Data Structure

1. Complexity

Handwritten by pankaj kumar

Complexity

Date
Page No. 01

- An algorithm is the step-by-step unambiguous instruction to solve a given problem.
- there are two main criteria to judging the merits of algorithms:
  - correctness :- does the algorithm give solution to the problem in a finite number of steps?
  - efficiency :- (how much resources in terms of memory and time does it take take to execute.

* Why the Analysis of Algorithms?

for a problem there are multiple algorithms are available for solving the same problem (for ex:- sorting algorithms are like insertion sort, selection sort, quicksort and many more) Algorithm analysis helps us to determine which algorithm is most efficient in terms of time and space consumed.

* Goal of Analysis of Algorithms?

- The goal of Analysis of Algorithms help us to determine ~~with~~ which algorithms is best in terms of running time and memory, developer effort etc.

* What is Running time analysis?

- It is the process of determining how processing time increases as the size of the problem (input size) increases.

* How to compare Algorithms?

- Execution time: Not a good measure as execution times are specific to a particular computer.
- Number of statements executed: Not a good measure, since the no. of statements varies with the programming languages as well as the style of the individual programmer.
- Ideal solution: Let us assume that we express the running time of a given algorithm as a function of input size n (i.e., $f(n)$) and compare these different function corresponding to running time. these should be independent of machine time, programming style etc.

✱ Rate of Growth:

- The rate at which the running time increases as a function of input

ex:- Total cost = cost of car + cost of bicycle

Total cost = cost of car (approximation)

ex:- $n^4+2n^2+100n+500 \approx n^4$

✱ Commonly used Rates of Growth:

Time		Name
1	—	Constant
logn	—	Logarithmic
n	—	Linear
nlogn	—	Linear logarithmic
n^2	—	Quadratic
n^3	—	Cubic
2^n	—	Exponential
n!	—	Factorial

✱ Types of Analysis:

- Worst case: Defines the I/p for which the algorithm takes a long time.
- Best case: Define the I/p for which the algorithm takes least time
- Average case: total running time and divide by no. of trials.

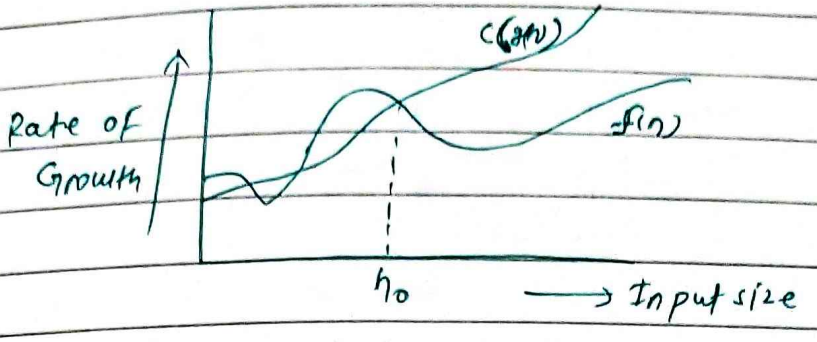
✱ Asymptotic Notation:

Having the expression for the best, average and worst cases, for all three cases we need to identify the upper and lower bounds. To represent there upper and lower bonds we use Asymptotic Notation

✱ Big-Oh Notation [Upper bound Function]:

- This notation gives the tight bound of the given function.
- Generally, it is represented as $f(n) = O(g(n))$. that means, at larger value of n, the upper bound of $f(n)$ is $g(n)$

ex:- if $f(n) = n^4 + 100n^2 + 10n + 50$ then n^4 is $g(n)$. That means $g(n)$ gives the maxm rate of growth for $f(n)$ at larger value



- O-notation defined as $O(g(n)) = \{ f(n)$: there exist positive constant c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0 \}$
- Our objective is to give the smallest rate of growth $g(n)$ which is greater than or equal to given algorithm's rate of growth $f(n)$

Example :- (i) upper bound for $f(n) = 3n+8$

$\therefore 3n+8 \le 4n$, for all $n \ge 8$

$\therefore 3n+8 = O(n)$ with $c=4$ and $n_0 = 8$

(ii) upper bound for $f(n) = n^2+1$

$n^2+1 \le 2n^2$ for all $n \ge 1$

$\therefore n^2+1 = O(n^2)$ with $c=2$ and $n_0 = 1$

(iii) upper bound for $f(n) = n^4+100n^2+50$

$n^4+100n^2+50 \le 2n^4$, for all $n \ge 11$

$\therefore n^4+100n^2+50 = O(n^4)$ with $c=2$ and $n_0 = 11$

(iv) upper bound for $f(n) = 2n^3 - 2n^2$

$2n^3 - 2n^2 \le 2n^3$ for all $n \ge 1$

$\therefore 2n^3 - 2n^2 = O(n^3)$ with $c=2$ and $n_0 = 1$

(v) upper bound for $f(n) = 410$

$410 \le 410$ for all $n \ge 1$

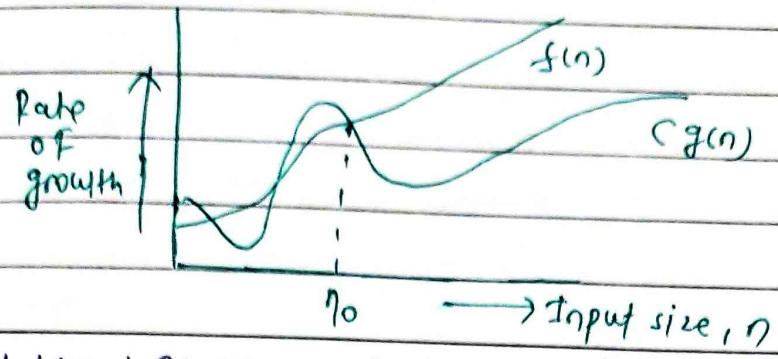
$\therefore 410 = O(1)$ with $c=1$ and $n_0 = 1$

(*) Omega-Ω Notation [lower-Bounding Function]:

- this notation gives the tighter lower bound of the given algorithm and we represent it as $f(n) = \Omega(g(n))$. that means, at larger

values of n, the tighter lower bound of f(n) is g(n)

- for $f(n) = 100n^2 + 10n + 50$, $g(n)$ is $\Omega(n^2)$



- Ω notation defined as $\Omega(g(n)) = \{f(n)$: there exist positive constant c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.
- Our objective is to give the largest rate of growth $g(n)$ which is less than or equal to given algorithm rate of growth $f(n)$.

Example:- ① Find lower bound for $f(n) = 5n^2$

i.e, $\exists\ c, n_0$ such that $0 \le cn^2 \le 5n^2$, $c = 5$ & $n_0 = 1$

$\therefore 5n^2 = \Omega(n^2)$ with $c = 5$ and $n_0 = 1$

② prove $f(n) = 100n + 5 \neq \Omega(n^2)$

$\exists\ c, n_0$ such that: $0 \le cn^2 \le 100n + 5$

$100n + 5 \le 100n + 5n\ (\forall\ n \ge 1) = 105n$

$cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$

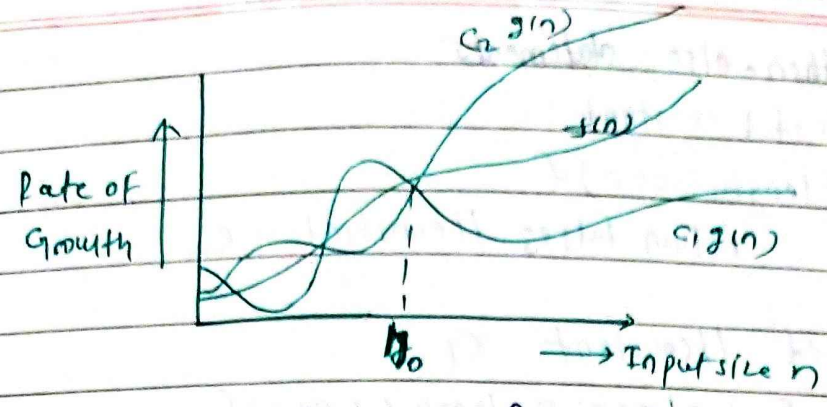
$\because n$ is +ve $\Rightarrow cn - 105 \le 0 \Rightarrow n \le 105/c$

$\because$ Contradiction : n cannot be ~~smaller than a constant~~.

③ $2n = \Omega(n)$, $n^3 = \Omega(n^3)$, $\log n = \Omega(\log n)$

(*) Theta - Θ Notation

- This notation decides whether the upper and lower bounds of a given function (algorithm) are the same
- If the upper and lower bound (O and Ω) give the same result, then the Θ notation will also have same rate of growth.



Θ defined as, $\Theta(g(n)) = \{f(n)$: there exist constant c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$. $g(n)$ is an asymptotic tight bound for $f(n)$.

Example:- Find Θ bound for $f(n) = \frac{n^2}{2} - \frac{n}{2}$

$\frac{n^2}{5} \le \frac{n^2}{2} - \frac{n}{2} \le n^2$ for all $n \ge 2$

$\therefore \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2)$ with $c_1 = \frac{1}{5}$, $c_2 = 1$ and $n_0 = 2$

(*) Guidlines for Asymptotic Analysis:

① loops: The running time of a loop is, at most, the running time of the statements inside the loop multiplied by the iterations.

ex:-
```
for(i=1; i<=n; i++)
    m=m+2; // constant time, c
```
Total time = $c \times n = cn = \boxed{O(n)}$

② Nested loops: product of sizes of all the loops-
```
for(i=1; i<=n; i++){
    for(j=1; j<=n; j++){
        k=k+1; // constant time
    }
}
```
Total time = $c \times n \times n = cn^2 = \boxed{O(n^2)}$

③ Consecutive statements: Add time complexities of each statement
```
for(i=1; i<=n; i++)
    m=m+2; // contant time
for(i=1; i<=n; i++){
    for(j=1; j<=n; j++){
        k=k+1 // constant time
    }
}
```
Total time = $c_1 n + c_2 n^2 = \boxed{O(n^2)}$

④ if-then-else statements

```
//test : constant
if(length()==0){
        return false; //constant c0
}
else{ //constant c1
    for(int n=0; n<length(); n++){
            //constant c2
            return false
    }
}
```

Total time = $c_0 + c_1 + c_2 n = \boxed{O(n)}$

⑤ logarithmic complexity

for(i=1; i<=n;)	for(i=n; i>=0;)
i=i*2;	i=i/2;

Sol: i = 1, 2, 4, 8, ... n

Let's loop is executing some k+1 times. At kth step $2^k = n$ taking logarithmic both side.

$\log(2^k) = \log n$

$\therefore k \log 2 = \log n$ or, $k = \log n$ $\therefore$ Total time = $\boxed{O(\log n)}$

✱ properties of asymptotic Notations:

- Transitivity: $f(n) = \theta(g(n))$ and $g(n) = \theta(h(n))$ then, $\boxed{f(n) = \theta(h(n))}$
- Reflexivity: $\boxed{f(n) = \theta(f(n))}$
- Symmetry: $f(n) = \theta(g(n))$ if and only if $\boxed{g(n) = \theta(f(n))}$
- Transpose symmetry: $f(n) = O(g(n))$ if and only if $\boxed{g(n) = \Omega(f(n))}$
- if $f(n)$ is in $O(kg(n))$: then $\boxed{f(n) \text{ is in } O(g(n))}$
- If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$, then $(f_1 + f_2)(n)$ is in $\boxed{O(\max(g_1(n), g_2(n)))}$

and $f_1(n) f_2(n)$ is in $\boxed{O(g_1(n) g_2(n))}$

Examples:—

① print("Hello") → 1
for (i=1; i<n; i++)
  print(i) → n
print("Bye") → 1

$T(n) = 1+1+n$
$= 2+n$
$\neq O(n)$

② i=1
while i<=n
Do
  print(i)
  i=i*2
Done

Iteration	1	2	3	4	5	...	k	k+1
variable i	1	2	4	8	16	...	n	
	2^0	2^1	2^2	2^3	2^4	...	2^{k-1}	2^k

$\therefore 2^k = n$, take log.
$\log_2 2^k = \log_2 n$
or, $k \log 2 = \log n$
$k = \log n$

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

③ i=n
while i>1
Do
  print(i)
  i=i/2
Done

Iteration	1	2	3	4	5	...	k	k+1
i=	n	n/2	n/4	n/8	n/16		$n/2^{k-1}$	1
	$n/2^0$	$n/2^1$	$n/2^2$	$n/2^3$	$n/2^4$		$n/2^{k-1}$	$n/2^k$

$\therefore n/2^k = 1$
or, $2^k = n$, same as above.
$k = \log n$

$T(n) = \log n$

④ i=1
while i<=n
Do
  print(i)
  i=i^2
Done

Iteration	1	2	3	4	...	k	k+1
i=	1	4	16	256		n	n
	2^{2^0}	2^{2^1}	2^{2^2}	2^{2^3}		$2^{2^{k-1}}$	2^{2^k}

$\therefore 2^{2^k} = n$
or, $\log_2 2^{2^k} = \log_2(n)$
or, $2^k \log_2 = \log_2(n)$
or, $2^k = \log n$ taking log both side
or, $\log_2 2^k = \log(\log n)$
$k = \log_2 \log_2 n$

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

(5) i=n
while i>1
Do
print(i)
$i = \sqrt{i}$
Done

Iteration	1	2	3	4	...	k	k+1
i=	n	$n^{1/2}$	$n^{1/4}$	$n^{1/8}$	...	$n^{\frac{1}{2^k}}$	

$n^{\frac{1}{2^0}}\ n^{\frac{1}{2^1}}\ n^{\frac{1}{2^2}}\ n^{\frac{1}{2^3}} \cdots n^{\frac{1}{2^{k-1}}}$

$\because n^{\left(\frac{1}{2^k}\right)} = 1$

taking log, $\log n^{(1/2^k)} = \log 1$

or, $\frac{1}{2^k}\log n = 0$

or, $\log 1 - \log 2^k + \log\log n = 0$

or, $k = \log_2 \log_2 n$

$T(n) = \log_2 \log_2 n$

* Independent Nested loops :-

(6) for (i=1; i<n; i++) ⟶ n
for (j=1; j<n; j=j*2) ⟶ log n
print (i, j)

$T(n) = O(A \cdot B) = T(n) = O(n \log n)$

(7) for(i=2; i<=n²; i=i²) — A
for(j=n/2; j<=n; j=j*2) — B
for(k=n²; k>=2; k=√k) — C

For A

Iteration	1	2	3	4	5	...	k	k+1
i=	2	4	6	8	10	...	n^2	.

2×1 2×2 2×3 2×4 2×5 ... 2×k 2×(k+1)

$\because 2 \times k = n^2$

or, $k = n^2/2$

$\therefore T(A) = O(n^2)$

For B

Iteration	1	2	3	4	5	----	k	k+1	k+2
i=	n/2	n	2n	4n	8n	----	n		

$2^{-1}\times n$ $2^{0}\times n$ $2^{1}\times n$ $2^{2}\times n$ $2^{3}\times n$ $2^{k-2}\times n$ $2^{k}\times n$

$2^k \cdot n = n$

$2^k = 1$

$\log_2 k = \log c$

$k = c$ → constant.

$T(B) = O(1)$

For C

Iteration	1	2	3	4	5	----	k	k+1	k+2
i=	n^2	n	$n^{1/2}$	$n^{1/4}$	$n^{1/8}$	----	2		

$n^{1/2^{-1}}$ $n^{1/2^{0}}$ $n^{1/2^{1}}$ $n^{1/2^{2}}$ $n^{1/2^{3}}$ $n^{1/2^{(k-2)}}$ $n^{1/2^{k}}$

$n^{1/2^k} = 2$

or, $\frac{1}{2^k}\log(n) = \log_2(2)$

or, $2^k = \log n$

or, $k = \log\log n$

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

∴ Total time = $O(n^2 \log\log(n))$

(8)

```
For(i=1; i<n; i++)
    For(j=1; j<n; j++)
        print(i,j)
```

Iteration	1	2	3	4	----	n
i=	1	2	3	4		n
j=	n	n	n	n		n

$T(8) = O(n^2)$

(9)

```
For(i=n; i>1; i= i-n/2)  → A
    For(j=1; j<=n^2; j=j+10)  → B
        For(k=n^10; k>1; k=k/2)  → C
            print(i,j,k)
```

For(A):

iteration	1	2	3			k
i=	n	n/2				

$T(n) = O(1)$

For(B):

iteration	1	2	3	4		k	k+1
j=	1	11	21	31			
						$1+10(k-1)$	$1+10(k)$

$1 + 10k = n^2$

$k = n^2$

$T(n) = O(n^2)$

For C:

Iteration	1	2	3	4		k	k+1
k=	n^{10}	$n^{10}/2$	$n^{10}/4$	$n^{10}/8$		1	1
						$\frac{n^{10}}{2^{k-1}}$	$\frac{n^{10}}{2^k}$

$\because \frac{n^{10}}{2^k} = 1$

or, $2^k = n^{10}$

or, $k \log 2 = 10 \log n$

or, $k = \log n$

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

Total Time: $O(n^2 \log n)$

* Dependent Nested loops:

10. For(i=1; i<=n; i++)
    For(j=1; j<=n; j=j+i)
    print(i, j)

Iteration	1	2	3	4		k	k+1
i	1	2	3	4		n	
j	n	n/2	n/3	n/4		n/n	→ $\leq n$

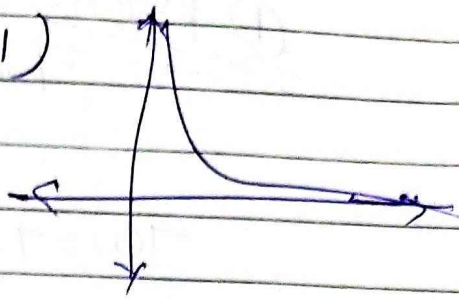
Iteration	1	2	3	4		k
j=	1	3	5	7		n

$1 + 2^{k-1} \leq n$ ∴ $k = ?$

$Total = n + n/2 + n/3 + n/4 + \cdots + n/n$

$= n(1 + 1/2 + 1/3 + 1/4 + \cdots + 1)$

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



(ii)

```
for(i=1; i<=n; i++)
    for(j=1; j<=i^2; j++)
        for(k=1; k<=n/2; k++)
            print(i,j,k)
```

dependent → $n/2$ → Independent

Iteration	1	2	3	4	k
i	1	2	3	4	n^2
j	1	4	9	16	n^2
k	$n/2$	$4 \cdot n/2$	$9 \cdot n/2$	$16 \cdot n/2$	$n^2 \cdot n/2$

$T(n) = n/2 + 4 \cdot n/2 + 9 \cdot n/2 + 16 \cdot n/2 + \cdots + n^2 \cdot n/2$

$= n/2 (1 + 4 + 9 + 16 + \cdots + n^2) = n/2 \cdot \left(\frac{n \cdot (n+1) \cdot (2n+1)}{6}\right)$

$= n/12 (2n^2 + 3n^2 + n)$

$= \frac{2n^4 + 3n^3 + n^2}{12} = n^4 + n^3 + n$

$\therefore T(n) = O(n^4)$

✱ Recursive Time Complexity :-

✱ Recursive Function :-

(1)

```
def fact(n):
    if n>1:
        return n*fact(n-1)
    return 1
```

$T(n)$ → $T(n-1)$

$T(n) = \begin{cases} T(n-1) & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$

✱ Back substitution:-

① problem:-

$$T(n) \rightarrow \begin{cases} T(n-1) & \text{if } n>1 \\ c & \text{if } n=1 \end{cases}$$

$T(n) = T(n-1)+c$ — (1)

$T(n-1) = T(n-2)+c$ — (2)

$T(n-2) = T(n-3)+c$ — (3)

$T(n-3) = T(n-4)+c$ — (4)

(2) in (1)

$T(n) = c+c+T(n-2) \Rightarrow 2c+T(n-2)$

put (3) For $T(n-2)$

$T(n) = c+c+c+T(n-3) \Rightarrow 3c+T(n-3)$

⋮

$$\boxed{T(n) = kc + T(n-k)}$$

$\because T(1) = c$

so, $(n-k)=1$

$k = n-1$

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

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

$$\boxed{T(n) = O(n)}$$

② problem:-

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

```
def Func(n):
    if n>1:
        For i=1 to n
            print(i)
        Func(n-1)
    return(1)
```

$T(n) = n + T(n-1) + c$ — (1)

$T(n-1) = n-1 + T(n-2) + c$ — (2)

$T(n-2) = n-2 + T(n-3) + c$ — (3)

$T(n-3) = n-3 + T(n-4) + c$

(2) in (1)

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

put (3) for $T(n-2)$

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

$\Rightarrow T(n) = kc - k + kn + T(n-k)$

$\Rightarrow \boxed{T(n) = kc' + kn + T(n-k)}$ — (5)

we know, $T(n) = c$ if $n \leq 1$

$T(n-k) = T(1)$

$n - k = 1$

$\boxed{k = n-1}$

substitute $k = n-1$ in (5)

$T(n) = (n-1)c' + (n-1)n + T(n-n+1)$

$= (n-1)c' + (n-1)n + T(1)$

$= nc' + n^2 - n + 1$

$\boxed{T(n) = \Theta(n^2)}$

(3) problem :-

```
def f(n):
    if n>1:
        return f(n/2)+f(n/2)
    return 1
```

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

$T(n) = c + 2T(n/2)$ —— (1)

$T(n/2) = c + 2T(n/4)$ —— (2)

$T(n/4) = c + 2T(n/8)$ —— (3)

(1) & (2)

↳ $T(n) = c + 2c + 4T(n/4) \Rightarrow 3c + 4T(n/4) \Rightarrow 3c + 4T\left(\frac{n}{2^2}\right)$

$T(n) = 3c + 4c + 8T(n/8) \Rightarrow 7c + 8T(n/8) \Rightarrow 7c + 8T\left(\frac{n}{2^3}\right)$

$T(n) = (2^k - 1)c + 2^k T\left(\frac{n}{2^k}\right)$ —— (4)

$\because T(n/2^k) = T(1)$ if $n = 1$

$\therefore \frac{n}{2^k} = 1 \Rightarrow 2^k = n$

$\therefore k = \log n$

* Put value of k in (4)

$T(n) = 2\log n \cdot c + 2^{\log n} T\left(\frac{n}{2^{\log n}}\right)$

or, $T(n) = 2\log n \cdot c + 2^{\log n} T(1)$

$T(n) = O(2\log n) = O(n)$

$x = 2^{\log n}$

$\log x = \log n \log 2$

$\log x = \log n$

$\therefore x = n$

(4) Problem :-

```
def fab(n):
    if n==1:
        return 0
    if n==2:
        return 1
    return fab(n-1) + fab(n-2)
```

$T(n-1)$ $T(n-2)$

$T(n) = \begin{cases} c & n = 1 \text{ or } n = 2 \\ c + T(n-1) + T(n-2) & \end{cases}$

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

Assume $T(n-1) \approx T(n-2)$

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

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

Date
Page No. 115

$T(n) = C + 2T(n-1)$ — (1)

$T(n-1) = C + 2T(n-2)$ — (2)

$T(n-2) = C + 2T(n-3)$ — (3)

$T(n-3) = C + 2T(n-4)$ — (4)

(1) + (2)

$\hookrightarrow T(n) = 3C + 4T(n-2)$

$T(n) = 7C + 8T(n-3)$

$T(n) = 15C + 16T(n-4)$

$T(n) = (2^k - 1) + 2^k T(n-k)$

$\because T(n-k) = T(1)$ if $n=1$

$n - k = 1$

or, $k = n-1$

$\therefore T(n) = 2^{(n-1)} - 1 + 2^{(n-1)} T(n-(n-1))$

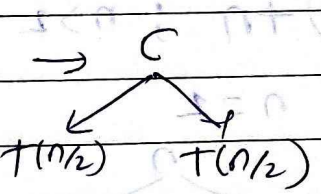
$= 2^{(n-1)} + 2^{(n-1)} - 1 + 1$

$\boxed{T(n) = O(2^n)}$

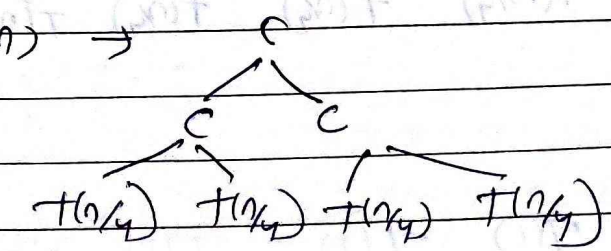
(2) Recursion Tree Method :-

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

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



- $T(n/2) = 2T(n/4) + C$



∴ $T(n) \rightarrow$ C → Total work → C

$T(n/2) \rightarrow$ C C → 2C

$T(n/4) \rightarrow$ C C C C → 4C

$T(n/8) \rightarrow T(n/8)\ T(n/8)\ T(n/8)\ T(n/8)\ T(n/8)\ T(n/8)\ T(n/8)\ T(n/8)$ → 8C

let after k-steps

$T(n/n) \rightarrow T(1)\ T(1)\ T(1)\ T(1)$ ---

$T(n/n) = T(1)$ or, $T(n/2^k) = T(1)$ or, $n = 2^k$, $k = \log n$

Total time $= C + 2C + 4C + \cdots + k \cdot C$

$= C + 2C + 4C + \cdots + 2^k \cdot C$

$= C(1 + 2 + 4 + 8 + \cdots + 2^k) \rightarrow$ G.P

$\frac{a \cdot r^n - 1}{r - 1}$; $a = 1, r = 2, n = k$

$\Rightarrow \frac{1 \cdot (2^k - 1)}{1} = 2^k - 1$

$= C \cdot 2^{\log n} = O(2^{\log n}) = \boxed{O(n)}$

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

C ; $n = 1$

Sol:- $T(n) \rightarrow n$ → total work → n

$T(n/2) \rightarrow n/2\ \ n/2$ → n

$T(n/4) \rightarrow T(n/4)\ T(n/4)\ T(n/4)\ T(n/4)$ → n ; $k \cdot n$

$T(n/8)$

after Estel

$T(n/n) \rightarrow T(1)\ T(1)\ T(1)$ ... → n

$T(n) = k \cdot n$
$= \log(n) \cdot n$
$\boxed{T(n) = n \log n}$

⇒ (3) Master's Theorem :-

Note :- $(\log n)^2 \neq \log^2 n$
$\log\log n$

$T(n) = a \cdot T(n/b) + O(n^k \log^p n)$
$a \geq 1, b \geq 1, k \geq 0$ and $p \in R$

(1) if $a > b^k$, Then $T(n) = O(n^{\log_b a})$

(2) if $a = b^k$,
- (a) if $p > -1$, Then $T(n) = O(n^{\log_b a} \cdot \log^{p+1} n)$
- (b) if $p = -1$, Then $T(n) = O(n^{\log_b a} \cdot \log\log(n))$
- (c) if $p < -1$, Then $T(n) = O(n^{\log_b a})$

(3) if $a < b^k$
- (a) $p \geq 0$ then, $T(n) = O(n^k \log^p n)$
- (b) if $p < 0$ then, $T(n) = O(n^k)$

Example :-

(1) $T(n) = 2T(n/2) + c$
$a = 2, b = 2, k = 0, p = 0$
$\therefore b^k = 1$, and $a > b^k$ i.e $2 > 1$
$\therefore T(n) = O(n^{\log_b a}) = O(n^{\log_2 2}) = \boxed{O(n)}$

(2) $T(n) = 2T(n/2) + n$
$a = 2, b = 2, k = 1, p = 0$
$\therefore b^k = 2^1 = 2$, i.e $a = b^k$ and $p > -1$.
$\therefore T(n) = O(n^{\log_b a} \cdot \log^{p+1} n) = O(n^{\log_2 2} \cdot \log^1 n) = O(n \log n)$

④ $T(n) = 16\,T(n/4) + n$

$a = 16,\ b = 4,\ k = 1,\ p = 0$

$\therefore b^k = 4,\quad a > b^k$

$\therefore T(n) = O(n^{\log_b a}) = O(n^{\log_4 16}) = O(n^2)$

⑤ $T(n) = 2T(n/2) + \frac{n}{\log(n)}$

$a = 2,\ b = 2,\ k = 1,\ p = -1$

$\therefore b^k = 2,\ a = b^k$

$\therefore T(n) = O(n^{\log_b a} \cdot \log\log(n))$

$= O(n^{\log_2 2} \cdot \log\log(n)) = O(n \log\log(n))$

⑥ $T(n) = 0.5\,T(n/2) + \frac{1}{n}$

$a = 0.5$ (x) cannot solve by master's theorem.

⑦ $T(n) = 7\,T(n/3) + n^2$

$a = 7,\ b = 3,\ k = 2,\ p = 0$

$\therefore b^k = 3^2 = 9,\ a < b^k$

$\therefore T(n) = O(n^k \log^p n) = O(n^2 \log^0 n) = O(n^2)$