Quick Sort:

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

Strategy applied here is divide & conquer

Here we'll do:

3	5	1	2	4

1) Select pivot element
2) Partitioning (make sure that pivot element reach to it's sorted position & left subarray elements ≤ pivotal element & right // // ≥ pivot element)

1	2	3	5	4

call quick sort

∴ subarray can be sorted/unsorted

Array, start index, ending index

```
QuickSort(A, p, r)
{
1 ←    if(r>p)
       {
n ←        choose pivot
q (sorted position of pivot) ←returns partitioning (A, P, r)
T(q-1) ← QuickSort(A, P, q-1) → left subarray
T(n-q) ← QuickSort(A, q+1, r) → 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;
}
```

3	5	1	2	4
p, q		S		

3	1	5	2	4
p	q	S		

3	1	2	5	4
p		q	S	

2	1	3	5	4

$$\text{Array } [\underbrace{1 \ldots\ldots}_{T(q-1)} \; q \; \underbrace{\ldots\ldots n}_{T(n-q)}]$$

Recurrence:

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

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

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

$\vdots$

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

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

$= O(n^2)$

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