AKG sir
Page No.
Date: / /

Insertion sort & its analysis:-

Input to this function is array of integers.
We consider the index of array starts from 1.

```
Insertion sort(A)
{
    n = A.length
(1) for j = 2 to A.length
    {
(2)   key = A[j] ;
(3)   i = j - 1 ;
(4)   while( i>0 && A[i] > key)
      {
(5)     A[i+1] = A[i];
(6)     i = i - 1;
      }
(7)   A[i+1] = key;
    }
}
```

comparison takes some time so we consider it

for (j=2; j<=A.length; j++)

assignment takes very less time so we don't take it

These all are the instructions

→ Running time = f(i/p size)

i/p size (n) → for sorting n is the length of array.

→ We require two things to analyse the function i.e. cost of each instruction & frequency of instruction.

↓ how many time the instruction requires to run.

↓ how many number of times the particular instruction is evaluated

Instruction number.	Cost of each instruction	Frequency of each instruction
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{j=2}^{n} t_j$
5	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$n-1$

(Cost of running instruction for one time only (that will be constant))

→ For while loop to be executed we do not know how many times comparision is made so we cannot give a constant cost to that instruction. So, we take that for j^{th} iteration t_j is the number of times cond" is checked

t_j :- for j^{th} iteration of the for loop t_j denotes number of times the condition is checked

→ The condition checking inside while loop takes some constant time to run. So we denote by c_4. but that condition will be checked for t_j time

→ The total cost of instruction is:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Best Case :- Array is sorted in increasing order
$t_j = 1$ for $j = 2, 3, \ldots n$

If $t_j = 1$ then the total cost is

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^{n} 1 + C_5 \sum_{j=2}^{n} 0 + C_6 \sum_{j=2}^{n} 0 + C_7(n-1)$$

$$\therefore T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_7(n-1)$$

$$\therefore T(n) = (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_2 + C_3 + C_4 + C_7)$$

Let $A = C_1 + C_2 + C_3 + C_4 + C_7$

$B = -(C_2 + C_3 + C_4 + C_7)$

$$\therefore \boxed{T(n) = An + B}$$ (linear function of n)
↳ for best case

Worst case :- When array is given in decreasing order.

For this case t_j becomes j because for every time the condition is checked.

The total cost is

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^{n} j + C_5 \sum_{j=2}^{n} (j-1)$$

$$+ C_6 \sum_{j=2}^{n} (j-1) + C_7(n-1)$$

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n}{2}(n+1) - 1\right) + C_5\left(\frac{n}{2}(n-1)\right) + C_6\left(\frac{n}{2}(n-1)\right) + C_7(n-1)$$

$$= \left(\frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}\right)n^2 + \left(C_2 + C_3 + \frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2} + C_1 + C_7\right)n - (C_2 + C_3 + C_4 + C_6 + C_7)$$

$\boxed{T(n) = An^2 + Bn + C}$ → for worst case.

where $A = \frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}$

$B = C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} + C_7$

$C = -(C_2 + C_3 + C_4 + C_7)$

→ Average Case:- When only half of the comparisons are made.

$t_j = j/2$ (t_j will be the average of best & worst case which is nearly $j/2$)

The total cost will be the order of n^2 (i.e. quadratic)

→ Best Case:- $O(n)$

→ Worst case:- $O(n^2)$

→ Average case:- $O(n^2)$

* Merge sort and its analysis:-

→ Input to this function is array of integers and the indices to the first & last element of array.

→ MergeSort (A, p, r) ← index of last element of arr
array index of first element of arr.

```
{
  if (p < r)
  {
    q = ⌊(p+r)/2⌋   → Divide
    MergeSort(A, p, q) ;  } conquer
    MergeSort (A, q+1, r) ;
    Merge(A, p, q, r);  → Combine
  } else { return } ;
}
```

to get integer value as division can give decimal numbers

T(n) = Running time of merge sort on i/p size n.

∴ T(n) = Divide step cost + conquer cost + Cost of merge

= const + T(n/2) + T(n/2) + cost of merge.

$T(n) = 2T(n/2) + O(1) + O(n)$; $n > 1$

→ Constant is known as O(1)

$\therefore T(n) = 2T(n/2) + O(1) + O(n)$; $n > 1$

$\therefore T(n) = O(1)$; $n = 1$

↓ recurrence equation

For factorial ← also a recurrence equation

$f(n) = n * f(n-1)$; $n > 1$

$f(n) = 1$; $n = 1$

where, n is number to find factorial of it

```
Merge(A, p, q, r)
{
    n1 = q-p+1;   //(q-(p)+1)
    n2 =  r-q;    //(r-(q+1)+1)

    L[1...n1+1] and R[1...n2+1]
    L[n1+1] = ∞      // We add +1 to mark the end of
    R[n2+1] = ∞      array by infinity.
    for i=1 to n1
    {
        L[i] = A[p+i-1]
    }
    for i=1 to n2
    {
        R[i] = A[q+i]
    }
     i=1, j=1
    for k=p to r.
    {
        if(L[i] <= R[j])
        {       A[k] = L[i]
        }       i++;
        else
        {
            A[k] = R[j]
            j++;
        }
    }
}
```

→ Keep n = 8 and find the ans.

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

Let $\theta(n) + \theta(1) = Cn$

$T(8) = 2T(4) + 8C$

$= 2[2T(2) + 4C] + 8C$

$= 2[2[2T(1) + 2C]] + 8C$

$= 8T(1) + 8C + 8C + 8C$

$= 8C + 8C + 8C + 8C$

$T(8) = 32C$

$T(1) = O(1) = C$

→ The time complexity of merge sort is $n\log n + n$

→ There will be some constant C so it will be

$Cn\log n + Cn$

$= (8\log_2 8 + 8)C$

$= (8 \times 3 + 8)C$

$= 32C$

Out of $n\log n$ & n, $n\log n$ is bigger so comparisons made by merge sort is $n\log n$

→ $T(n) = aT\left(\frac{n}{b}\right) + D(n) + f(n)$

- a: number of sub problems
- $T\left(\frac{n}{b}\right)$: each of size (n/b)
- $D(n)$: cost of dividing problem of size n into sub problems of size (n/b)
- $f(n)$: cost of combining sol^n of sub problems to get sol^n of original problem

$a \geq 1$

$b > 1$

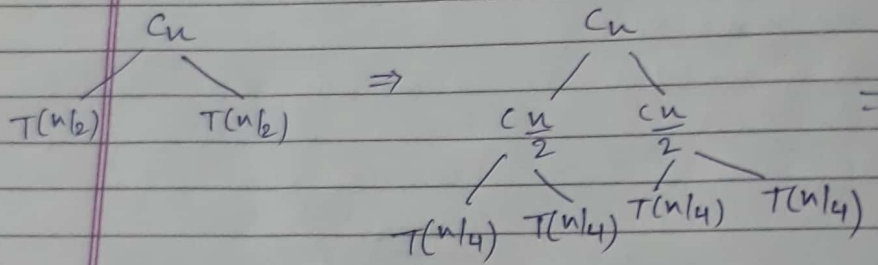
→ Above equation is the recurrence equation.

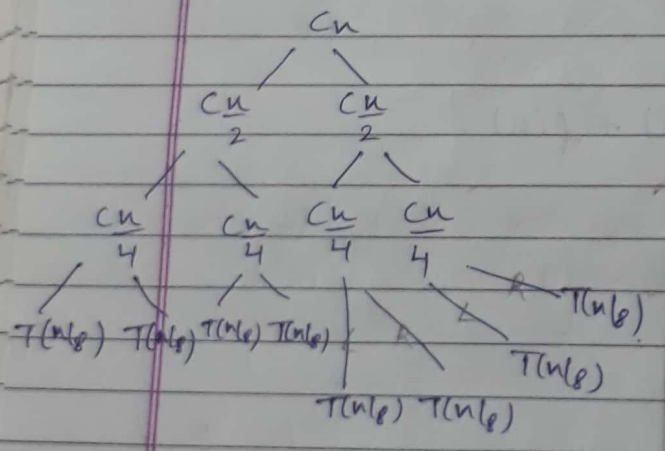
Q:- Solve the equation $T(n) = 2T(n/2) + Cn$ using recurrence tree / recursion tree method

→

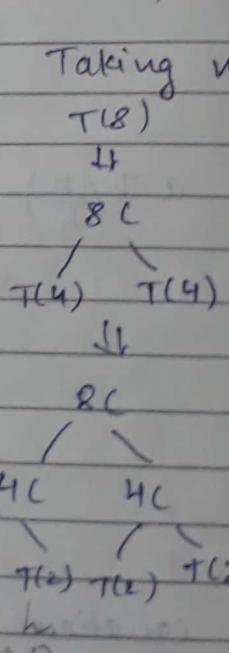
$T(n) \Rightarrow Cn$

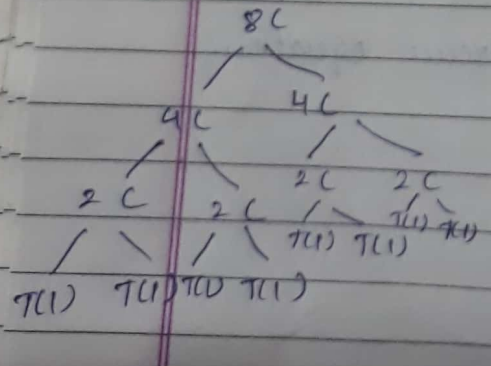
$T(n/2)$ $T(n/2)$ ← dividing the problem into two parts and then merging them requires a constant time Cn

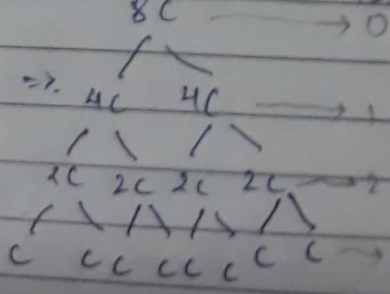




Taking $n = 8$.







$\Rightarrow 0$

$\Rightarrow 1$

$\Rightarrow 2$

$\Rightarrow 3$

$4 = 3 + 1$

$= \log_2 8 + 1$

Total

$i = 0$
$i = 1$
$i = 2$
$i = 3$

$i = \log n - 1$

$i = \log n$

using

Page No.
Date : / /

Total no. of levels = $\log_2 n + 1$

Height = $\log_2 n$

$\lg n = \log_2 n$

	# Nodes at level i	cost per node	cost at level i
i=0	1	c_n	c_n
i=1	2	$c_n/2$	c_n
i=2	4	$c_n/4$	c_n
i=3	8	$c_n/8$	c_n
	⋮	⋮	⋮
i=logn-1	$2^{\log n - 1}$	$c_n / 2^{\log n - 1}$	c_n
i=logn	$2^{\log n}$	$c_n / 2^{\log n} = \theta(1) = c$	c_n

Total cost :- $T(n) = c_n + c_n + c_n + \cdots$ (logn + 1) times

$\therefore T(n) = cn(\log n + 1)$

$\therefore T(n) = cn \log n + c_n = \theta(n \log n)$

Suppose at level m we get 1

$\therefore \frac{n}{2^m} = 1$

$\therefore n = 2^m$

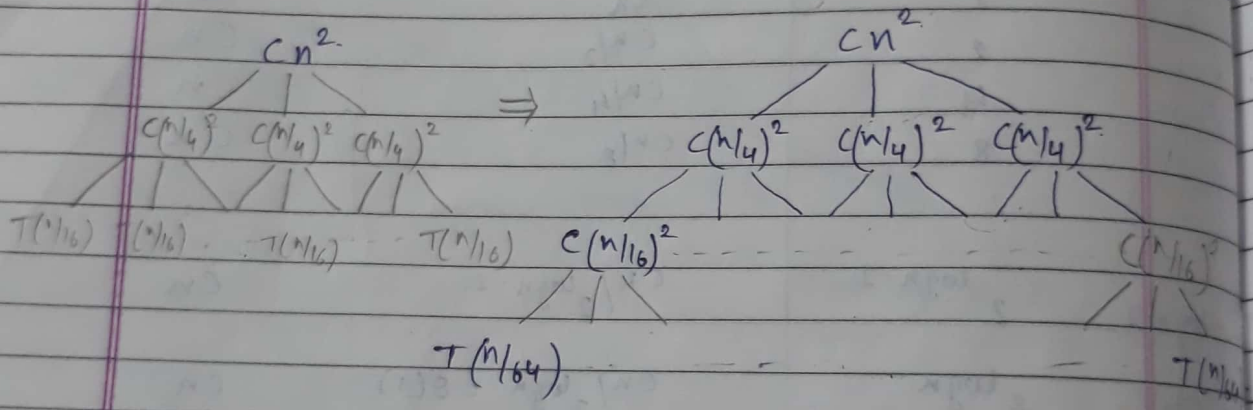
$\therefore m = \log_2 n$

Eg:- Assume the following recurrence relation for some algorithm.

$$T(n) = 3T(n/4) + cn^2 \quad ; n > 1$$
$$= \theta(1) \quad ; n = 1$$

→

$T(n)$: cn^2 ⇒ $T(n/4)$ $T(n/4)$ $T(n/4)$



	# Nodes at level i	Cost per node	Cost at level i
$i=0$	$3^0 = 1$	cn^2	cn^2
$i=1$	$3^1 = 3$	$c(n/4)^2$	$3c(n/4)^2$
$i=2$	$3^2 = 9$	$c(n/16)^2$	$9c(n/16)^2$
$i=3$	$3^3 = 27$	$c(n/64)^2$	$27c(n/64)^2$
$i = \log_4 n - 1$	$3^{\log_4 n - 1}$	$c\left(\frac{n}{\log_4^{n-1}}\right)^2$ → $c\left(\frac{n}{4^{\log_4 n - 1}}\right)^2$	$3^{\log_4 n - 1} c\left(\frac{n}{4^{\log_4 n - 1}}\right)^2$
$i = \log_4 n$	$3^{\log_4 n}$	$c\left(\frac{n}{\log_4^n}\right)^2$ → $c\left(\frac{n}{4^{\log_4 n}}\right)^2$	$3^{\log_4 n} c\left(\frac{n}{4^{\log_4 n}}\right)^2$
		$= \theta(1)$	$= 3^{\log_4 n} \cdot \theta(1)$

Total cost :-

$$T(n) = cn^2 + 3c\left(\frac{n^2}{4}\right) + 9c\left(\frac{n}{4^2}\right)^2 + 27c\left(\frac{n}{64}\right)^2 + \cdots + 3^{\log_4 n - 1} \cdot c\left(\frac{n}{4^{\log_4 n - 1}}\right)^2 + 3^{\log_4 n} \cdot O(1)$$

$$\therefore T(n) = cn^2\left[1 + \frac{3}{4^2} + \frac{9}{16^2} + \frac{27}{64^2} + \cdots + \frac{3^{\log_4 n - 1}}{(4^{\log_4 n - 1})^2}\right] + 3^{\log_4 n} \cdot c$$

$$= cn^2\left[1 + \frac{3}{4^2} + \left(\frac{3}{4^2}\right)^2 + \left(\frac{3}{4^2}\right)^3 + \cdots + \left(\frac{3}{4^2}\right)^{\log_4 n - 1}\right] + 3^{\log_4 n} \cdot c$$

Here $r = 3/16 < 1$

$$\leq cn^2\left[1 + \frac{3}{4^2} + \left(\frac{3}{4^2}\right)^2 + \cdots \infty\right] + c \cdot 3^{\log_4 n}$$

Assuming there are ∞ terms then

$$S_n = \frac{a}{1-r} \quad r < 1$$

$$\leq cn^2\left[\frac{1}{1 - \frac{3}{16}}\right] + c \cdot n^{\log_4 3}$$

$$\therefore T(n) \leq \frac{16}{13} cn^2 + c \cdot n^{\log_4 3}$$

The order of this algorithm is of $O(n^2)$

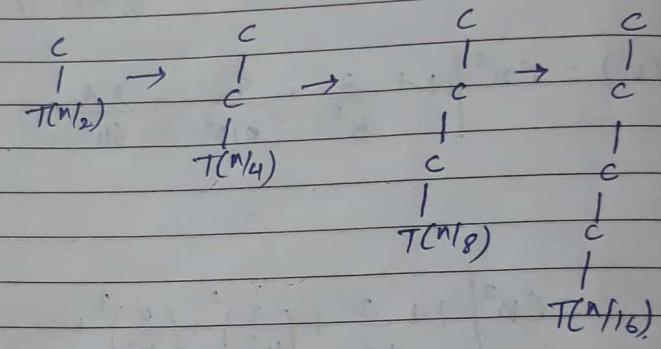
finding ... required ...

Eg:- Write recurrence equation for binary search and solve using tree method.

$$T(n) = T\left(\frac{n}{2}\right) + c \quad ; \quad n > 1$$

$$= c \quad ; \quad n = 1$$

$\Rightarrow$ T(n) $\Rightarrow$



Total cost:-

$T(n) = c + c + \text{---} + (\log n + 1)$ times

$\therefore T(n) = c(\log n + 1)$

$\therefore \boxed{T(n) = c \log n + c}$

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

* Notations:-

Big	Small.	Merge sort	Ins. sort
O	o	$c_1 n \log n$	$c_2 n^2$
Ω	ω		
Θ	θ		
$\downarrow$ more used			