

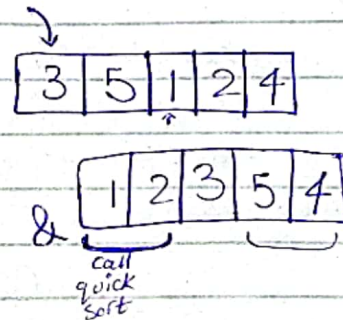
Quick Sort:

Work fast in some cases than merge sort.
Faster 2 to 3 times. Designed by Tony Hoare in 1959

Strategy applied here is divide & conquer

Here we'll do:

- 1) Select pivot element
 - 2) Partitioning (make sure that pivot element reach to it's sorted position & left subarray elements \leq pivotal element & right \geq pivot element)
- \therefore Subarray can be sorted/unsorted



Quick Sort (A, p, r)

Array \downarrow start index \downarrow ending index \downarrow

{
if ($r > p$)
{

$n \leftarrow$ choose pivot

$q \leftarrow$ partitioning (A, p, r)

sorted position of pivot \leftarrow returns

$T(q-1) \leftarrow$ Quick Sort ($A, p, q-1$) \rightarrow left subarray

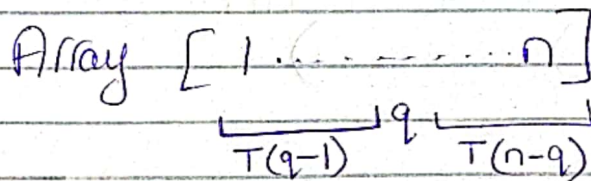
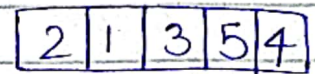
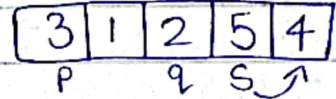
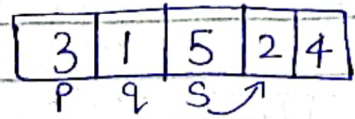
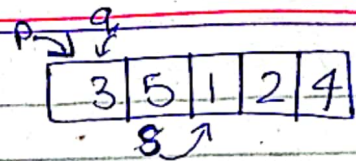
$T(n-q) \leftarrow$ Quick Sort ($A, q+1, r$) \rightarrow right subarray

}
}

```

Partitioning(A, P, r)
{
    q ← P
    for (S ← P+1 to r)
    {
        if (A[S] < A[P])
        {
            swap(A[++q], A[S])
        }
    }
    swap(A[P], A[q])
    return q;
}

```



Recurrence:

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(q-1) + T(n-q) + n & 1 \leq q \leq n \end{cases}$$

If we've sorted array then we get $O(n^2)$
 so this problem is due to selecting 1st
 element as pivot so solution is either
 select randomly or middle one as pivot
 so select it & swap with 1st one

$$T(n) = T(q-1) + T(n-q) + n$$

If $q=1$ (1st index) (pivot at first)

$$T(n) = T(0) + T(n-1) + n$$

$$= 1 + T(n-1) + n$$

$$T(n) = T(n-1) + (n+1)$$

2nd

$$T(n) = [T(n-2) + n] + (n+1)$$

$$= T(n-2) + n + (n+1)$$

3rd

$$T(n) = [T(n-3) + (n-1)] + n + (n+1)$$

$$= T(n-3) + (n-1) + n + (n+1)$$

$$= 1 + 2 + 3 + \dots + (n-1) + n + (n+1)$$

$$= \frac{(n+1)((n+1)+1)}{2}$$

$$= O(n^2)$$

In Best case: $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$

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

$$T(n) = O(n \log n)$$