Quick SORT

Quick SORT is divide & conquer type algorithm
First we will see it's Algo. & then
Analyse it.

```
Void QuickSort (int a[], int l, int h)
{
    int J;
    if (l<h)
    {
        J = partition (a, l, h)
        Quicksort (a, l, J);
        Quicksort (a, J+1, h);
    }
}
/* This was recursive function */
/* Now partion function */
int partition (int a[]; int l, int h)
{
    int i, J, Pivot;
    Pivot = a [l];
    i = l ; J = h;
    do {
        do { i++;} while (A[i] <= Pivot);
        do {J++;} while (A[J] > Pivot);
    if (i<J)
    swap (& A[i] & A[J], );
    }
```

while (j<J);
swap (&pivot & A[J]);
return J; }

Now lets take an array to be sorted using be Quick SORT.

50, 70, 60, 90, 40, 80, 10, 20, 30, ∞

no. of elements = 10 = n
so let l=0, h=n-1 = 9
let pivot = 50

i = 0, J = 9

(0)
i will start from left side & look for elements greater than 50, once it found the element greater than 50 then it will stop further.

Now it is 'J' turns

'J' starts from right & moves to left looking for elements smaller than or equal to pivot once it found element it will stop.

If i smaller than J, then we will swap value of A[i] & A[J] this process will go until 'i' is smaller than 'J'

when J becomes smaller than i, then there will be swapping.

Pivot & A[J]

& then we will return value of J;

to Quick Sort & (x)^h

& these steps will go on until all the array elements are sorted.

Now analysis of Quick sort

Average Case time complexity is $O(n \log n)$

Best case scenario is when pivot comes to the middle after partition & complexity is $O(n \log n)$

If list is already sorted.

Then from algorithm we can see that pivot after swapping comes to end of array which is work case & complexity in work case $O(n^2)$!

How to Overcome

If list is already sorted the bring middle element at Ist position & then apply sorting.

Now, we can say that if position is at end ~~then~~ after swapping then worst case complexity $O(n^2)$

Average Case :- If Partition not occurs at end & complexity is $O(n \log n)$;