→ Quick Sort.

- is a sorting algthm that uses the divide & conquer strategy.

3 steps of quick sort are as follows,

- Divide :

Split the array based on pivot element. All the ele. that are less than pivot should be in left sub array & all the ele. that are more than pivot should be in right sub array.

- Conquer

Recursively sort the 2 sub arrays.

- Combine

Combine all the ele. in a group to form a list of sorted ele.

Consider an array A[] whose i/p ranging from 0 to n-1 then we can formulize the division of array ele as

A[0] ... A[m-1], A[m], A[m+1] ... A[n-1]

ele. are less than A[m] — mid — ele. are greater than A[m]

→ Algorithm Quick(A[0...n-1], low, high)

// Pblm description: Sorting of ele.

// i/p: Array A[0...n-1] unsorted ele.

// o/p: ascending order.

if (low < high) then

{ // split the array into 2 sub arrays

m ← partition(A[low ... high]) // m is mid of the array

Quick(A[low ... m-1])

Quick(A[mid+1 ... high])

```
Algorithm Partition (A[low ... high])
// Pblm description : partitions the sub
                      array using pivot.
// i/p : A -> left (low), right (high).
// o/p : Pivot occupies the right posn.
pivot ← A[low]
  i ← low
  j ← high+1
  while (i <= j) do
  {
      while (A[i] <= pivot do
          i ← i+1
      while (A[j] >= pivot) do
          j ← j-1
      if (i <= j) then
          swap (A[i], A[j]) // swap
  }
  swap (A[low], A[j]) // when i crosses j
                      swap A[low] & A[j]
return j // rightmost index
```

(or)

```
p ← A[l]
i ← l ; j ← r+1
Repeat
  Repeat i ← i+1 until
                A[i] >= p
  Repeat j ← j-1 until
                A[j] <= p
  swap (A[i], A[j])
until i >= j
swap (A[l], A[j])
Swap (A[l],
      A[j])
return j.
```

→ Example: {50, 30, 10, 90, 80, 20, 40, 70}

50	30	10	90	80	20	40	70

→ choose pivot & split the array into sub

Pivot element : 50

50	30	10	90	80	20	40	70
pivot	i					j	j

Left side (i)
① => 50 > 30 moves the ptr i to right →
② => 50 > 10 "
③ => 50 < 90 stop incrementing.

Right side (j)
④ => 50 < 70 moves ptr j ← to left.
50 > 40 so swap A[i] & A[j]
A[i] = 90 & A[j] = 40

50	30	10	40	80	20	90	70
pivot		Left	i	i	j	j	

↳ 50 > 10, increment
↳ 50 < 80 stop incrementing

Right

50 < 90 decrement by 1

50 > 20, stop decrementing j.

Swap A[i] & A[j]

i.e. 80 & 20

50	30	10	40	20	80	90	70

j ↑ i ---- j ↑ i

A[i] < pivot

A[j] > pivot.

So continue incrementing i & decr j

50	30	10	40	20	80	90	70

i, j

50	30	10	40	20	80	90	70

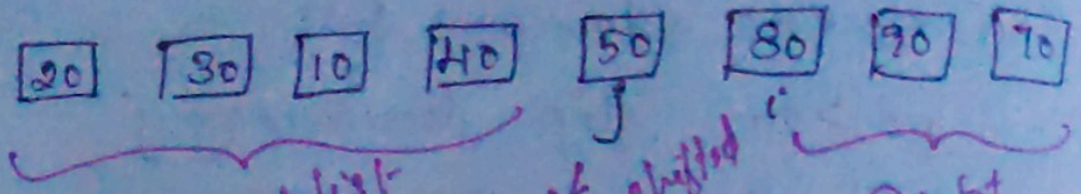
j i

A[j] < 50 & j has crossed i.

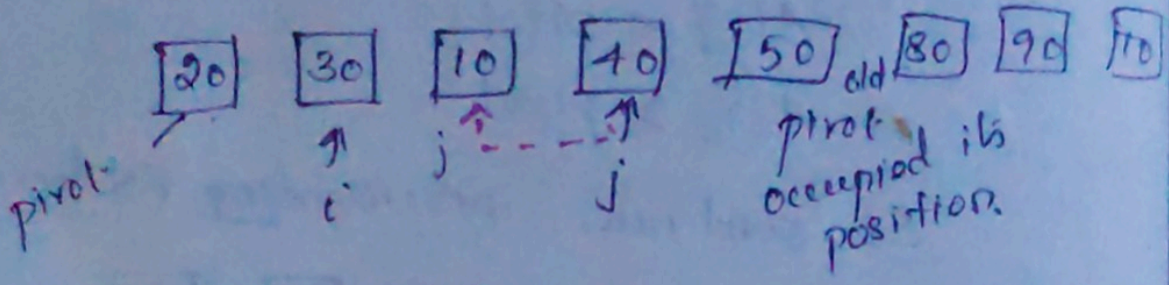
So swap A[j] & pivot.

i.e. & 50

80



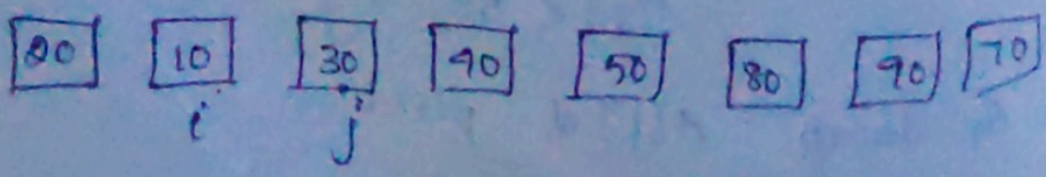
-> Take left sublist :



↳ 20 < 30 -> slop incre

↳ 20 < 40 -> ~~slop~~ decrement by 1
20 > 10 -> stop
so swap A[i] & A[j]

A[i] A[j]
20 & 10



20 > 10 so increment

20 10 30 40 50 80 90 70

i j.

A[i] > pivot or A[j] > pivot decrement j.

20 10 30 40 50 80 90 70

j i

A[j] < pivot & can't decrement j

So swap.

Unsorted

10 20 30 40 50 80 90 70

i j

pivot

left sublist. pivot right sublist

A[i] > pivot.
↳ stop increment

A[j] < pivot stop

∴ swap

10 20 30 40 50 | 80 | 70 | 90

i j

A[i] < 80 increment.

10 20 80 40 50 | 80 | 70 | 90

i j

A[i] > pivot decrement j.

10 20 30 40 50 80 70 90

j i

So swap A[j] & A[pivot].

10	20	30	40	50	70	80	90

This is a sorted list.

↳ Analysis:

Best case (split in the middle)

If the array is always partitioned at the mid, then it brings the best case efficiency of an algorithm.

$T(n) = T(n/2) + T(n/2) + n$

$T(n/2)$ — left sub array; $T(n/2)$ — right subarray; n — Time req. for partitioning the sub array.

& $T(1) = 0$.

→ Using Master theorem

$$T(n) = 2T(n/2) + n$$

Here $f(n) \in n^1$ $\therefore d = 1$

Now, $a = 2$ & $b = 2$.

As from case 2 We get $a = b^d$ i.e. $2 = 2^1$,
we get,

$$T(n) \text{ i.e. } c(n) = \Theta(n^d \log n)$$

$$\therefore c(n) = \Theta(n \log n).$$

Thus, Best case time Complexity of quick sort is $\Theta(n \log_2 n)$.

//

9) And 5 3 1 9 8 2 4 7

8 3 2 9 7 1 5 4