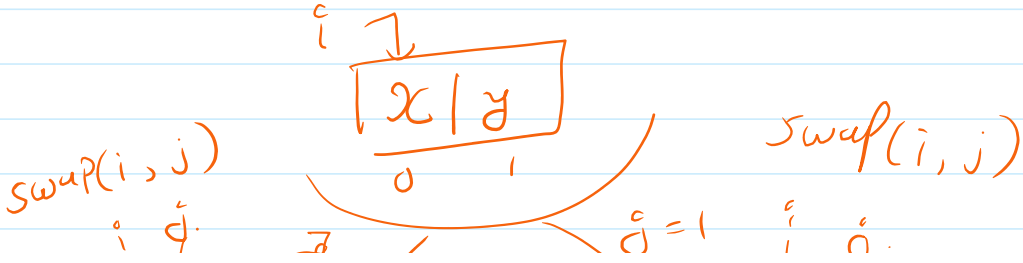
999.

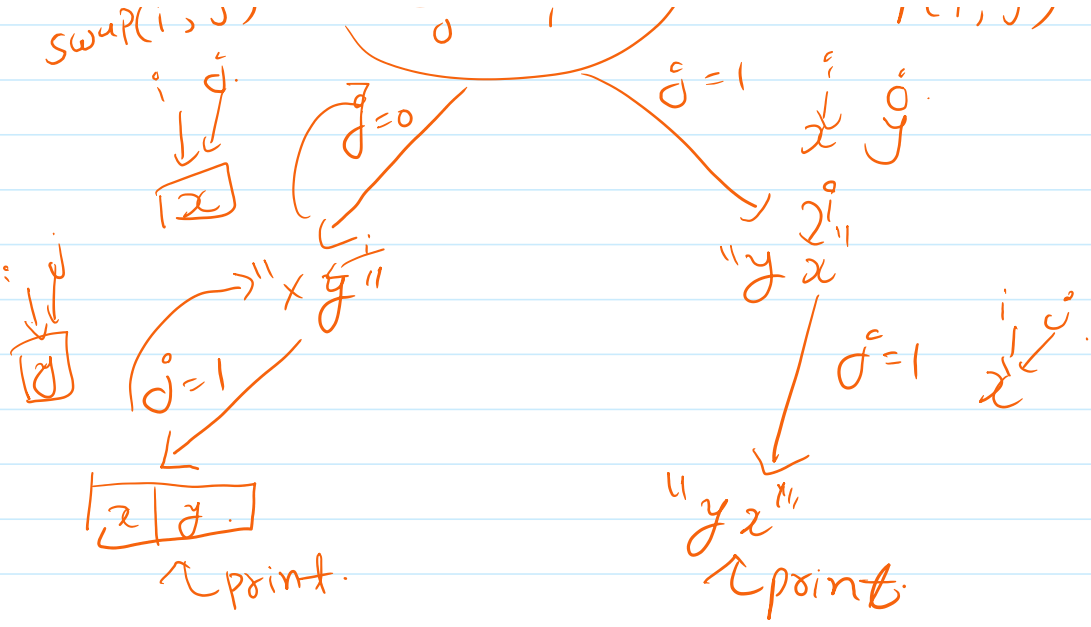
Permutations of String :-

String Str = "abc"

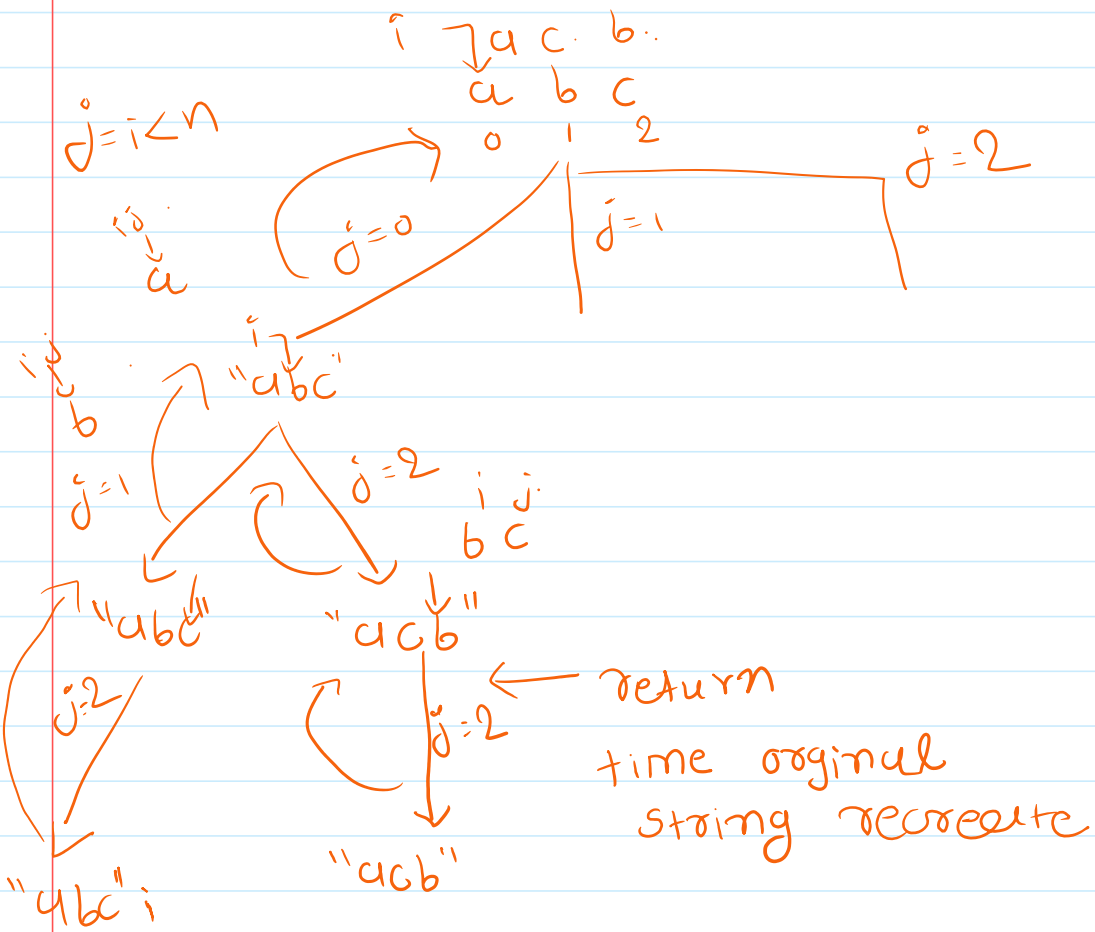
↳ abc
bca
bac
cab
cba
acb







by reference pass



a b c

swap a c b

swap a b c

T.C = O(n!)