Page No.
Date : / /

Unit : 3 : Divide and Conquer Algorithms.

* What is Divide and Conquer algorithm method.

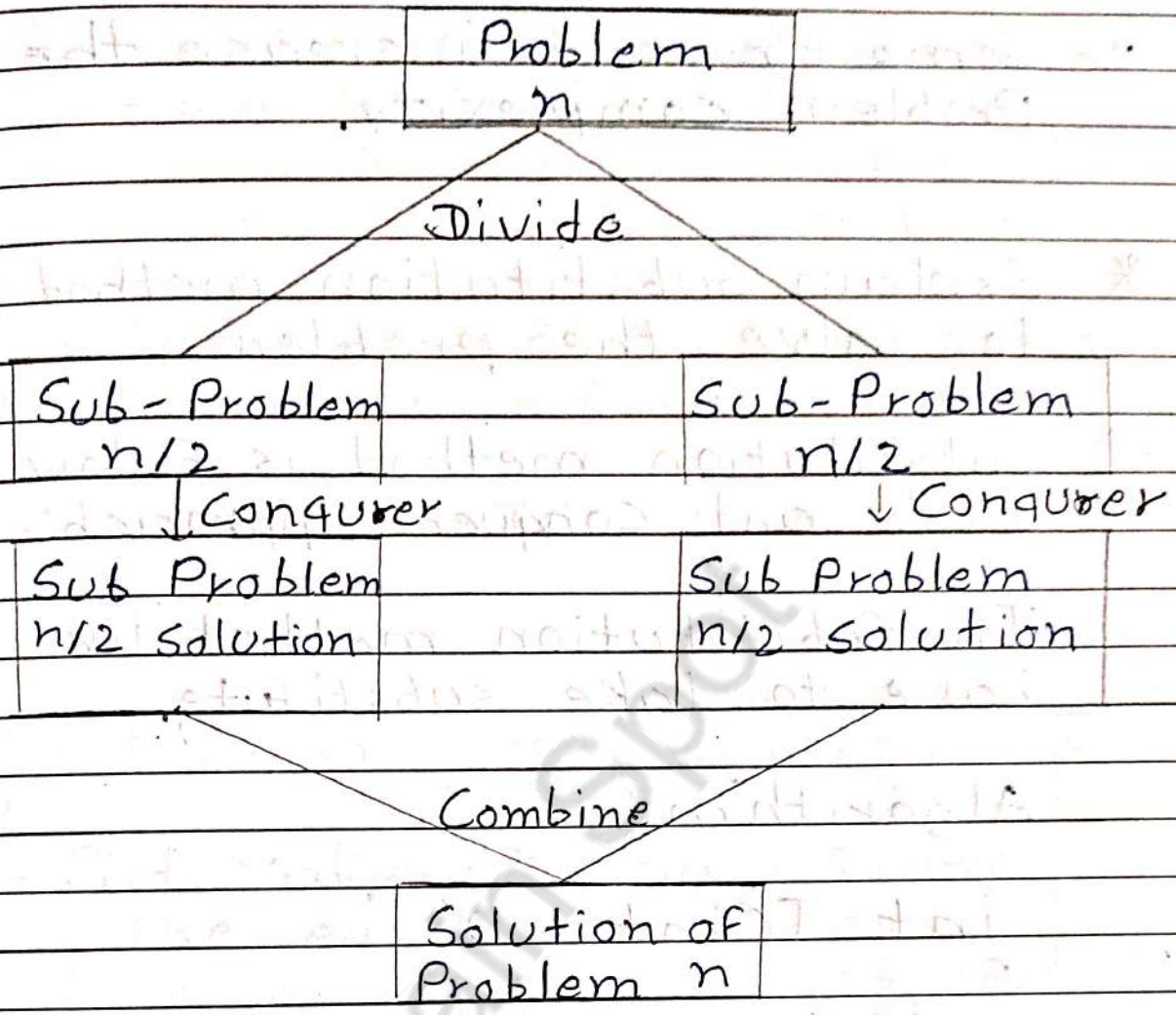
=> In Divide and Conquer method, Problem is divided into three parts.

a) Divide
b) Conquer
c) Combine.

a Divide : In this part, we have to divide the whole problem into the sub-problem.

b Conquer : In this part, we have to find the solution of sub problem.

c Combine : In this part, we have to combine the solution of the sub problem.



- Advantages:

1 In this method, Problem is divided So, it is decrease the time.

2 This Method can reduces time complexity Problem.

- Disadvantages:

1 In this method, some time Crash the system.

2 Some time it increase the Problem complexity.

* Explain Substitution method to solve the problem.

=> Substitution method is follow divide and conquer approach.

In Substitution method, we have to take substitute.

Algorithm:

```
int T(int n)
{
    if (n=1)
        return 1;
    else
        return T(n-1);
}
```

Example:

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

For Substitution Method, we have to take T(n).

$\therefore T(n) = T(n-1) + 1$ — (1)

In this method, we have take n-1, n-2, ... substitute for n.

$\therefore T(n-1) = T(n-2) + 1$ — (2)

$\therefore T(n-2) = T(n-3) + 1$ — (3)

$\therefore T(n-3) = T(n-4) + 1$ — (4)

Put value of T(n-1) into the equaction one.

$\therefore T(n) = T(n-2) + 2$ — (5)

Again, put T(n-2) value in equation 5,

$\therefore T(n) = T(n-3) + 3$ — (6)

Here, We can see one pattern in equation 5 and 6. ∴

So, We can take K, in the place of 2 or 3.

$\therefore T(n) = T(n-k) + k \quad -(7)$

From the example, we have value of $T(1) = 1$.

For Finding ~~tha~~ the, value of k,

$\therefore n - k = 1$

$\therefore k = n - 1$

Put value of k into the equaction 7.

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

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

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

From the example, we have value of $T(1)$

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

$\therefore T(n) = n$

According to Order of growth, Function $T(n)$ is fall in Big O Notation.

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

* Explain Recurrence Tree method to solve the problem.

=> Recurrence Tree method is follow divide and conquer approach.

In this Method, we have to create a tree to solve the Problem.

If Given Problem,

$$T(n) = \begin{cases} nT(n/2) + n^2, & n > 1 \\ n \cdot 1, & n = 1 \end{cases}$$

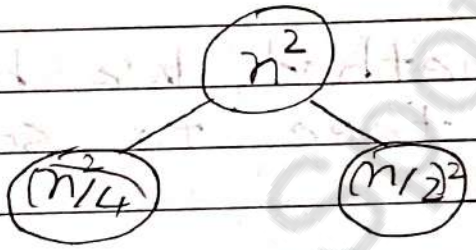
Here, $T(n) = nT\left(\frac{n}{2}\right) + n^2$

Here, $n/2$ become division factor of root node n^2 and n become a number in which tree is divide.

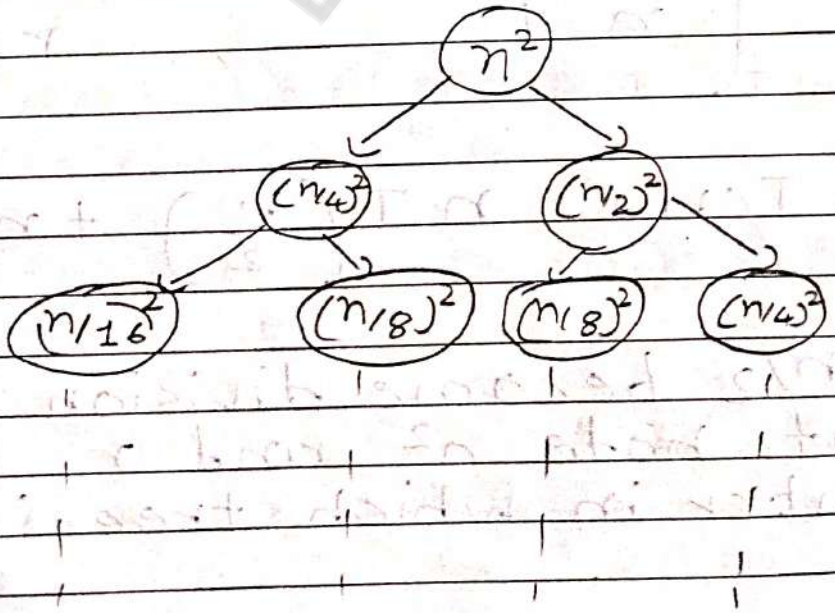
Page No.
Date : / /

Ex. $$T(n) = \begin{cases} T(n/4) + T(n/2) + n^2, & n > 1 \\ 1, & n = 1 \end{cases}$$

Here, Given root element is n^2 and $n/4$ and $n/2$ become division factor of n^2.



In this method, we have to find the cost for divide the root element at every level of tree.



Page No.
Date : / /

Here, total Work done for

$$T(n) = n^2 + \frac{5}{16}n^2 + \frac{25}{256}n^2 + \dots$$

$$T(n) = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \dots\right)$$

According to Order of growth $T(n)$ will be fall into the $\Theta(n^2)$

$\therefore T(n) = \Theta(n^2)$

* Explain Master method to solve the problem.

=> Master method is follow Divide and Conquer approach.

If Given $T(n) = aT(n/b) + \Theta(n^k \lfloor \log n \rfloor^p)$

where, $a >= 1, b > 1, k >= 0$

For Finding the solution, we have to use this three case,

Case 1: if $\log_b a > k$ then

$\Theta(n^{\log_b a})$

Case 2 : $\log_b a = k$ then

a) If $P > -1$ then $\Theta(n^k (\log n)^{P+1})$

b) If $P = -1$ then $\Theta(n^k \log(\log n))$

c) If $P < -1$ then $\Theta(n^k)$

Case 3 : $\log_b a < k$ then

a) If $P \geq 0$ then $\Theta(n^k (\log n)^P)$

b) If $P < 0$ then $O(n^k)$

According to value of $\log_b a$ we have to select the case.

Ex.

$$T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$$

We have to compare this eqn to,

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k (\log n)^P)$$

So, We get $a = 2$, $b = 4$, $k = 0.5$ & and $P = 0$.

First we have to find value of $\log_b a$

$\therefore \log_b a = \log_4 2 = 0.5$

Value of $\log_b a < k$ and $P = 0$

So, Case 3(a) Condition is satisfied.

So, ~~FC~~ $T(n) = \Theta(n^{0.5} (\log n)^0)$

$$T(n) = \Theta(n^{0.5})$$

* Explain Binary Search Tree with its example.

=> For Binary Search Tree, Tree must follow this two condition,

1) Left side element contain less value to the node element.

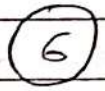
2) Right side element contain greater value or equal value to the node element.

Ex. Create

A Binary Search Tree is also called an Ordered Binary Tree.

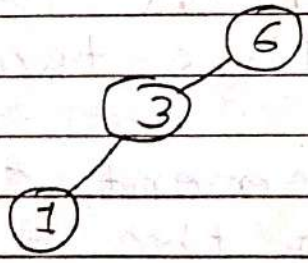
Ex. Create Binary Search Tree,
6, 3, 1, 5, 2, 8, 7

→ Take First element : 6

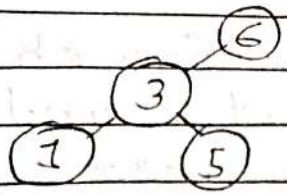


take Second element : 3,
3 < 6 So, Place in left side.

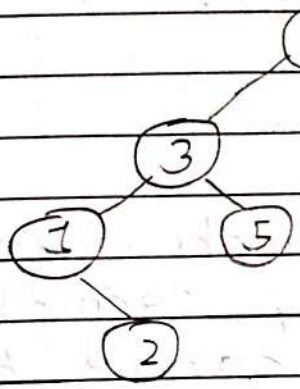
take Data = 1, 1 < 3, So, Place in Left side.



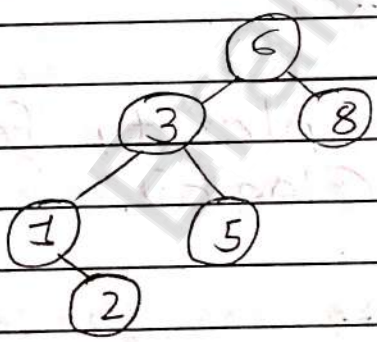
take Data = 5, 5 > 3 and 5 < 6
So, Place in Right side of 3



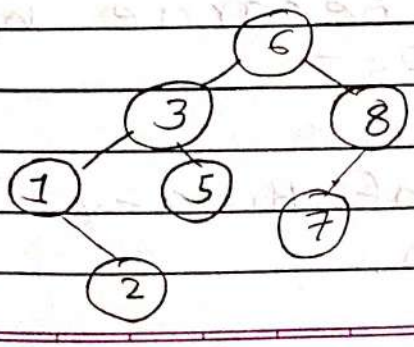
take Data = 2, 2 > 1, So, Place in Right side of 1.



take Data = 8, 8 > 6, So, Place in Right side of 6.



take Data = 7, 7 < 8, So, Place in Left side of 8.



Page No.
Date : / /

* Explain Binary Search Trees time complexity and Write Binary Search Tree Operations with its time complexity.

=> Time Complexity For Binary Tree search Tree:

- Best Case:

If Binary Search Tree is a balanced binary search Tree then it is become best case for BST.

So, Height of Binary tree becomes log(n).

So, Time Complexity for BST become O(logn).

- Worst Case:

If Binary Search Tree is a skewed binary search tree then it is become worst case for BST.

So, Height of Binary Tree becomes n.

So, Time Complexity for BST become O(n).

=> There are three types of Operation is Perform on Binary Search Tree.

1) Insert A Node
2) Search A Node
3) Delete A Node.

1 Insert A Node:

Using Insertion operation, we can insert new node in Binary Search Tree.

For Insert the New element in a Node, we have to traverse all the element in BST.

So, Time Complexity for Worst-Case For Insertion operation become O(n).

2 Search A Node:

Using Searching Operation, we can search a any element in BST.

For Search the element in Binary Search tree, we have to traverse all the element.

So, Time Complexity, For Searchin Operation, For Worst Case is become O(n).

3 Delete A Node:

For Delete a Node, there are three possibility For Node.

(i) Delete, node has no child
(ii) Delete, node has one child
(iii) Delete, node has two child.

For delete the element in Binary Search tree, we have to traverse all the element.

So, Time Complexity, For ~~Sear~~ Deletion Operation, For Worst Case is become O(n).

Page No.
Date : / /

* Explain Strassen's Matrix Multiplication Method with Example.

=> If there are two Matrix,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Using Strassen's Matrix Multiplication, we can find Multiplication of 2x2 Matrix.

$$\therefore \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Using Strassen's Matrix Multiplication,

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_2 = (A_{21} + A_{22}) * B_{11}$$
$$P_3 = (B_{12} - B_{22}) * A_{11}$$
$$P_4 = (B_{21} - B_{11}) * A_{22}$$
$$P_5 = (A_{11} + A_{12}) * B_{22}$$
$$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$$

For Value of,

$$C_{11} = P_1 + P_4 + P_7 - P_4$$

$$C_{12} = P_3 + P_5$$

$$C_{21} = P_2 + P_4$$

$$C_{22} = P_1 + P_3 + P_6 - P_2$$

In this Method there are 7 multiplication is perform.

So, ~~Recursive~~ Equation will be

$$T(n) = \begin{cases} 7T\left(\frac{n}{2}\right) + an^2, & n > 1 \\ 1, & n \leq 1 \end{cases}$$

Using Master Method, we can find Time complexity.

$$\therefore T(n) = 7T\left(\frac{n}{2}\right) + an^2$$

So, Value of $a = 7$, $b = 2$, $k = 2$ and $p = 0$

Value of $\log_b a = \log_2 7 = 2.80$

So, $\log_b a > k$.

So, Time Complexity $\Theta(n^{2.8})$

-> Example:

$$A = \begin{bmatrix} 7 & 5 \\ 6 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix}$$

Value of $A_{11} = 7$, $A_{12} = 5$
$A_{21} = 6$, $A_{22} = 3$

Value of $B_{11} = 2$, $B_{12} = 1$
$B_{21} = 5$, $B_{22} = 1$

Using Strassen's Matrix Multiplication,

$P_1 = (10)(2) = 20$
$P_2 = (18)(9) = 18$
$P_3 = 0$
$P_4 = 3(3) = 9$
$P_5 = 12(1) = 12$
$P_6 = (-1)(3) = -3$
$P_7 = 2(6) = 12$

$C_{11} = 20 + 9 + 12 - 9 = 32$
$C_{12} = 12$
$C_{21} = 18 + 9 = 27$
$C_{22} = 20 + 0 + (-3) - 18$
$C_{22} = 9$

$$A \times B = \begin{bmatrix} 32 & 12 \\ 27 & 9 \end{bmatrix}$$

* Explain Merge Sort with its example.

=> Merge Sort is the sorting method that follows the divide and conque method.

In Merge Sort, Array is divide into the different part.

Merge Sort is one most popular and efficient mettod.

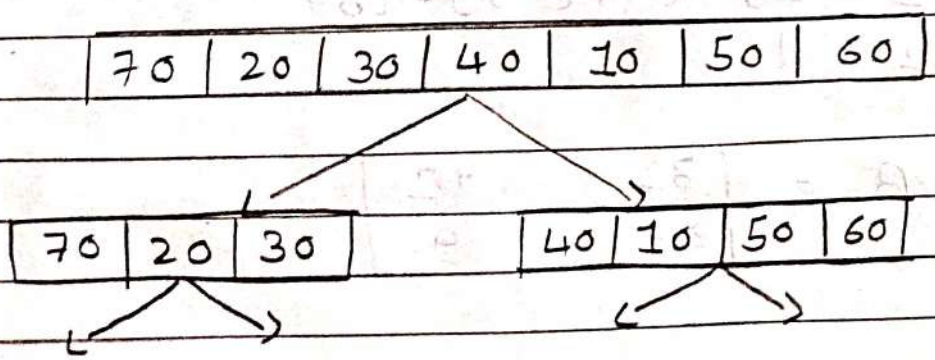
Ex.

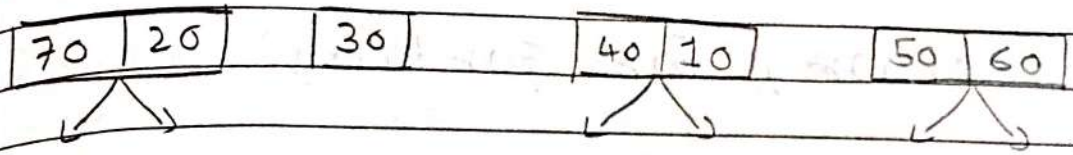
70	20	30	40	10	50	60

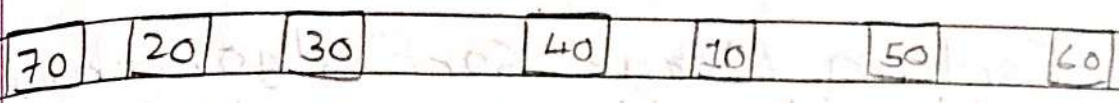
In Merge Sort, this array is divide into equal part.

This array is divide into ~~two~~ Three and four part.

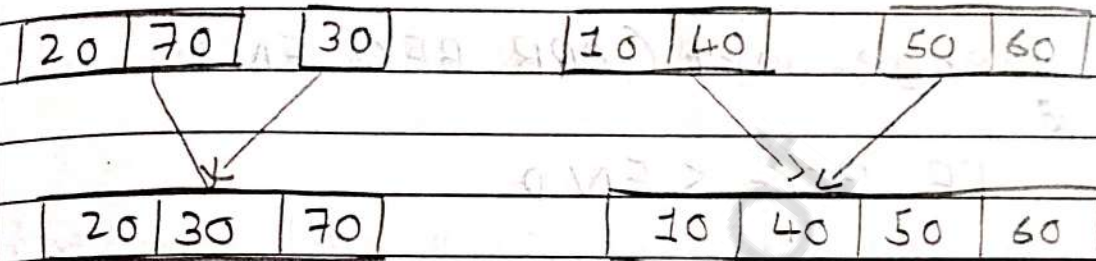
This array is divide into equal part until the array contains only one element.

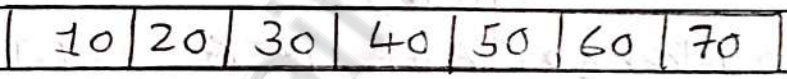






Here, We Merge 20 and 70 or 10 and 40 or 50 and 60





This is Sorted element list.

-> Advantages:

1 It can be used for internal and external sorting.

2 It is efficient and easy algorithm

-> Disadvantages:

Merge Sort is needs a extra

memory for sorting.

* Explain Merge sort algorithm with its time complexity.

=> Merge Sort Algorithm:

```
Merge_Sort (ARR, BEG, END)
{
  IF BEG < END

     SET MID = (BEG+END) / 2
     CALL Merge-sort(ARR, BEG, MID)
     CALL Merge_Sort(ARR, MID+1,
          END)
     Merge(ARR, BEG, MID, END)

  END
```

For Merge Sort, Equaction of Time Complexity will be,

$$T(n) = \begin{cases} 2T(n/2) + n, & n > 1 \\ 1, & n = 1 \end{cases}$$

Page No.
Date : / /

Using Mater Method, We can Find Time complexity.

$T(n) = 2T(n/2) + n$

Value of $a = 2$, $b = 2$, $k = 1$ and $P = 0$.

Value of $\log_b a = \log_2 2 = 1$

Here, $\log_b a = k$

So, Time Complexity = $\Theta(n \log n)$

For All the Case, Merge Sort time complexity is $O(n \log n)$.

* Explain Quick sort with its example.

=> Quick sort is the fastest algorithm to the another sort.

Quick sort working is easy and fastest.

In Quick sort, we have to set first.

element as a Pivot element.

Left side of the Pivot element is less than the pivot element.

Right side of the pivot element is greater than the pivot element.

Ex. 90, 77, 60, 99, 55, 88, 66

Here, 90 element is pivat element then check R->L, 66<90->swap

66, 77, 60, 99, 55, 88, 90
P

then check L->R, 99>90->swap.

66, 77, 60, 90, 55, 88, 99.
P

then check R->L, 88<90->swap

66, 77, 60, 88, 55, 90, 99
P

Here, we see the Left side is small element and Right side has large element to the Pivot.

So, take First element as Pivat.

check R → L, 55 < 66 → swap

55, 66, 77, 60, 88, 66, 90, 99
P P

check L → R, 66 > 77 → swap

55, 66, 60, 88, 77, 90, 99
P

check R → L, 66 > 60 → swap

55, 60, 66, 88, 77, 90, 99
P

Again Pivot element is set in proper position, than take 55 as a pivot.

55, 60, 66, 88, 77, 90, 99
P

Again change Pivot element.

55, 60, 66, 88, 77, 90, 99
P

Again change Pivot element

55, 60, 66, 88, 77, 90, 99
P

Again change Pivot element
check R → L, 77 < 88 → swap

55, 60, 66, 77, 88, 90, 99

Here, all the element are sorted.

-> Advantages :

Working of Quick sort is fast and more efficient than other sort.

-> Disadvantages :

Quick sort worst case complexity is $O(n^2)$.

* Explain Quick Sort Algorithm with time complexity.

=> Quick Sort Algorithm :

```
Quick_sort(ARR, BEG, END)

IF (BEG < END)

  CALL PARTITION(ARR, BEG, END,
       LOC)
  CALL Quick_sort(ARR, BEG, LOC-1)
  CALL Quick_sort(ARR, LOC+1, END)
END
```

-> Best Case :

When algorithm always choose middle element as a pivot element than it is become best case.

For Best Case, Time Complexity equation,

$$T(n) = 2T(n/2) + \Theta(n)$$

Using Mater Method, we can find time complexity

Value of $a = 2$, $b = 2$, $k = 1$ and $p = 0$

Value of $\log_b a = \log_2 2 = 1$

Here, $\log_b a = k$

So, Time Complexity = $\Theta(n \log n)$

For Best Case, Quick Sort time complexity is $\Theta(n \log n)$

-> Worst Case :

If Algorithm take smallest or largest element as a pivot element than it is become worst case for Quick sort.

For Worst Case, time Complexity equation,

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

Using Substitution method, we can find time complexity.

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

$$T(n-2) = T(n-3) + c(n-2)$$

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

$$T(n) = T(n-3) + c(n-2) + c(n-1) + n$$

$$\therefore T(n) = T(n-3) + c(c(n-2) + (n-1) + n)$$

$$\therefore T(n) = c(cn + (n-1) + (n-2)) + T(n-3)$$

$\therefore T(n) = c(n + (n-1) + (n-2) + \ldots + (n-k)) + T(n-(k+1))$

$\therefore n-(k+1) = 1$

$\therefore k = n-2$

$\therefore T(n) = c(n + (n-1) + (n-2) + \ldots + (n-n-2)) + T(n-n+2-1)$

$\therefore T(n) = c(n + (n-1) + (n-2) + \ldots + 2) + T(1)$

$\therefore T(n) = 1 + 2 + 6n + (n-1) + (n-2)$

$\therefore T(n) = c(n + (n-1) + (n-2) + \ldots + 2) + 1$

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

According to Order of Growth $T(n)$ is fall into the $O(n^2)$.