Date : .....
Page : .....

Name:- Pravin Gupta
Std:- Bsc. CSIT (5th sem)
Subject:- Design and Analysis of Algorithms (DAA)
Campus:- Birendra Multiple Campus, Bharatpur chitwan

For videos

Youtube channel:- GUPTA TUTORIAL

For notes

Website:- www.pravingupta.com.np
Telegram channel:- GUPTA TUTORIAL

For any query

Facebook id:- Pravin Gupta (Anil)
Instagram id:- prabin.gupta.92

For Contribution Scan this one QR Code

Unit-1

Unit-1

Foundations of Algorithm Analysis

Date: .....
Page: .....

Algorithm :-

An algorithm can be defined as a well-defined computational procedure that takes some values, or the set of values, as an input and produces some values or the set of values as an output. An algorithm is thus a sequence of computational steps that transform the input into output.

Algorithms properties :-

(i) Correctness :- It should produce the output according to the requirement of the algorithm.

(ii) Fitness Finiteness :- Algorithm must complete after a finite number of instructions have been executed.

(iii) An Absence of Ambiguity :- Each step must be defined, having only one interpretation.

(iv) Definition of Sequence :- Each step must have a unique defined preceding and succeeding step. The first step and the last step must be noted.

(v) Input/output :- Number and classification of needed inputs and results must be stated.

(vi) Feasibility :- It must be feasible to execute each instruction.

(vii) Flexibility :- It should also be possible to make changes in the algorithm without putting so much effort on it.

Date :
Page :

Need of Algorithm

- To understand the basic idea of the problem.
- To find an approach to solve the problem.
- To improve the efficiency of existing techniques.
- To understand the basic principles of designing the algorithms.
- A good design can produce a good solution.
- To understand the flow of the problem.
- With the help of algorithm, we convert art into a science.
- To understand the principle of designing.

V.V.V. Vinay

RAM Model

Algorithms can be measured in a machine-independent way using the Random Access Machine (RAM) Model. This model assumes a single processor. In the RAM model, instructions are executed one after the other, with no concurrent operations. The assumptions made in the RAM model to accomplish this are:-

- Each simple operation (+, −, *, /) takes 1 time step.
- Loops and subroutines are not simple operations.
- Each memory access takes one time step and there is no shortage of memory.

For any given problem the running time of an algorithm is assumed to be the number of time steps. The space used by an algorithm is assumed to be the number of RAM memory cells. In computing time complexity, one good approach is to count primitive operations. This approach of simply counting primitive operations gives rise to a

computational model called the Random Access Machine (RAM)
The RAM model consists of following elements.

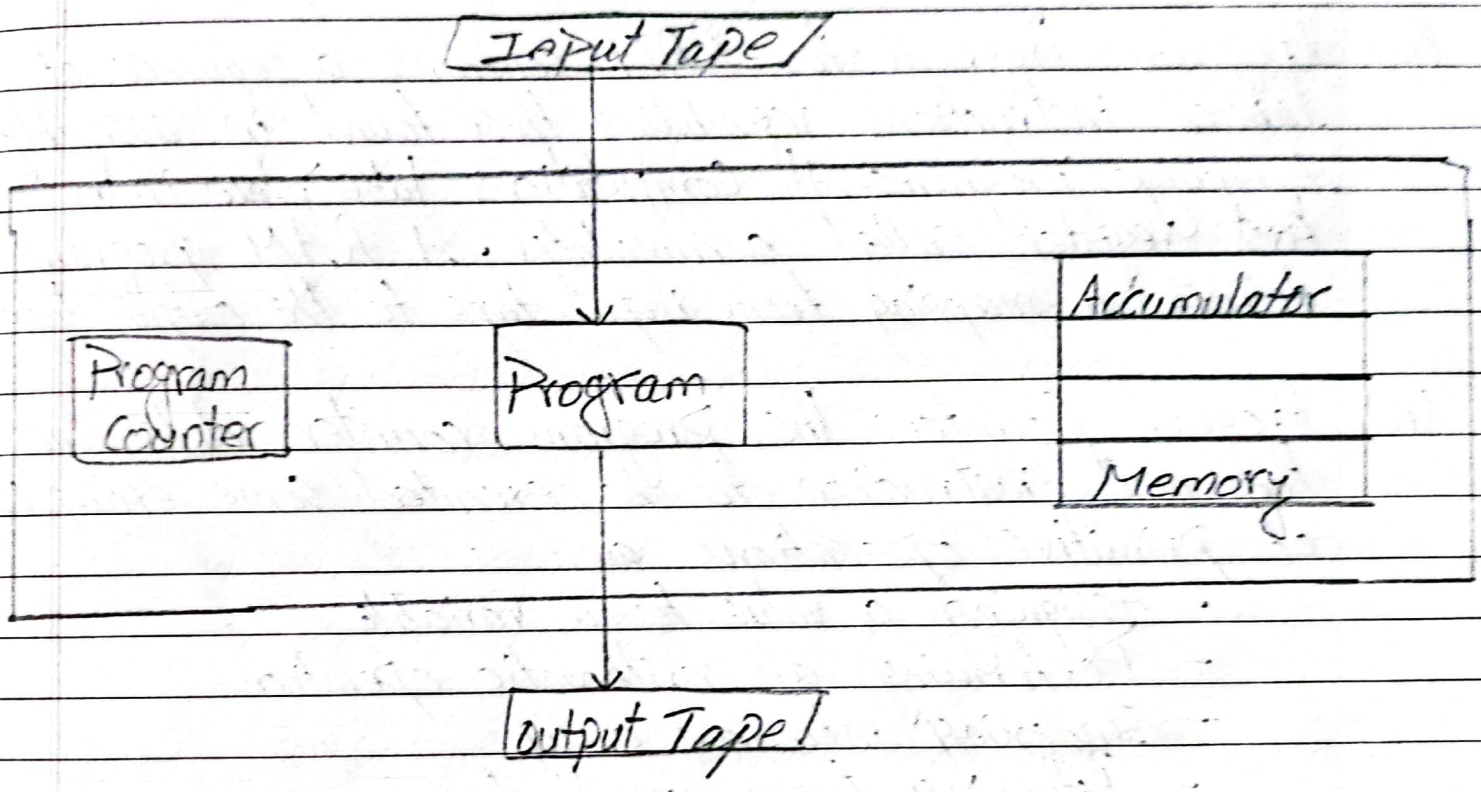


Fig:- RAM Model

(i) Input tape / output tape:- Input tape consists of a sequence of squares, each of which can store integer. Whenever one square is read from the tape head moves one square to the right. The output tape is also a sequence of squares, in each square an integer can be written. Output is written on the square under the tape head and the after the writing, the tape head moves one square to the right over writing on the same square is not permitted.

(ii) Memory:- The memory consists of a sequence of registers, each of which is capable of holding an integer.

(iii) Program:- Program for RAM contains a sequence of labeled instructions resembling those found in assembly language programs. All computations take place in the first register called accumulator. A RAM program defines a mapping from input tape to the output tape.

(iv) Program Counter:- The program counter determines the next instruction to be executed. Some examples of primitive operations are:-

- Assigning a value to a variable
- Performing an arithmetic operation
- Indexing into an array
- Returning from method
- Calling a method
- Comparing two numbers etc.

Example Algorithm to find the factorial of given numbers.

```
factorial (int n){
        if (n<0)
            fact = 1
            for (i=1; i<n; i++)
                fact = fact * i;
            return fact;
        }
```

Step count according to RAM Models: $7n+3$

Time and Space Complexity

Time complexity of an algorithm is the total time it takes to solve a problem. Time complexity is measured in terms of number of computational steps.

Space complexity of an algorithm is the total memory space it takes to store data requires to solve a problem. The space complexity is measured in terms of number of data variables used during the computation.

Time and space complexity depends on lots of things like hardware, operating system, processors etc. However, we don't consider any of these factors while analyzing the algorithm. The time complexity of an algorithm is commonly expressed using asymptotic notations.

- Big O ~ O(n);
- Big Theta ~ Θ(n)
- Big Omega ~ Ω(n)

Best case complexity:- It gives lower bound on the running time of the algorithm for any instance of input(s). This indicates that the algorithm can never have lower running time than best case for particular class of problems.

Worst case complexity:- It gives upper bound on the running time of the algorithm for all the instances of input(s). This indicates that number no input can overcome the running time limit posed by worst case complexity.

Average case Complexity:- It gives average number of steps required on any instance of the input(s).

Detailed Analysis of Algorithms:-

The process of finding complexity of given algorithm by counting number of steps and number of memory references used by using RAM model is called detailed analysis of algorithm.

For detailed time complexity of algorithm

→ Time complexity of simple operations that takes 1 step time like assignment (e.g. i=0), addition (e.g. a=b+c), simple statements like printf, scanf, return etc take very small constant time which does not affect time complexity of our algorithm much so we can neglect them.

→ We mainly analyze time complexity of algorithm based on the loops like for loop, while loop etc. We may have many condition in this case some of the simple cases are as follows:-

(i) If loop is like for(i=0, i<=n; i++) i.e loop is simply running from 0 to n and incrementing simply by 1. In this case time complexity will be O(n).

(ii) For nested loops eg two loops running simply as in (i) which are nested. In this case time complexity will be product of time complexity of each loop.

For e.g

```
for (i=0; i<=n; i++)        — O(n)
{
    for (j=0; j<=n; j++)    — O(n)
        printf("Hello");
}
```

Time complexity = $O(n) \times O(n)$
$= O(n^2)$

(iii) For loops like

```
for (i=0; i<=n; i*5)
```

i.e incrementing by multiplication. In this case time complexity = $O(\log_5 n)$ (constant multiplication)

$= O \log_5 n$

For detailed space complexity of Algorithm

space complexity is the total memory refrences used by the algorithm.

- If total ~~number~~ memory refrences used by the algorithm is constant like 1,2,3,4,5 etc then the space complexity will be $O(1)$.
- If Array is taking n memory refrences then the space complexity will be $O(n)$.

Example

Find detailed analysis of following factorial algorithm

```
#include<stdio.h>
int main()
{
    int i, n, fact=1;
    printf("Enter a number to calculate its factorial\n");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
        fact=fact*i;
    printf("factorial of %d=%d\n", n, fact);
    return 0;
}
```

Time Complexity

The declaration statement takes 1 step
printf statement takes 1 step
scanf statement take 1 step
In for loop,
i=1 takes 1 step
i<=n takes (n+1) step
i++ takes n step
within for loop fact=fact*i takes n step
printf statement takes 1 step
return statement takes 1 step

Total time complexity = 1+1+1+1+n+1+n+n+1+1
= 3n+7
= O(1) x O(n) + O(1)
= O(n) + O(1)
= O(n)

smallar terms are always neglected when there are higher ones like n, n^2 etc.

since time complexity of constant is O(1)

Space Complexity

Total memory refrences used = 3

(memory space is needed 1 for i, 1 for n and 1 for fact)

Hence, space complexity = O(1)

(For constant O(1))

Imp

Bubble Sort

```
Bubble_Sort(A,n)
{
    for(i=1; i<=n; i++)
    {
        for(j=0; j<n-i; j++)
        {
            if(A[j]>A[j+1])
            {
                temp=A[j];
                A[j]=A[j+1];
                A[j+1]=temp;
            }
        }
    }
}
```

~~Time Comp~~

Space complexity

1+1+1+1+n

= n+4 = O(n) + O(1) = O(n)

= O(n)

Time complexity

Within first for loop;

i=1 takes 1 step

i<=n takes (n+1) steps

i++ takes n steps

Within second for loop:

J=0 takes n step

J<n-i takes [n + (n-1) + (n-2) + ... + 2 + 1]

J++ takes [(n-1) + (n-2) + ... + 2 + 1]

In if statement:

It takes at most 3*(n-1)

So total time complexity (TC) = $1 + (n+1) + n + [n + (n-1) + (n-2) + \dots + 3 + 2 + 1] + [(n-1) + (n-2) + \dots + 3 + 2 + 1] + 3*[(n-1) + (n-2) + \dots + 3 + 2 + 1]$

$= 2n + 2 + \left[n + \frac{n(n+1)}{2} + \frac{n(n-1)}{2}\right] + 3*\frac{n(n-1)}{2}$

$= 2n + 2 + n + \frac{n^2}{2} + \frac{n}{2} + \frac{n^2}{2} - \frac{n}{2} + \frac{3n^2}{2} - \frac{3n}{2}$

$= \frac{5n^2}{2} + 3n - \frac{3n}{2} + 2$

$= \frac{(5n^2 + 3n)}{2} + 2$

$= \frac{O(1) \times O(n^2) + O(1) \times O(n) + O(1)}{O(1)}$

$= O(n^2) + O(n) + O(1)$

$= O(n^2).$

Sequential Search algorithm

```
Sequential_Search(A, n, key)
{
    int i, flag=0;
    for(i=0; i<n; i++)
    {
        if(A[i]==key)
        {
            flag=1;
        }
    }
    if(flag==1)
        print "Search successful"
    else
        print "Search unsuccessful"
}
```


Space complexity

The variable 'i' takes 1 unit space
The variable 'n' takes 1 unit space
The variable 'key' takes 1 unit space
The variable 'flag' takes 1 unit space
The variable 'A' takes n unit space

Now total space complexity is $1+1+1+1+n$

$= 4+n$

$= O(1) + O(n)$

$= O(n)$

Time Complexity

Declaration statement takes 1 step time.
In for loop;
i = 0 takes 1 step time
i<n takes (n+1) step time
i++ takes n step time
within for loop if condition takes n step time
within if statement flag = 0 takes at most n step time
If statement outside the for loop takes 1 step time
Print statement takes 1 step time
Now, total time complexity is

$$= 1+1+n+1+n+n+n+1+1$$
$$= 5+4n$$
$$= O(1) + O(1) \times O(n)$$
$$= O(1) + O(n)$$
$$= O(n)$$

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

Computational and Asymptotic Complexity

Whenever we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm. But when we calculate complexity of an algorithm it does not provide exact amount of resource required. So instead of taking exact amount of resource we represent that complexity in general form which produces the basic nature of that algorithm. We use that general form for analysis process.

Complexity analysis of an algorithm is very hard if we try to analyze exact. We know that the complexity (worst, best or average) of an algorithm is the mathematical function of the size of the input. So, if we analyze the algorithm in term of bound (upper and lower) then it would be easier. for this purpose, we need the concept of asymptotic notations. Asymptotic Notations is a way of comparing function that ignores constant factors and small input sizes. Three notations are used to calculate the running time complexity of an algorithm.

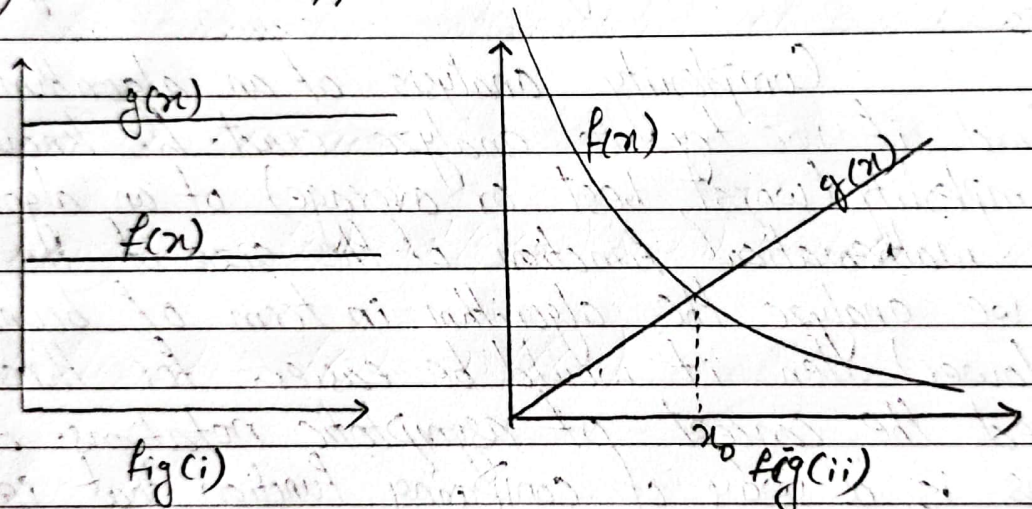
Why asymptotic notations important?

- They gives a simple characterization of an algorithm's efficiency.
- They allows the comparison of the performance of various algorithm

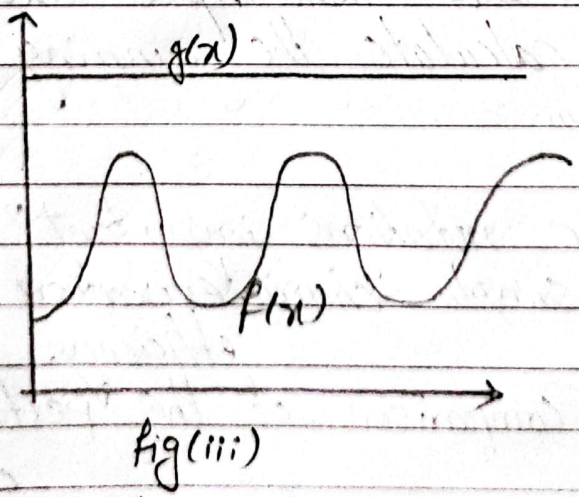
(i) Big Oh(O) notation

When we have only asymptotic upper bound then we use O notation. If f and g are any two functions from set of integers to set of integers then function f(n) is said to be big Oh of g(n) ie $f(n) = O(g(n))$ if and only if there exists two positive constants C and n_0 such that for all $n >= n_0$, $f(n) <= C * g(n)$

The above relation says that g(n) is an upper bound of f(n)



fig(i) fig(ii)



fig(iii)

Fig:- Geometric intrepretation of Big-Oh notation

Some properties

Transitivity :- $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then $f(n) = O(h(n))$

Reflexivity :- $f(n) = O(f(n))$

Transpose Symmetry :- $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

$O(1)$ is used to denote constants.

for all values of $n >= n_0$, plot shows clearly that $f(n)$ lies below or on the curve of $c*g(n)$

Ex: Find big Oh of given function $f(n) = 3n^2 + 4n + 7$

Sol - we have,

$$f(n) = 3n^2 + 4n + 7 <= 3n^2 + 4n^2 + 7n^2 <= 14n^2$$

$$f(n) <= 14n^2$$

where, $c = 14$ and $g(n) = n^2$,

thus

$$f(n) = O(g(n)) = O(n^2)$$

(ii) Big Omega (Ω) notation

Big omega notation gives asymptotic lower bound. If f and g are any two functions from set of integers to set of integers, then function $f(n)$ is said to be big omega of $g(n)$ i.e $f(n) = \Omega(g(n))$ if and only if there exists two positive constants c and n_0 such that for all $n >= n_0$, $f(n) >= c*g(n)$

The above relation says that g(n) is a lower bound of f(n)

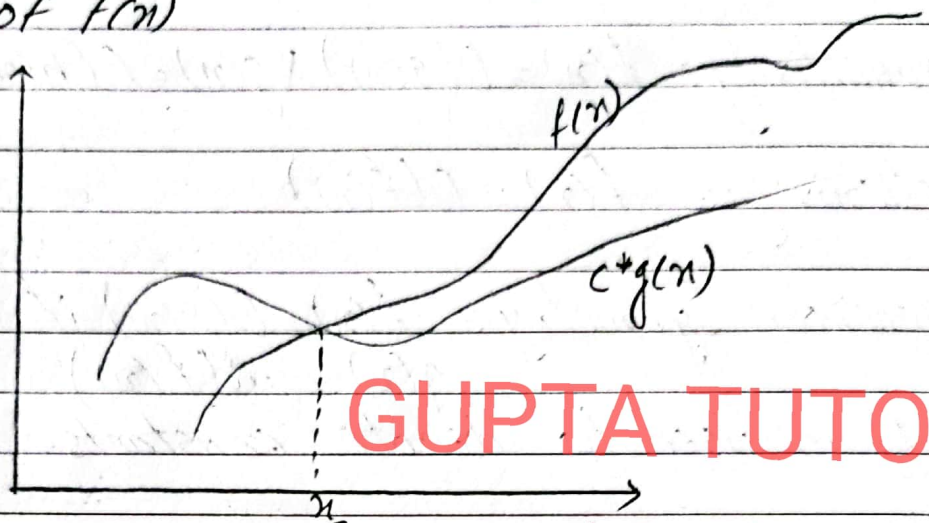


Fig:- Geometric interpretation of Big Omega notation

Some properties

Transitivity:- $f(n) = \Omega(g(n))$ & $g(n) = \Omega(h(n))$ then $f(n) = \Omega(h(n))$

Reflexivity:- $f(n) = \Omega(f(n))$

Example :-

Find big omega of $f(n) = 3n^2 + 4n + 7$

we have,

$$f(n) = 3n^2 + 4n + 7 >= 3n^2$$

$$\Rightarrow f(n) >= 3n^2$$

where, $c = 3$ and $g(n) = n^2$

thus, $f(n) = \Omega(g(n)) = \Omega(n^2)$

(iii) Big Theta (Θ) notation

When we need asymptotically tight bound then we use this notation. If f and g are any two functions from set of integers to set of integers then function f(n) is said to be big theta of g(n) i.e $f(n) = \Theta(g(n))$ if and only if there exists three positive constants c1, c2 and n_0 such that for all $n >= n_0$, $c1 * g(n) <= f(n) <= c2 * g(n)$

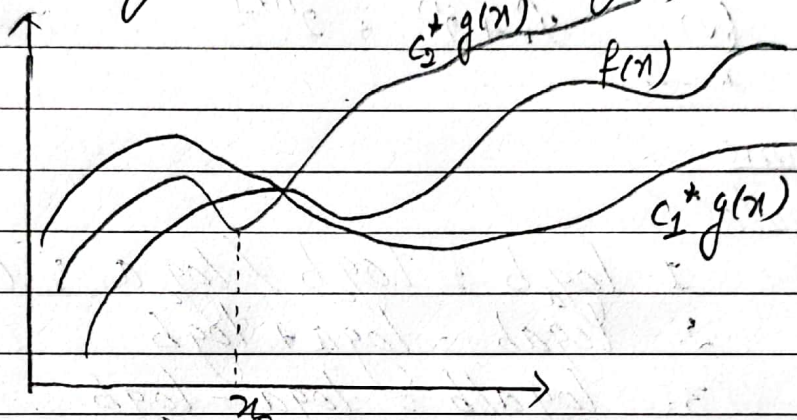


Fig:- Geometric interpretation of Big Theta notation

Some properties

Transitivity :- $f(n) = \Theta(g(n))$ & $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$

Reflexivity :- $f(n) = \Theta(f(n))$

Symmetry :- $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

e.g If $f(n) = 3n^2 + 4n + 7$, $g(n) = n^2$, then prove that $f(n) = \Theta(g(n))$

Proof :- let us choose c_1, c_2 and n_0 values are 14, 1 and 1 then

$f(n) <= c_1 * g(n)$, $n >= n_0$ as $3n^2 + 4n + 7 <= 14 * n^2$, and

$f(n) >= c_2 * g(n)$, $n >= n_0$ as $3n^2 + 4n + 7 >= 1 * n^2$

for all $n >= 1$ (in both cases)

so, $c_2 * g(n) <= f(n) <= c_1 * g(n)$ is trivial. Hence $f(n) = \Theta(g(n))$.

Mathematical foundation

Exponents Formulas

(i) $x^a \times x^b = x^{a+b}$

(ii) $x^a / x^b = x^{a-b}$

(iii) $(x^a)^b = x^{ab}$

(iv) $x^n + x^n = 2x^n$

(v) $2^n + 2^n + 2^{n+1}$

Logarithms Formulas

1. $\log_a b = \log_c b / \log_c a$; $c > 0$ (making base same)
2. $\log ab = \log a + \log b$
3. $\log a/b = \log a - \log b$
4. $\log(a^b) = b \log a$
5. $\log x < x$ for all $x > 0$
6. $\log 1 = 0$, $\log 2 = 1$, $\log 1024 = 10$
7. $a^{\log_b n} = n^{\log_b a}$

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} a^i \leq \frac{1}{1-a} \text{ ; if } 0 < a < 1$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Recurrence Relation

A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. To solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence. To solve recursive algorithms we need to define their recurrence relation and by using any one of the recurrence relation solving method we calculate their complexity.

Example

Recursive algorithm for finding factorial

$T(n) = 1$ when $n = 1$

$T(n) = T(n-1) + O(1)$ when $n > 1$

Recursive algorithm for finding N^{th} Fibonacci Number

$T(1) = 1$ when $n = 1$

$T(2) = 1$ when $n = 2$

$T(n) = T(n-1) + T(n-2) + O(1)$ when $n > 2$

Recursive algorithm for binary Search

$T(1) = 1$ when $n = 1$

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

Note

Cost of Solving recursive algorithm = cost of dividing problem + cost of solving subproblems + cost of merging solutions.

Solving Recurrences

The process of finding Solution of given recurrences relation in terms of big Oh notation is called Solving Recurrences. There are a lot of methods for Solving recurrences relations some of popular method are listed below:-

1. Iteration method
2. Recursion Tree (2078)
3. Substitution
4. Master Method. (2076)

1. Iteration Method.

Here we expand the given relation until the boundary is not meet. Expand the relation so that summation independent on n is obtained. The Iteration method is also known as the Iterative Method, Backward Substitution, Substitution method, and Iterative Substitution. It is a technique or procedure in computational mathematics used to solve a recurrence relation that uses an initial guess to generate a sequence of improving approximate solutions for a class of problems, in which the n^{th} approximation is derived from the previous ones.

Eg 1. Solve following recurrence relation by using iterative method

$T(n) = 2T(\frac{n}{2}) + 1$ when $n > 1$

$T(n) = 1$ when $n = 2$

Sol^n Given,

$$T(n) = 2T(\frac{n}{2}) + 1 \quad \text{..... (1)}$$

$$T(n) = 1$$

Put $n = \frac{n}{2}$ in eq^n (1)

$$T(\frac{n}{2}) = 2 \cdot T(\frac{n/2}{2}) + 1$$

$$\therefore T(\frac{n}{2}) = 2 \cdot T(\frac{n}{4}) + 1$$

Substute the value of $T(\frac{n}{2})$ in eq^n (1) then

$$T(n) = 2\{2 \cdot T(\frac{n}{4}) + 1\} + 1$$

$$T(n) = 2^2 \cdot T(\frac{n}{2^2}) + 2 + 1 \quad \text{---- (2)}$$

Again, Put $n = \frac{n}{2^2}$ in eq^n (1) then

$$T(\frac{n}{2^2}) = 2 \cdot T(\frac{n/2^2}{2}) + 1$$

$$T(\frac{n}{2^2}) = 2 \cdot T(\frac{n}{2^3}) + 1$$

Substute the value of $T(\frac{n}{2^2})$ in eq^n (2)

$$T(n) = 2^2\left\{2 \cdot T\left(\frac{n/2^2}{2}\right) + 1\right\} + 2 + 1$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 2^2 + 2 + 1$$

$$\therefore T(n) = 2^3 \cdot T\left(\frac{n}{2^3}\right) + 2^2 + 2^1 + 2^0$$

$$\vdots$$

$$T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + 2^{k-1} + \cdots + 2^2 + 2^1 + 2^0 \quad \text{---} (3)$$

For simplicity assume,

$$\frac{n}{2^k} = 1 \quad \text{---} (*)$$

$$n = 2^k \quad \text{---} (**)$$

Taking log on both sides

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

$$\log n = k \log 2 \qquad (\because \log(a^b) = b \log a)$$

$$\log n = k \times 1 \qquad (\because \log 2 = 1)$$

$$\therefore k = \log n$$

Then eq^n (3) becomes

$$T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + 2^{k-1} + \cdots + 2^2 + 2^1 + 2^0$$

eq^n (*)

$$T(n) = 2^k \times T(1) + 2^{k-1} + \cdots + 2^2 + 2^1 + 2^0$$

$$T(n) = \frac{(2^{k+1} - 1)}{(2-1)} \quad \left(\text{Induction Proof } 2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1\right)$$

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

$T(n) = 2^k \cdot 2 - 1$

$T(n) = 2 \cdot n - 1$ (from eq^n (1))

$T(n) = O(1) * O(n) - O(1)$

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

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

2 Solve following recurrence relation by using iterative method

$T(n) = T(n/3) + O(n)$ when $n > 1$

$T(n) = 1$ when $n = 1$

Sol^n Given,

$$T(n) = T\left(\frac{n}{3}\right) + O(n) \quad \text{---- (1)}$$

$$T(n) = T\left(\frac{n}{3}\right) + c \cdot n \quad \text{---- (I)}$$

$T(n) = 1$.

put $n = \frac{n}{3}$ in eq^n (1)

$$T\left(\frac{n}{3}\right) = T\left(\frac{n/3}{3}\right) + c \cdot \frac{n}{3}$$

$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{3^2}\right) + c \cdot \frac{n}{3}$$

Substitute the value of $T\left(\frac{n}{3}\right)$ in eq^n (I)

$$T(n) = T\left(\frac{n}{3^2}\right) + c \cdot \frac{n}{3} + c \cdot n \quad \text{---- (2)}$$

Again, $n = \frac{n}{3^2}$ in eq^n (1)

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

$$T\left(\frac{n}{3^2}\right) = T\left(\frac{n}{3^3}\right) + c \cdot \frac{n}{3^2}$$

Substitute the value of $T\left(\frac{n}{3^2}\right)$ in eq^n (2)

$$T(n) = T\left(\frac{n}{3^3}\right) + c \cdot \frac{n}{3^2} + c \cdot \frac{n}{3} + c \cdot n$$

$$\therefore T(n) = T\left(\frac{n}{3^3}\right) + c \cdot \frac{n}{3^2} + c\frac{n}{3} + c \cdot n \quad \cdots (3)$$

Again, put $n = \frac{n}{3^3}$ in eq^n (1)

$$T\left(\frac{n}{3^3}\right) = T\left(\frac{n/3^3}{3}\right) + c \cdot \frac{n}{3^3}$$

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

Substitute the value of $T\left(\frac{n}{3^3}\right)$ in eq^n (3)

$$\therefore T(n) = T\left(\frac{n}{3^4}\right) + c \cdot \frac{n}{3^3} + c \cdot \frac{n}{3^2} + c\frac{n}{3} + c \cdot n$$

$$T(n) = T\left(\frac{n}{3^k}\right) + c\frac{n}{3^{k-1}} + c \cdot \frac{n}{3^{k-2}} + \cdots + c\frac{n}{3^2} + \frac{cn}{3} + cn$$

Simplicity assume

$$\frac{n}{3^k} = 1$$

or, $n = 3^k$

or, Taking log on bothside

$$\log n = \log 3^k$$
$$\log n = k \log 3$$
$$k = \frac{\log n}{\log 3}$$
$$\therefore k = \log_3 n$$

Now, eq^n (4) becomes,

$$T(n) = T\left(\frac{n}{3^k}\right) + Cn + \frac{Cn}{3^{k-1}} + \cdots + \frac{Cn}{3^2} + \frac{Cn}{3} + Cn$$

$$T(n) = T(1) + \left\{ Cn + \frac{Cn}{3^{k-1}} + \cdots + \frac{Cn}{3^2} + \frac{Cn}{3^1} + \frac{Cn}{3^0} \right\}$$

$$T(n) = 1 + Cn\left\{ \frac{1}{3^{k-1}} + \cdots + \frac{1}{3^2} + \frac{1}{3^1} + \frac{1}{3^0} \right\}$$

(from question)

$$T(n) = 1 + Cn\left\{ \frac{1}{1-\frac{1}{3}} \right\}$$

(Formula $\frac{1}{1-\frac{1}{3}}$ if there is $\frac{1}{2}$ on power then $\frac{1}{1-\frac{1}{2}}$)

$$T(n) = 1 + Cn\,\frac{3}{2}$$

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

3. Solve following recurrence relation by using iterative method

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

Sol^n We have,

$$T(n) = T(n-1) + O(1)$$
$$T(n) = T(n-1) + 1 \quad \text{--- (1)} \quad [\text{since } O(1) = 1 \text{ where } c=1] \quad \text{--- (1)}$$

put $n = n-1$ in eq^n (1)

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

Substitute the value of $T(n-1)$ then

$$T(n) = T(n-2) + 1 + 1 \quad \text{--- (2)}$$

put $n = n-2$ in eq^n (1)

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

Substitute the value of $T(n-2)$ in eq^n (2) then

$$T(n) = T(n-3) + 1 + 1 + 1 \quad \text{--- (3)}$$

Again, put $n = n-3$ in eq^n (1)

$$T(n-3) = T(n-3-1) + 1$$
$$T(n-3) = T(n-4) + 1$$

Substitute the value of $T(n-3)$ in eq^n (3)

$$T(n) = T(n-4) + 1 + 1 + 1 + 1$$

$$T(n) = T(n-k) + 1 + \dots + 1 \text{ (k times)} \quad \cdots (4)$$

let choose $n-k = 1 \quad \cdots (5)$

$\Rightarrow k = n-1 \quad \cdots (5')$

Then eq^n (4) becomes

$$T(n) = T(1) + 1 + \dots + 1 \text{ (k times)} \quad (\text{From eq}^n (5))$$

$$T(n) = 1 + k * 1 \quad (\text{From eq}^n (5'))$$

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

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

$$T(n) = n$$

$$T(n) = O(n)$$

4. Solve following recurrence relation by using iterative method

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

Sol^n we have,

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

Put $n = \frac{n}{2}$ in eq^n (1)

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n/2}{2}\right) + \frac{n}{2}$$

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

Substute the value of $T\left(\frac{n}{2}\right)$ in eq^n (1)

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

$$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + n + n \quad \cdots \text{(2)}$$

put $n = \frac{n}{2^2}$ in eq (1)

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

$$T\left(\frac{n}{2^2}\right) = 2\,T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}$$

Substitute the value of $T\left(\frac{n}{2^2}\right)$ in eq (2)

$$T(n) = 2^2\left\{2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right\} + n + n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + n + n + \cdots + n \text{ (k times)} \quad \cdots \text{(3)}$$

put $\frac{n}{2^k} = 1 \quad \cdots (*)$

$$n = 2^k \quad \cdots (**)$$

Taking log on both side

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

$$\log n = k \log 2 \quad (\log 2 = 1)$$

$$\log n = k$$

Then eq (3) becomes

$$T(n) = n \cdot T(1) + n + n + \cdots + n \text{ (k times)}$$

$$= n + k * n$$

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

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

5. Solve following recurrence relation by using iterative method

$$T(n) = T(n/3) + n$$

Sol? Given,

$$T(n) = T\left(\frac{n}{3}\right) + n \quad \text{---- (1)}$$

put $n = \frac{n}{3}$ in eq? (1)

$$T\left(\frac{n}{3}\right) = T\left(\frac{n/3}{3}\right) + \frac{n}{3}$$

$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{3^2}\right) + \frac{n}{3}$$

then,

$$T(n) = T\left(\frac{n}{3^2}\right) + \frac{n}{3} + n \quad \text{---- (2)}$$

put $n = \frac{n}{3^2}$ in eq? (1)

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

$$T\left(\frac{n}{3^2}\right) = T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}$$

then,

$$T(n) = T\left(\frac{n}{3^3}\right) + \frac{n}{3^2} + \frac{n}{3} + n \quad \text{---- (3)}$$

Again,

put $n = \frac{n}{3^3}$ in eq? (1)

$$T\left(\frac{n}{3^3}\right) = T\left(\frac{n/3^3}{3}\right) + \frac{n}{3^3}$$

$$T\left(\frac{n}{3^3}\right) = T\left(\frac{n}{3^4}\right) + \frac{n}{3^3}$$

Then,

$$T(n) = T\left(\frac{n}{3^4}\right) + \frac{n}{3^3} + \frac{n}{3^2} + \frac{n}{3} + n \cdots$$

$$\vdots$$

$$T(n) = T\left(\frac{n}{3^k}\right) + \frac{n}{3^{k-1}} + \cdots + \frac{n}{3^2} + \frac{n}{3} + n \quad \text{--- (1)}$$

lets put $\frac{n}{3^k} = 1$

$$n = 3^k \quad \text{--- (**)}$$

Taking log on both sides

$$\log n = \log 3^k$$

$$\log n = k \log 3$$

$$k = \frac{\log n}{\log 3}$$

Then eq^n (1) becomes

$$T(n) = T(1) + \frac{n}{3^{k-1}} + \cdots + \frac{n}{3^2} + \frac{n}{3^1} + \frac{n}{3^0}$$

$$= 1 + n\left(\frac{1}{3^{k-1}} + \frac{1}{3^{k-2}} + \cdots + \frac{1}{3^2} + \frac{1}{3^1} + \frac{1}{3^0}\right)$$

$$= 1 + n\left(\frac{1}{3^0} + \frac{1}{3^1} + \frac{1}{3^2} + \cdots + \frac{1}{3^{k-2}} + \frac{1}{3^{k-1}}\right)$$

Since this is a geometric series of common ratio = 1/3
Thus,

$$T(n) = 1 + \frac{n\left(1 - (1/3)^k\right)}{1 - 1/3} \qquad \left[\because S_n = \frac{a(1-r^n)}{1-r}\right]$$

$$T(n) = 1 + \frac{n\left(1 - 1/3^k\right)}{2/3}$$

$$T(n) = 1 + \frac{3n}{2}\left(1 - \frac{1}{n}\right) \qquad \text{(from eq (1))}$$

$$T(n) = 1 + \frac{3n}{2}\left(\frac{n-1}{n}\right)$$

$$T(n) = 1 + \frac{3n^2}{2n} - \frac{3n}{2n}$$

$$T(n) = 1 + \frac{3n}{2} - \frac{3}{2}$$

$$T(n) = \frac{3n}{2} - \frac{1}{2}$$

$$T(n) = O(n)$$

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

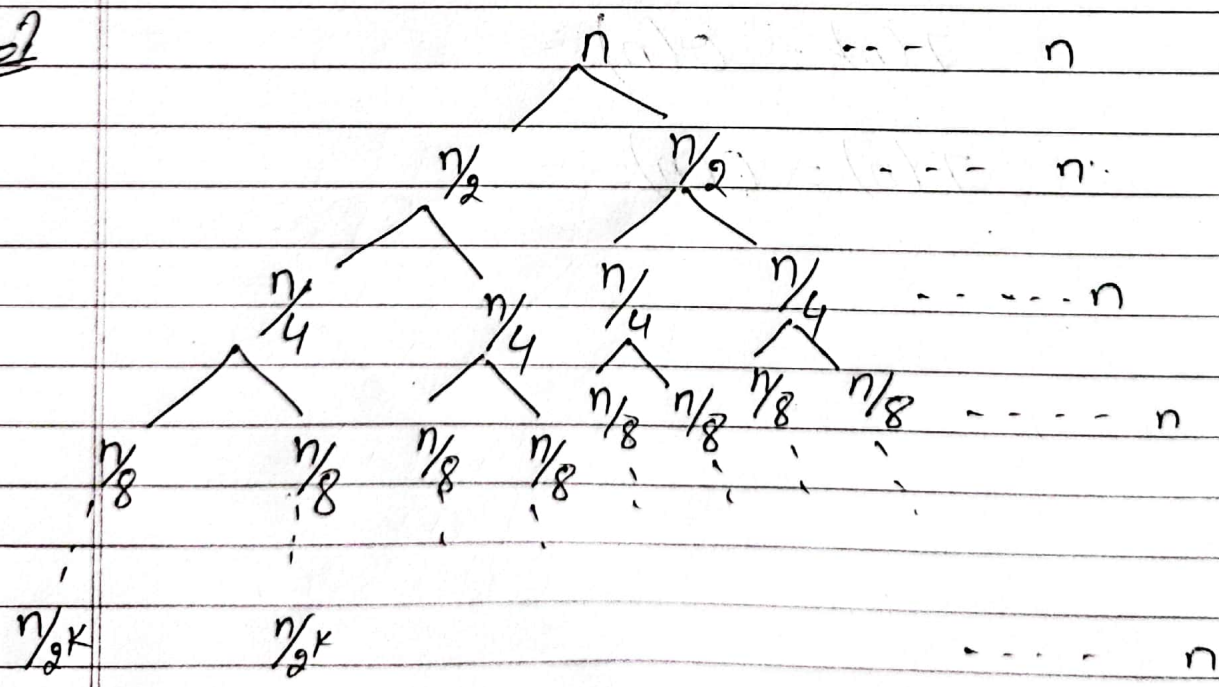
2. Recursion Tree :-

Recursion Tree Method is a pictorial representation of an iteration method which is in the form of a tree where at each level nodes are expanded. In Recursion tree, each root and child represents the cost of a single sub-problem. We sum the costs within each of the levels of the tree to obtain a set of pre-level costs and then sum all pre-level costs to determine the total cost of all levels of the recursion.

e.g ⊛ Solve following recurrence relation by using recursion tree method

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

Sol



Now,

$$T(n) = n + n + n + n + \dots + n \text{ (k times)} + T\left(\frac{n}{2^k}\right)$$

$$T(n) = n * k + T\left(\frac{n}{2^k}\right) \quad \dots (1)$$

For simplicity assume that

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

$$n = 2^k$$

Taking log on both sides

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

$$\log n = k \log 2$$

$$\log n = k \times 1$$

$$\therefore k = \log n$$

Now, eq^n (1) becomes,

$$T(n) = n * k + T(1)$$

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

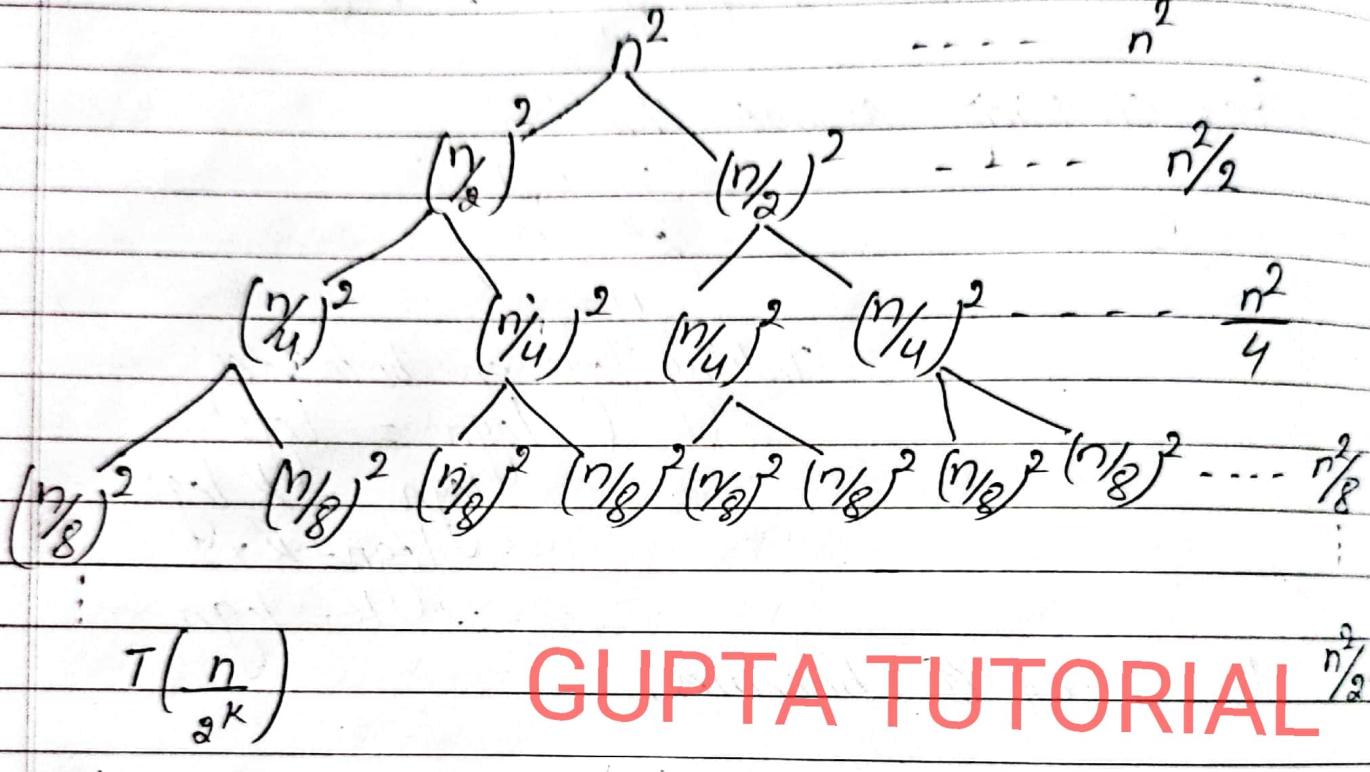
$$T(n) = O(n \log n) + O(1)$$

$$\therefore T(n) = O(n \log n) \text{ Ans}$$

Eg (Ex.) Solve following recurrence relation by using recursion tree method

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

Sol:



Now,

$$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \dots$$

$$= n^2\left[\left(\frac{1}{2}\right)^0 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \dots\right]$$

(GP)

$$= n^2\left[\frac{1}{1-1/2}\right]$$

$$= 2n^2$$

$$= O(2) \times O(n^2)$$

$$= O(n^2)$$

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

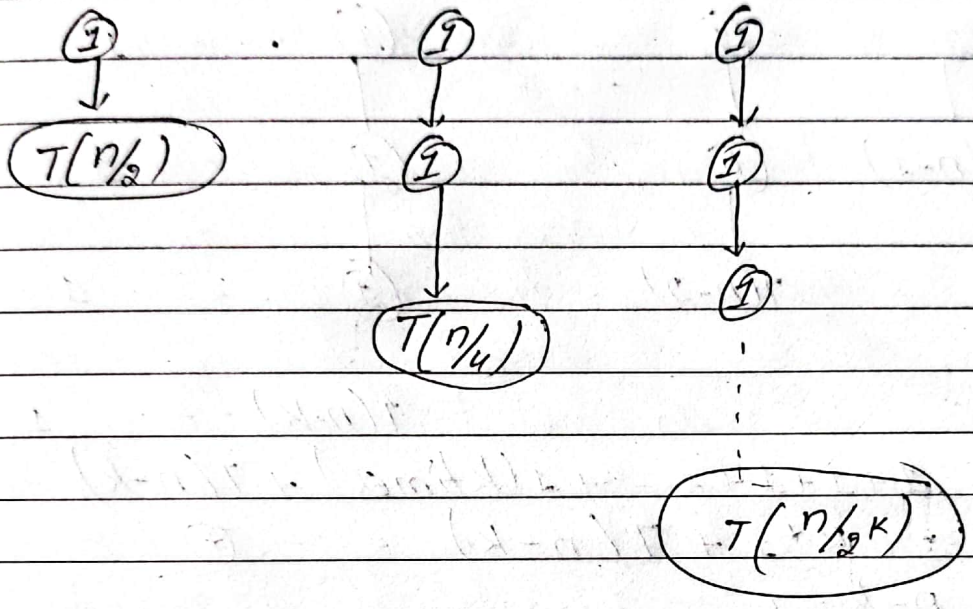
Ans

(*) Solve following recurrence relation by using recursion tree method

$T(1) = 1$ when $n=1$

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

Sol:



Cost at each level = 1

For simplicity assume that $n/2^k = 1$

$$n = 2^k$$

Taking log on both sides,

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

$$\log n = k \log 2$$

$$\log n = k$$

Summing the cost at each level,

Total cost = $1 + 1 + 1 + \dots + T(n/2^k)$

$= 1 + 1 + \dots + 1$ (k times) $+ T(1)$

$= k * 1 + 1$

$= (k+1)$

$= \log n + 1$

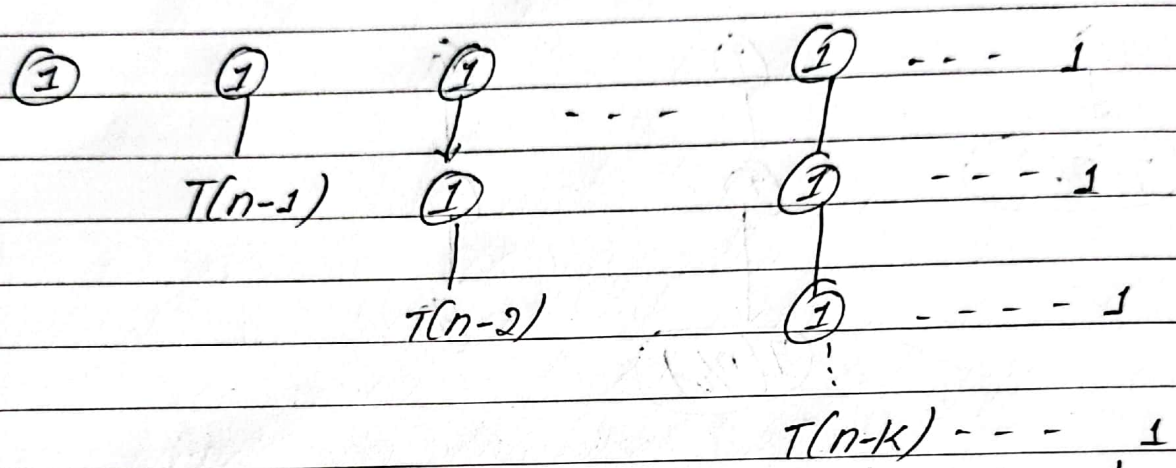
$O(\log n) + O(1) = O(\log n)$ Ans

① Solve following recurrence relation by using recursion tree method.

$T(1) = 1$ when $n=1$

$T(n) = T(n-1) + 1$ when $n>1$

Sol?



Total cost = $1+1+1+\dots+1$ (k times) $+ T(n-k)$

$= k*1 + T(n-k)$ --- ①

let's put $n-k=1$

$k = n-1$

then eq ① becomes

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

$= n-1+1$

$= n$

$= O(n)$

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

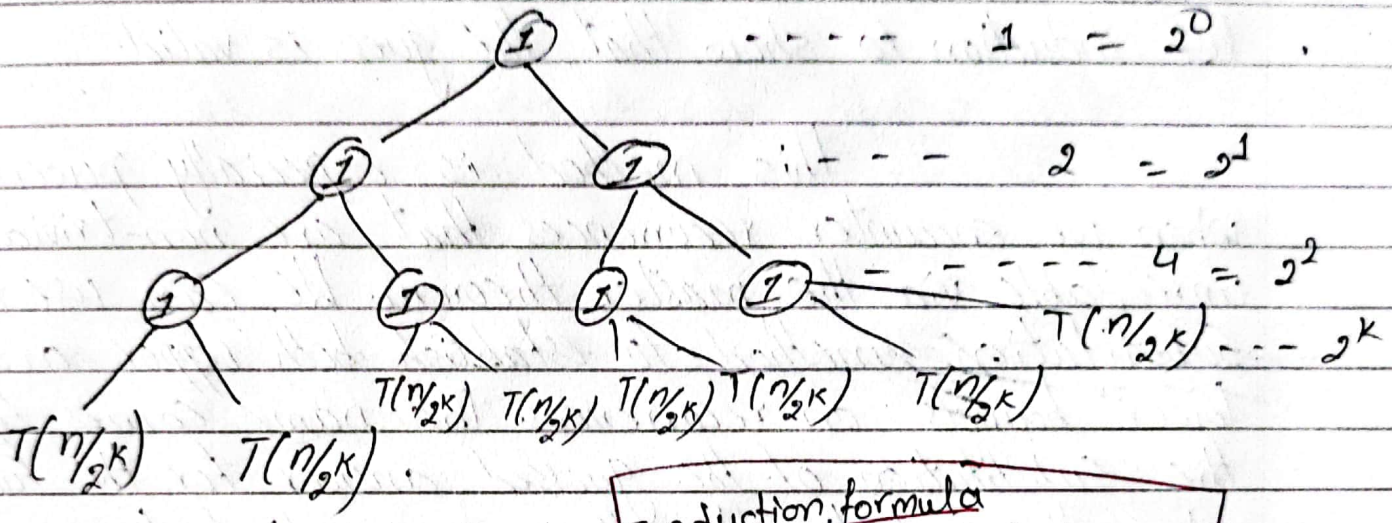
2018

(*) Solve following recurrence relation by using recursion tree method

$T(1) = 1$ when $n = 1$

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

Sol)



Induction formula

$2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1$

Now,

$T(n) = 2^0 + 2^1 + 2^2 + \cdots + 2^k$

$= 1 + 2(2^k - 1)$ (or) $\frac{(2^{k+1} - 1)}{2-1}$

$\frac{2-1}{}$

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

$= 2^* 2^k - 2 + 1$ $= 2 \cdot 2^k - 1$

$= 2^* 2^k - 1$ $= 2 * 2^k - 1$

For simplicity assume that $\frac{n}{2^k} = 1$

$n = 2^k$

Taking log on both sides

$\log n = \log 2^k$

$\log n = k \log 2$

$\log n = k$

Now, $T(n) = 2^* n - 1$

$= 2n - 1 \Rightarrow O(2) + O(n) - O(1) = O(n)$ Ans

3. **Substitution Method**

The substitution method for solving recurrences is famously described using two steps:

- Guess the form of the solution.
- Use induction to show that the guess is valid.

This method is especially powerful when we encounter recurrences that are non-trivial and unreadable via the master theorem. We can use the substitution method to establish both upper and lower bounds on recurrences. The name comes from the substitution of the guessed answer for the function when the inductive hypothesis is applied to smaller values. This method is powerful but it is only applicable to instances where the solutions can be guessed.

Example: Solve the following recurrence relation by using Substitution method

OR

Show the $O(n^3)$ the complexity of following recurrence relation by using Substitution method.

$$T(n) = 1 \qquad n = 1$$
$$T(n) = 4T(n/2) + n \qquad n > 1$$

Sol:

Guess,

$$T(n) = O(n^3)$$
$$\Rightarrow T(n) \leq cn^3, \text{ for all } n >= n_0 \text{ ----} (1)$$

Now, prove this by mathematical induction as

Basic step: For $n=1$

$T(n) \leq c * 1^3$ Definition

$1 \leq c$ which is true for all +ve value of c

Inductive steps: lets assume that it is true $\forall k < n$

then $T(k) \leq ck^3$ ---- (2)

It is also true for $k = \frac{n}{2}$

Now, eq^n (2) becomes

$T(n/2) \leq c(n/2)^3$

$\leq c\frac{n^3}{8}$

Now, $T(n) = 4T(n/2) + n$

$\leq 4\frac{cn^3}{8} + n$

$= cn^3/2 + n$

$= cn^3 - \frac{cn^3}{2} + n$

$= cn^3 - n(cn^2/2 - 1) \leq cn^3$

Hence, $T(n) \leq cn^3$ for $\forall n > 0$

Thus $T(n) = O(n^3)$

Proved ##

Example Show that $O(n^2)$ is the solution of following recurrence relation by using substitution method

$$T(n) = 1 \qquad n = 1$$
$$T(n) = 4T(n/2) + n \qquad n > 1$$

Sol Guess: $T(n) = O(n^2)$

$T(n) \leq cn^2$ for $\forall n > n_0$ ---- (1)

Now, proof this relation by using mathematical induction

Basic step :- For $n = 1$

$T(n) = c * 1^2$ Definition

$1 \leq c$ which is true for all +ve values of c

Inductive step :- Lets assume that it is true $\forall k < n$

Then $T(k) \leq ck^2$ ---- (2)

It is also true for $k = n/2$

Now eq (2) becomes,

$$T(n/2) \leq c(n/2)^2$$
$$= c \cdot n^2/4$$

Now,

$$T(n) = 4T(n/2) + n$$
$$= 4 \frac{cn^2}{4} + n$$
$$= cn^2 + n$$
$$\therefore T(n) = cn^2 + n$$

It is not possible to show that $cn^2 + n \leq cn^2 \ \forall n > 0$. Thus we try to subtract lower order term as.

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

$\Rightarrow T(n) \leq cn^2 - dn$ [since $cn^2 - dn \leq Bcn^2$] ---- (3)

where c and d are +ve constants

Now, proof this relation by using mathematical induction,

Basic step: for $n = 1$,

$T(n) = c \cdot 1^2 - d \cdot 1$ definition

$= 1 \leq c - d$ which is true for all +ve values of c and $d < c$

Inductive step:- lets assume that it is true $\forall k < n$

then $T(k) \leq ck^2 - dk$ ---- (4)

It is also true for $k = n/2$

Now eq^ (4) becomes

$T(n/2) \leq c(n/2)^2 - d(n/2)$

$= \frac{cn^2}{4} - dn/2$

Now,

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

$\leq 4\left[\frac{cn^2}{4} - \frac{dn}{2}\right] + n$

$\leq cn^2 - 2dn + n$

$\leq cn^2 - dn - dn + n$

$\leq (cn^2 - dn) + n(d-1)$

$\leq (cn^2 - dn)$

$\therefore T(n) < (cn^2 - dn) \ \forall \ n > 0$

Thus,

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

Proved

Example

Show that $O(n^3)$ is the solution of following recurrence relation by using substitution method

$$T(n) = 8T(n/2) + n^2$$

Sol?

Guess: $T(n) = O(n^3)$

$T(n) \leq cn^3$ for $\forall n > n_0$ ---- (1)

Now, proof this relation by using mathematical induction

Basic step: for $n = 1$

$T(n) = c * 1^3$ Definition

$1 \leq c$ which is true for all +ve values of c

Inductive step:- lets assume that it is true $\forall k < n$

then $T(k) \leq ck^3$ ---- (2)

It is also true for $k = n/2$

Now, eq^n (2) becomes,

$$T(n/2) \leq c(n/2)^3$$

$$= \frac{cn^3}{8}$$

Now, $T(n) = 8T(n/2) + n^2$

$$= 8\left(\frac{n^3}{8}\right) + n^2$$

$$= cn^3 + n^2$$

$\therefore T(n) = cn^3 + n^2$

It is not possible to show that $cn^3 + n^2 \leq cn^3$ $\forall n > 0$, thus we try to substract lower order term as,

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

$\Rightarrow T(n) \leq cn^3 - dn^2$ [since $cn^3 - dn^2 \leq cn^3$] ---- (3)

where c and d are +ve constant.

Now, proof this relation by using mathematical induction.

Basic step: for $n = 1$

$T(n) \leq c \cdot 1^3 - d \cdot 1$ Definition

$1 \leq c - d$ which is true for all +ve value of c and d ...

Inductive step: lets assume that it is true $\forall k < n$

then, $T(k) \leq ck^3 - dk^2$ .... (i)

It is also true for $k = n/2$

Now, eq^n (4) becomes

$$T(n/2) \leq c(n/2)^3 - d(n/2)^2$$
$$\leq c\frac{n^3}{8} - d\frac{n^2}{4}$$

Now,

$$T(n) \leq 8T(n/2) + n^2$$
$$\leq 8\left[c\frac{n^3}{8} - d\frac{n^2}{4}\right] + n^2$$
$$\leq cn^3 - 2dn^2 + n^2$$
$$\leq cn^3 - dn^2 - dn^2 + n^2$$
$$\leq (cn^3 - dn^2) - n(dn - 1) \leq (cn^3 - dn^2)$$
$$\Rightarrow T(n) \leq (cn^3 - dn^2) \quad \forall \; n > 0$$

Thus, $T(n) = O(n^3)$.

Proved

4. **Master Method**

Master Method is a direct way to get the solution. The master method works only for following types of recurrences or for recurrences that can be transformed to following type.

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

where, $a \geq 1$, $b > 1$ are constant, $f(n)$ asymptotically positive function

If the recurrence relations is in this form then there are following four possible cases occurred.

Master Method Case 1

If $f(n) = O(n^{\log_b a - \epsilon})$ for constant $\epsilon > 0$ (For greater)

Then,

$$T(n) = \theta(n^{\log_b a})$$

Master Method Case 2

If $f(n) = n(n^{\log_b a + \epsilon})$ for constants $\epsilon > 0$ (For smaller)

Then,

$$T(n) = \theta(f(n))$$

Master Method Case 3

If $f(n) = \theta(n^{\log_b a})$ for constants $\epsilon > 0$ (for equal)

Then,

$$T(n) = \theta(f(n) \cdot \log n)$$

In the above three cases we are comparing the values of $f(n)$ and $n^{\log_b a}$ and then find complexity of the given recurrence relation

Master Method Case 4

In this case the master method cannot be applied.

Example

1. Solve the following recurrence relation by using Master's method

$$T(n) = 3T\left(\frac{n}{2}\right) + n$$

Sol) Given,

$$T(n) = 3T\left(\frac{n}{2}\right) + n$$ is comparing with

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

$\therefore a = 3,\ b = 2,\ f(n) = n$

Formula $\left(\log_a b = \frac{\log b}{\log a}\right)$

Now, $n^{\log_b a} = n^{\log_2 3} = n^{\frac{\log_a 3}{\log_a 2}} = n^{1.584}$

Now, $n^{1.584} > f(n)$ so, we apply theorem 1

$f(n) = O(n^{\log_b a - \varepsilon})$ where choose $\varepsilon = 0.584$

Thus, it's complexity

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3}) = \Theta(n^{1.584})$$

Thus, $T(n) = \Theta(n^{1.584})$

2. Solve the following recurrence relation by using Master's method

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

Sol? Given,

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$ is comparing

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

$\therefore a = 9, \quad b = 3, \quad f(n) = n$

Now,

$$n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2} = n^{2\log_3 3} = n^2$$

Since,

$n^2 > f(n)$ so,

$f(n) \leq n^{\log_b a - \epsilon}$ where $\epsilon = 1$

Thus it satisfy the first case of Master's method

Thus it's complexity

$$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_3 9}) = \theta(n^2)$$

Thus;

$$T(n) = \theta(n^2)$$ Ans

3. Solve the following recurrence relation by using Master's method

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

Sol? Given,

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

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

$\therefore a = 4, \quad b = 2, \quad f(n) = n^2$

Now,

$$n^{\log_b a} = n^{\log_2 4} = n^{\log_2 2^2} = n^{2\log_2 2} = n^2$$

Since, $f(n) = n^{\log_b a}$

so, we apply third case of Master's method

Thus it's complexity,

$$T(n) = \Theta(f(n)\log n) = \Theta(n^2 \log n)$$

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

4. Solve the following recurrence relation by using Master's method

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

Sol) Given,

$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$ is comparing with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$\therefore a = 3, \quad b = 4, \quad f(n) = n\log n$

Now,

$$n^{\log_b a} = n^{\log_4 3} = n^{\frac{\log 3}{\log 4}} = n^{0.792}$$

Since,

$f(n) \geq n^{\log_b a + \varepsilon}$ where choose $\varepsilon = 0.208$

Thus it satisfy the second case of Master's method

Thus it's complexity,

$$T(n) = \Theta(f(n)) = \Theta(n\log n)$$

thus, $T(n) = \Theta(n\log n)$ Ans

5. Solve the following recurrence relation by using Master's method

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

Sol? Given,

$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$ is comparing with

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

$\therefore a = 2, \quad b = 4, \quad f(n) = \sqrt{n} = n^{1/2} = n^{0.5}$

Now,

$$n^{\log_b a} = n^{\log_4 2} = n^{\frac{\log 2}{\log 4}} = n^{0.5}$$

Since,

$f(n) = n^{\log_b a}$

Thus it satisfy the third case of Master's method

it's complexity,

$$T(n) = \theta(f(n)\log n) = \theta(n^{0.5}\log n)$$

Thus,

$$T(n) = \theta(n^{0.5}\log n)$$ Ans

6. Solve the following recurrence relation by using Master's method

$$T(n) = 2T\left(\frac{2n}{3}\right) + 1$$

Sol? AT First Convert this relation into Master's form as

$$T(n) = 2T\left(\frac{2n}{3}\right) + 1$$

$$T(n) = 2T\left(\frac{n}{3/2}\right) + 1 \cdot n^0.$$

$T(n) = 2T\left(\frac{n}{1.5}\right) + n^0$ Now comparing

$a = 2,\ b = 1.5,\ f(n) = n^0$

Now,

$$n^{\log_b a} = n^{\log_{1.5} 2} = n^{\frac{\log 2}{\log 1.5}} = n^{1.709}$$

Since,

$$f(n) \leq n^{\log_b a}$$

Thus, we apply 1st case of Master's method

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1.709})$$

$$\therefore T(n) = \Theta(n^{1.709})$$

Eg 7 Solve following recurrence relation by using Master method.

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

Sol Given,

$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\log n}$$ is comparing with

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

$\therefore a = 4,\ b = 2,\ f(n) = \frac{n^2}{\log n}$

Now,

$$n^{\log_b a} = n^{\log_2 4} = n^{\log_2 2^2} = n^{2\log_2 2} = n^2$$

Test case 1

$$f(n) = O(n^{\log_b a - \epsilon})$$

$$\Rightarrow f(n) \leq n^{\log_b a - \epsilon}$$

or, $f(n) \leq n^{2-\epsilon}$

or, $\frac{n^2}{\log n} \leq \frac{n^2}{n^\epsilon}$

or, $\frac{n^2}{\log n} \leq \frac{n^2}{n^{0.1}}$ where we choose $\epsilon = 0.1$

$\Rightarrow n^{n^{0.1}} \leq n^2 \log n$

To satisfy this relation the value of $\log n$ must be greater than $n^{0.1}$

But $n^{0.1}$ is a polynomial in 0.1 thus $n^{0.1}$ must be greater than $\log n$

i.e $n^{0.1} > \log n$

Thus master method failed in this case

Test case 2: $f(n) = n(n^{\log_b a + \epsilon})$

$\Rightarrow f(n) \geq n^{\log_b a + \epsilon}$

or, $\frac{n^2}{\log n} \geq n^{2+\epsilon}$

or, $\frac{n^2}{\log n} \geq n^{2+0.1}$

$\therefore n^2 \geq n^2 n^{0.1} \log n$ which is false

Test case 3: $f(n) = \theta(n^{\log_b a})$

$\Rightarrow f(n) = n^{\log_b a}$

$\frac{n^2}{\log n} = n^2$

$n^2 = n^2 \log n$ which is false

Since master method is false in all three case thus, in this recurrence relation master method cannot be applied.

2076

5. Solve the following recurrence relation using master's method

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

Sol?

Given,

$T(n) = 7T\left(\frac{n}{2}\right) + n^2$ is comparing with

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

$\therefore a = 7, \quad b = 2, \quad f(n) = n^2$

Now,

$$n^{\log_b a} = n^{\log_2 7} = n^{\frac{\log 7}{\log 2}} = n^{2.807}$$

Since,

$f(n) \leq n^{\log_b a}$ where choose $\epsilon = 0.807$

Thus, it satisfy the first case of Master's Method

Thus, it's complexity

$$T(n) = \theta(n^{\log_b a}) = \theta(n^{2.807})$$

$\therefore T(n) = \theta(n^{2.807})$ Ans

(b) $T(n) = 4T\left(\frac{n}{4}\right) + kn$

Sol?

Given,

$T(n) = 4T\left(\frac{n}{4}\right) + kn$ is comparing with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$\therefore a = 4, \quad b = 4, \quad f(n) = kn^1$

Now,

$$n^{\log_b a} = n^{\log_4 4} = n^1$$

Since, $f(n) = n^{\log_b a}$

Thus, it satisfy the third case of Master's method

it's complexity

$$T(n) = \theta(f(n) \log n) = \theta(kn \log n)$$

$$\therefore T(n) = \theta(kn \log n)$$ Ans