★ Quick Sort:

→ Quick sort is a devide and conqure sort. And it is also recursive sort.

→ In a quick sort we define three pointers.

① Pivot
② p
③ q

→ Pivot stand for center point.

→ q is used to find small elements with Help of pivot.

→ ~~Per~~ conditions:

→ while scanning (left to righ and right To left) if "p" pointer and "q" pointer crossing each other then "Pivot and q" Replace their position.

→ If not ~~crossing~~ p and q pointer are Not crossing each other then simply "p and q" pointers change position Respectively.

Ex → Perform quick sort on following data:
X = 35, 50, 15, 25, 80, 20, 90, 45

→ Pass-I :

35, 50, 15, 25, 80, 20, 90, 45
pivot p q

→ we set 1st element 35 as pivot.

→ Now with the help of pivot and p pointer we scan (check) Big element with comparing pivot and p.

→ As it is with the help of comparing pivot and q we find the small element.

a) 35, 50, 15, 25, 80, 20, 90, 45
pivot p q

→ There is p and q not crossing Each other we change the position.

b) 35, 20, 15, 25, 80, 50, 90, 45
pivot p q

→ we find the big element.

c) 35, 20, 15, 25, 80, 50, 90, 45
pivot q p

→ There is p and q pointers are crossing each other we change the position of pivot and q.

d) 25 20 15 35 80 50 90 45

→ Now there is pivot is fixed.

→ Pass-II :

a) 25, 20, 15, 35, 80, 50, 90, 45
pivot

b) 25, 20, 15 — 35 — 80, 50, 90, 45
pivot p q pivot p ... q
left sub array Right sub array

→ In left sub array p and q pointer Are crossing each other we change the position of pivot and q.

→ In right sub array p and q pointer Are not crossing we change position of p and q.

c) 15, 20, 25 — 35 — 80, 50, 45, 90
pivot p ... q

→ In right sub array p and q pointer Are crossing each other we change the position of pivot and q.

d) 15, 20, 25, ^{35}, 45, 50, 80, 90

→ Now the array is sorted.

Time and space complexity :

space complexity :

→ space complexity of an algorithm represent the amount of memory space required by the algorithm in its life cycle.

→ The space required by an algorithm is equal to the sum of the following two components:

ⓐ A fixed part that is a space required to store certain data and variables, That are independent of the size of the problem. For example, simple variables and constants used, program size, etc....

ⓑ A variable part is a space required

dynamic memory allocation, recursion stack space, etc...

→ Space complexity of any algorithm p is S(P)

S(P) = C + SP(I), where C is the fixed part And S(I) is the variable part of algorithm. I is instance characteristic.

Eg: Algorithm : Sum(A,B)

step-1 :- start
step-2 :- C ← A + B + 10
step-3 :- stop

→ Time complexity:

→ Time complexity of an algorithm represents the amount of time required By the algorithm to run to completion. Time requirements can be defined as A numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.