Quick Sort

→ Sort in ascen/desc. order.

T.C ⇒ O(N log N) S.C ⇒ O(1)

arr[] = 4 6 2 5 7 9 1 3

S1 → Pick a pivot element (any element)

P = 4

⇒ Place the pivot in the correct place in the sorted array

S2 ⇒ Smaller on the left, larger on the right

0	1	2	3	4	5	6	7
2	1	3	4	6	5	7	9
smaller			P	greater			

1 2 3 4 5 6 7 9 → Sorted.

let's pick smaller one and again perform same steps

2 1 3

p = 2

S-1 ⇒ 1 2 3

smaller larger

S-2 ⇒ $\boxed{1}$ 2 3

wherever array has '1' element we don't do anything.

because '1' element is sorted in itself.

So, now in Original place it get changed

Now for larger

6 5 7 9

P = 6

S1 ⇒ 5 6 7 9

Smaller Larger

S2 ⇒ $\boxed{5}$ 6 $\boxed{7 \quad 9}$ Original array

7 9

↳ single array

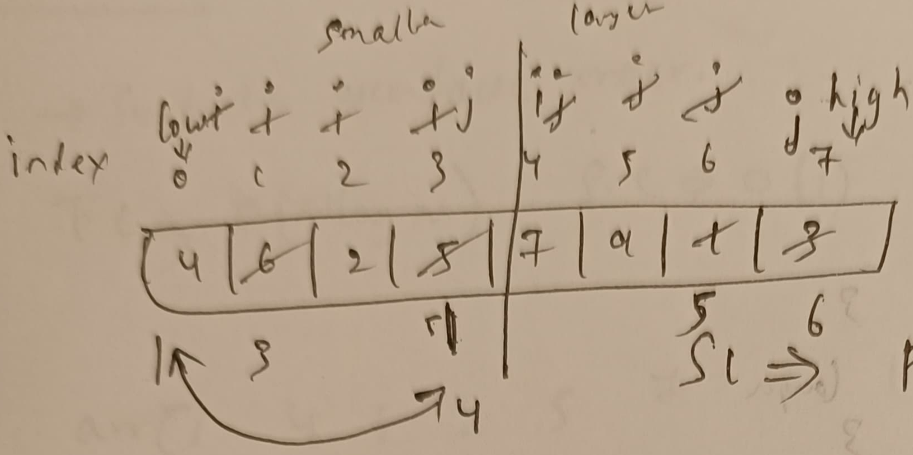
Now for larger

P = 7 7 9

S1 ⇒ 7 9

S2 ⇒ 7 9

* We will use low and high pointer.



S1 ⇒ pivot = a[low] (1st element)

S2 ⇒

figure out 1st guy i greater than pivot in the left

and smaller guy lesser than pivot in the right

and then swap

same thing

1 3 2 | 4 | 7 9 5 6 → keep doing.

arr(low, partion-1)

partition index

arr(partion+1, high)

```
qs(arr, low, high)
{ if (low < high)
  {
  partioIndex = f(arr, low, high)
       qs (arr, low, partioIndex-1)
} }    qs (arr, partionIndex+1, high);
```

```
int f (arr, low, high)

{
  pivot = arr[low];
    i = arr[low];
    j = high;
  while (i<j)
                                     pivot
{ // for left
  while (arr[i] <= ~~arr[pivot]~~ && i <= high-1)
   {i++}

  // for right                   pivot
  while (arr[j] > ~~arr[pivot]~~ && j >= low+1)

   {j--}
  if (i<j) swap(arr[i], arr[j]);
   return j;
}.
```

TC => O(n logN)

SC => O(1)