3. Towers of Hanoi

4. Fibinocci series.

"Recursion":-

Ex:- write a 'C' Program to Print Array of 'n' numbers.

A {10, 20, 30, 40, 50}
1 2 3 4 5

```
main ()
{
    Print_Array (a[], i, J);
}
Print_Array (a[], i, J)
{
    If (i==J)
        Pf (a[i]);
    else
    {
        Printf (a[i])
        Print_Array (a[], i+1, J);
    }
}
```

⇒ Tail Recursion :-

* i) In the Recursive program after the function call no work. [In the Recursive Program only one function call & after that No work] then it is called Tail Recursive Program.

Ex: Above Program.

(ii) Lot of stack space waste. [unnecessarily]
As after function call, there is nothing to kept in stack.

(iii) Advantage with the Tail Recursion is, we can write equivalent Non-Recursive Program very easily with the help of for loop or while loop.

⇒ Non-Tail Recursion :-

```
NTR (n)
{
  If (n≤1)
    return
  else
  {
    NTR (n-2);
    Pf (n);
    NTR (n-1);
  }
}
```

i/p :- n = 5

NTR (5)
1. If
2. NTR (3)
3.
4.
1
2. NTR (1)
3 Pf 1. If (1≤1) ✓

o/p :-
3, 2, 5, 2, 4, 3, 2

without Dynamic Programming :

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

$= 2^n$

max (n-level tree)

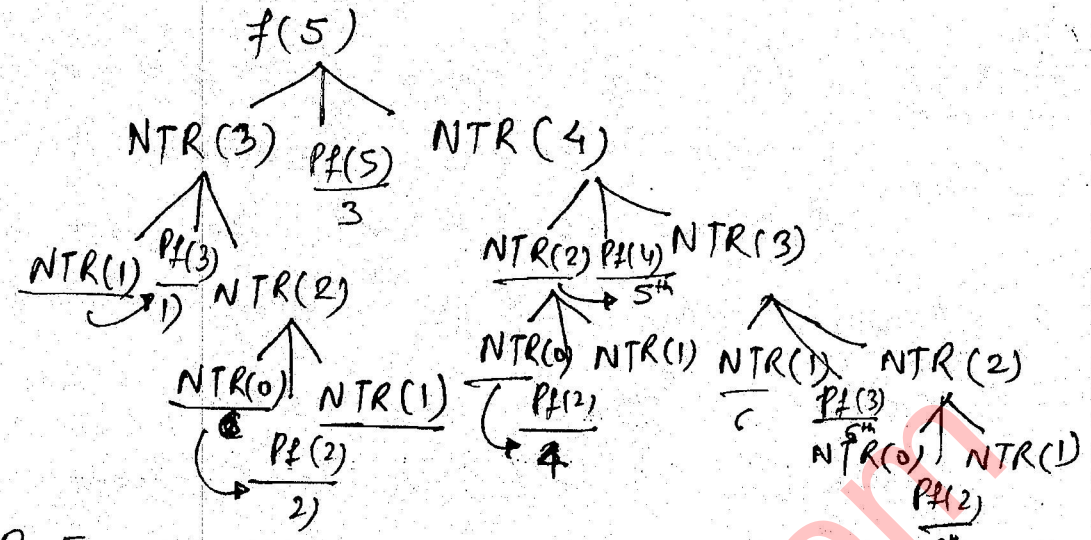
Stack :- O(n)

with Dynamic Prog.

Time :- T(n) = Take only distinct (n-1) will covers all

O(n)

Space ⇒ n level stack
n B. (array, Table
⇒ (2n) ⇒ O(n)



O/P:- 3, 2, 5, 2, 4, 3, 2

★ Note :-

★ i) In the ~~Non~~ - Recursive Program after the function call there is something to do then it is called Non-tail Recursive Program

★ ii) we are not wasting stack space.

★ iii) writting equivalent Non-Recursive is diffjicult. [But Possible]

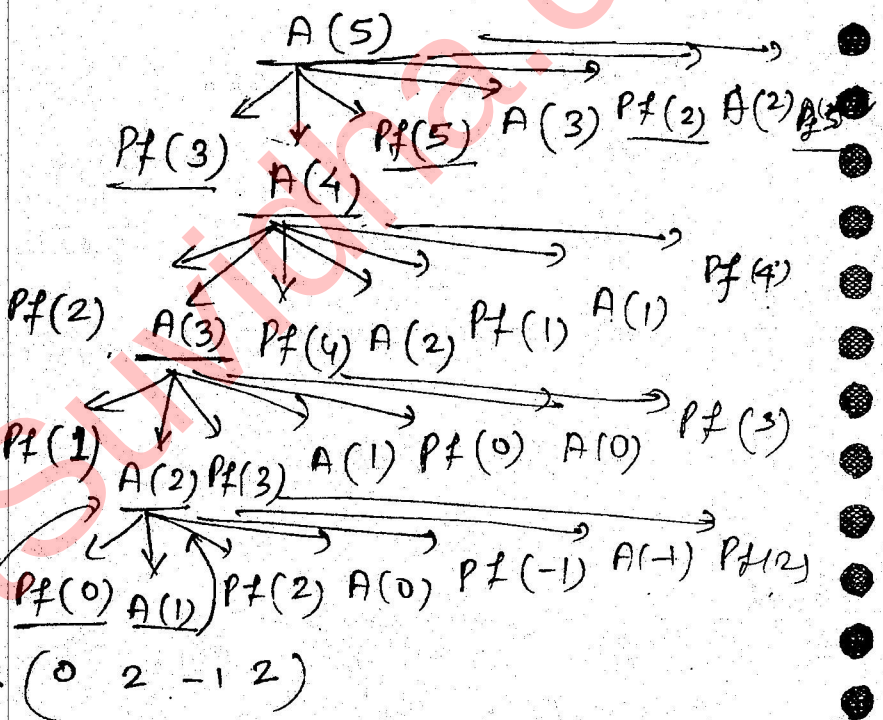
★★ After writting Equivalent Non-Recursive Program to Non-tail Recursion, then also stack (Data Structure) required. to perform Push pop.

★ Bingry Search is tail Recursive Program stack space is O(logn) But we can write equivalent Non-Recursive Program with O(1) stack-space.

Note :- for the given Recursive Program if we write equivalent Non-Recursive Program and in the Non-Recursive Program also "stack" required then that Recursive Program is called Non-tail otherwise Tail Recursive Program.

Ex: ②

```
A(n)
{
 If (n≤1)
  return;
 else
 {
  Pf(n-2);
  A(n-1);
  Pf(n);
  A(n-2);
  Pf(n-3);
  A(n-3);
  Pf(n);
 }
}
```



Time :- 3^n (w/o Dynamic Prog., with Dynamic = O(n)

i/p = 5

o/p = 3, 2, 1, 0, 2, -1, 2, 3, 0,

3, 2, 1, 0 2, -1, 2, 3, 0, 3, 4, 0, 2, -1, 2, 14, 5, 1, 0, 2, -1,

2, 3, 03, 2, 0, 2, -1, 2, 5

:- "Indirect Recursion" :-

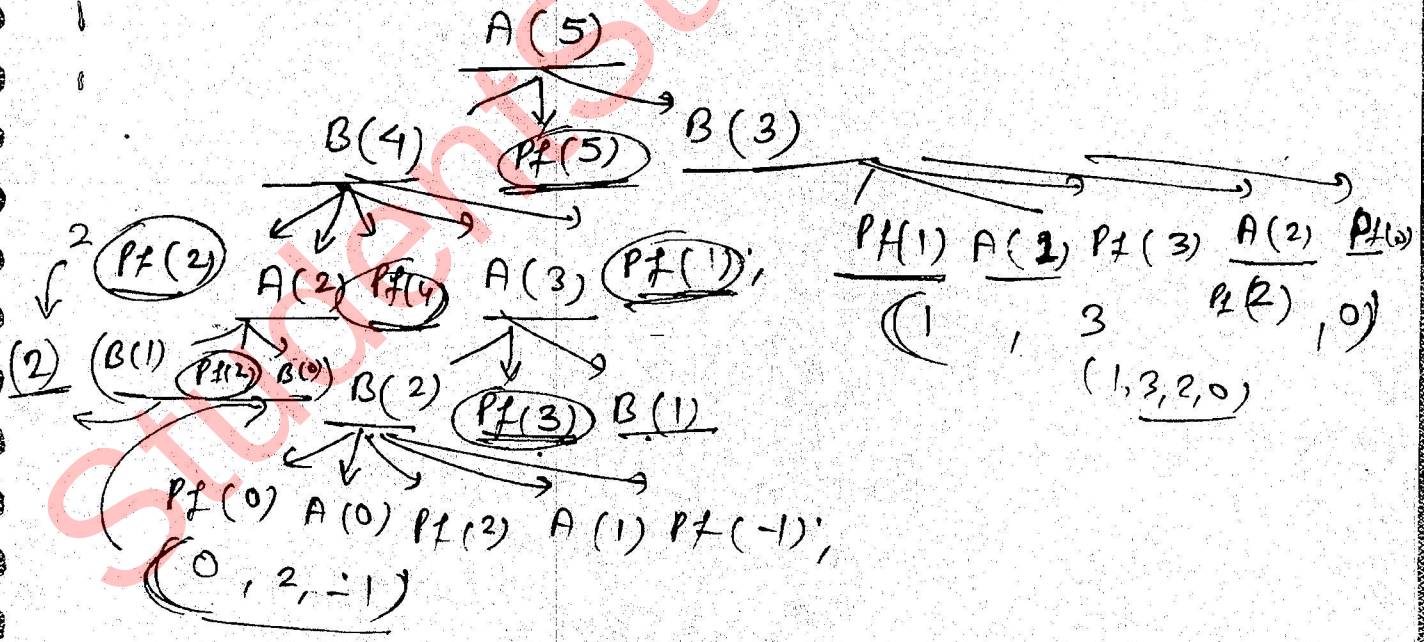
Date :- 14/oct/2017

```
B(n)
{
  If (n<=1)
  return;
  else
  { Pf(n-2);
    A(n-2);
    Pf(n);
    A(n-1);
    Pf(n-3);
  }
}
```

```
A(n)
{
  If (n<=1)
    return;
  else
  { B(n-1)
    Pf(n);
    B(n-2);
  }
}
```

i/p = A(5)

o/p = 2, 2, 4, 0, 2, -1, 3, 5, 1, 3, 2, 0



Time :- $O(2^n)$ (w/o dynamic)

Dis Hnd function call :- $(n+n) = 2n$ (A, B)

$O(n)$

"Nested Recursion" :-

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } n = 0 \\ A(m-1, A(m, n-1)) & \text{otherwise} \end{cases}$$

i/p :- A(1,5)

$A(1,5) \xrightarrow{7}$ $A(0, A(1,4)) \xrightarrow{6}$ $A(0, A(1,3)) \xrightarrow{5}$ $A(0, A(1,2)) \xrightarrow{4}$ $A(0, A(1,1)) \xrightarrow{3}$ $A(0, A(1,0)) \xrightarrow{2}$ $A(0,1) \xrightarrow{2}$] n+1

o/p = 7

i/p :- A(2,5)

$A(2,5) \xrightarrow{13}$ $A(1, A(2,4)) \xrightarrow{11}$ $A(1,11) \Rightarrow 23$

$A(1, A(2,3)) \xrightarrow{9}$ $A(1,9) \Rightarrow 11$

$A(1, A(2,2)) \xrightarrow{7}$ $A(1,7) \Rightarrow 9$

$A(1, A(2,1)) \xrightarrow{5}$ $A(1,5) \Rightarrow 7$

$A(1, A(2,0)) \xrightarrow{3}$ $A(1,3) \xrightarrow{5}$

$A(1,1) \xrightarrow{3}$

Ap

1,0 = 2
1,1 = 3
1,2 = 4
1,3 = 5
1,4 = 6
⋮
1,10 = 12

i/p = A(3,5)

A(3,5) ⇒ 253

~~A(2, A(3,5))~~

A(2, A(3,4)) ⇒ 253

A(2, A(3,3)) ⇒ 125

A(2, A(3,2)) ⇒ 61

A(2, A(3,1)) ⇒ 29

A(2, A(3,0)) ⇒ A(2,5) ⇒ 13

A(2,1) ⇒ 5

2,0 ⇒ 3
2,1 ⇒ 5
2,2 ⇒ 7
2,3 ⇒ 9
2,4 ⇒ 11
2,5 ⇒ 13

⇒ "Infix / Postfix" :-

Ex: ①

Infix :- a+b

Postfix :- ab+

Prefix :- +ab

★ Note :- In all the above three Notations, operands order can not change but operators order may change.

Infix: a+b*c

Postfix :- a+bc* ⇒ ~~ab*c~~ abc*+
①

Prefix :- a+*bc ⇒ ~~+a*bc~~ +a*bc
①

{ Processor like Compiler Convert 'Postfix' from infix to Postfix

Ex:③ :- Infix : $a + b * d \uparrow e - g * f / h + c$

Postfix : $a + b * d e \uparrow - g f * / h + c$

$a + [b d * e \uparrow] - [g f * h / +] c$

~~$a b + d * e \uparrow - g f * h c +$~~

$a b d * e \uparrow * + - g f * h / c +$

$a b d * e \uparrow * + g f * h / c * - +$

$a \quad \underset{\substack{1 \\ 2 \\ 5}}{b d e \uparrow *} + \quad \underset{\substack{3 \\ 4}}{g f * h /} - c +$

6

Prefix :- $+ - + a * b \uparrow d e / * g f h c$

Ex:④ :- $*$ Left to Right Associativity ; $\uparrow$ Right to Left Associativity

Infix :- $(d * (a-b) \uparrow (c+h)) \uparrow (g * e) / f$

(2) (1) (3) (4) (5) (6) (7)

$d a b - \quad c h + \uparrow * \quad g e * \uparrow f /$

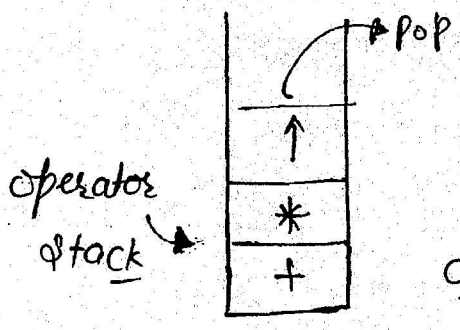
Postfix : $d a b - c h + \uparrow * g e * \uparrow f /$

Prefix :- $/ \uparrow * d \uparrow - a b + c h * g e f$

	Priorities	Associativity
$+, -$	⇒ (1)	L-R
$*, /$	⇒ (2)	L-R
$\uparrow$	⇒ 3	R-L

Prefix - Postfix using stack :-

Infix: a + b * c ↑ d



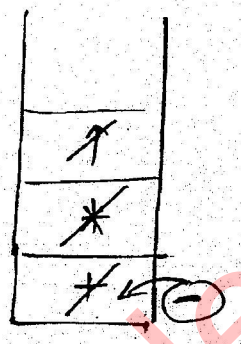
Postfix :

a b c d ↑ * +

operands order won't change so printed directly.

only ~~operands~~ operators are pushed into stack after comparing their Priority. If Priority greater ~~or equal~~ (≥) of Top of stack then Pop, otherwise Push. in equal case (go for Associativity)

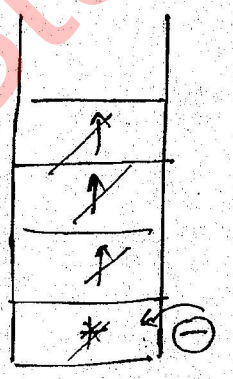
Infix : a + b * c ↑ d − e



+ & − both same Priority Associativity L-R so + will be done 1st so Pop +.

a b c d ↑ * + −

Ex: Infix: a * b ↑ c ↑ d ↑ e + − f



↑ Associativity R-L so last one will be done 1st so push.

a b c d e ↑ ↑ ↑ * f −

Note :- Infix - Postfix Conversion uses 'operator stack' [only operators will push inside stack].

ii) To convert n - length infix expression into Postfix will take O(n) time for

Every symbol Push / Pop / Print ⇒ O(1).

So n-symbol ⇒ n * O(1) ⇒ O(n)

Top	Next	operation
Low	High	⇒ Push
High	Low	⇒ Pop

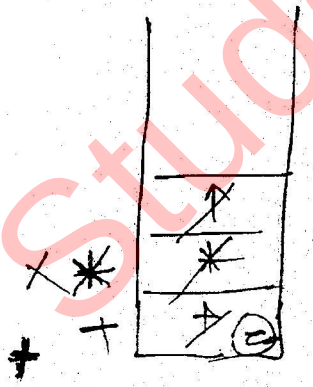
Same Priority

Associativity (L-R) ⇒ Pop

Associativity (R-L) ⇒ Push

Ex:- a + b * d ↑ e − g * f / h + c

Stack Size :-

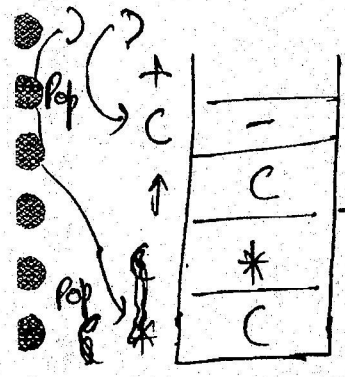


Time ⇒ O(n) [every case].

Size of stack = 3 (unit of operator stack)

a b d e ↑ * + g f * h / − c +

Ex:- Infix:- $(d*(a-b)\uparrow(c+h))\uparrow(g*e)/f$



stack size = 5 units

~~d a b - c h + * g e * ↑ f /~~

★ Note :- To Execute the Postfix Expression, one Scan is Enough But to Execute the Prefix & Infix multiple Scans required.

:- "Prefix to Postfix" :-

Ex:- ① Prefix : $*ab$

Postfix : $ab*$

Ex: ② :- $*(+cq)(-db) \Rightarrow cq+db-*$

Ex: ③ :- $*(+(-(/abc)(\uparrow(+(-de)f)g)))*ih$

$ab/c-$ $de-f+g\uparrow+ih**$

Postfix :- $ab/c-de-f+g\uparrow+ih**$

Ex: ④ :- $*a(-b(/c(\uparrow(\uparrow ef)(*g(+(*(/hi)j)k)))))$

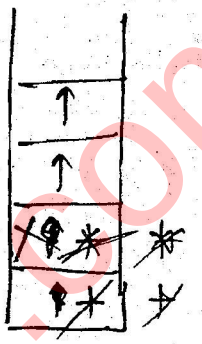
Ans :- $abcef\uparrow hi/j*k+g*\uparrow/-*$ $hi/j*k+g*\uparrow c/b-a*$

$abcef\uparrow hi/j*k+g*\uparrow/-*$

Prefix to Postfix conversion will use both stack.
operand & operator

"Post fix Evaluation" :-

Infix : 2+3*4/6↑(1↑7)+2*3-4



Postfix:- 234* 617↑↑/+ 23*+4-

4 unit of stack

234*617↑↑/+23*+4-

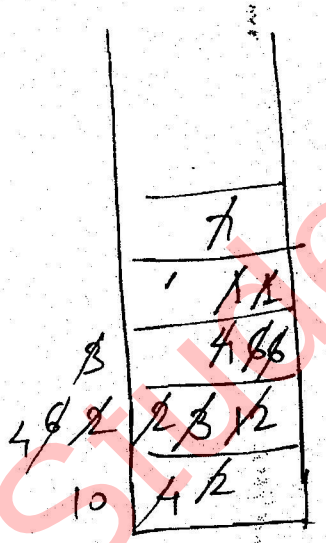
Postfix Evaluation :-

234*617↑↑/+23*+4-$ ← end symbol

operator then Pop1 [2 times] (operands for OP1 & OP2)

Evaluate r = OP1 OP OP2

then Push (r) into stack



OP2 = Pop()
OP1 = Pop()
r = OP1 OP OP2
12 = 3 × 4

7
OP2 = Pop()
OP1 = Pop()

OP2 = 6
OP1 = 12

10 - 4 = (6)

(1)
1↑7 = 1

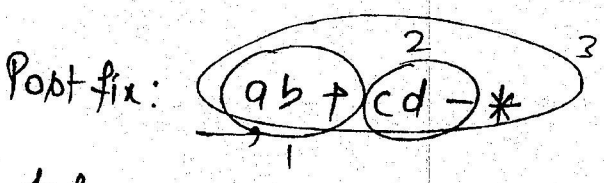
(6)
6↑1 = 6

12/6 = 2

2+2 = 4

* Postfix Evaluation uses operand stack &&

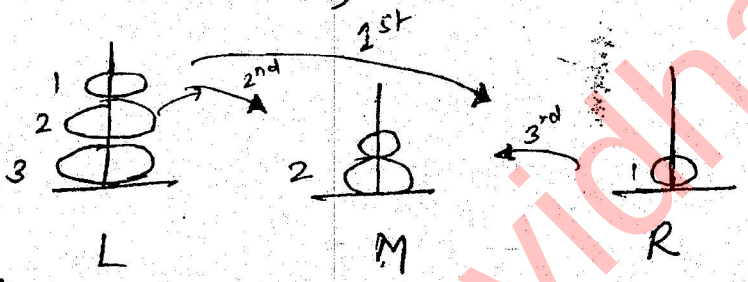
Time complexity => O(n)



Infix : (a+b)*(c-d)

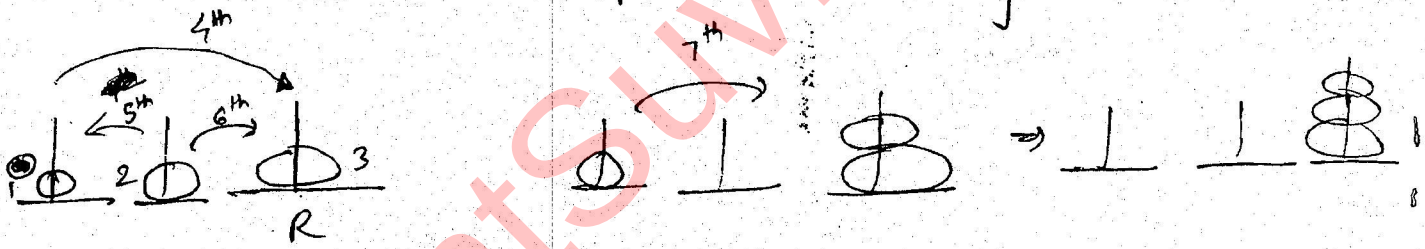
Towers of Hanoi :-

n = 3 (3-discs)



To If (3, L, M, R) left to Right using middle

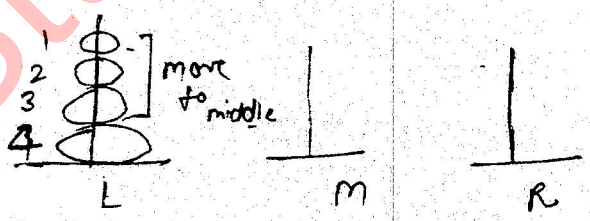
{
1. L-R
2. L-M
3. R-M
4. L-R
5. M-L
6. M-R
7. L-R
}



7 moves

# of moves = $2^n - 1$

n = 4 (4 discs)



1. L-M
2. L-R
3. M-R
4. L-M
5. R-L
6. R-M
7. L-M

Left to middle 3 discs n(3) = 7 moves.

~~To(4, L, M)~~

ToH (4, L, M, R)

{

7 moves ⇐ ToH (3, L, R, M) /* moving 3 discs from Left to middle using Right.

1 move ⇐ move (L→R) /* moving 4th Disk to Right.

7 moves ⇐ ToH (3, M, L, R) /* move 3 discs from middle to Right using left

15 moves }

"Algorithm" :-

```
ToH (n, L, M, R)
{
    If (n==0)
        return;
    else
    {
        ToH (n-1, L, R, M);   ⇒ T(n-1)
        move (L→R);           ⇒ C
        ToH (n-1, M, L, R);   ⇒ T(n-1)
    }
}
```

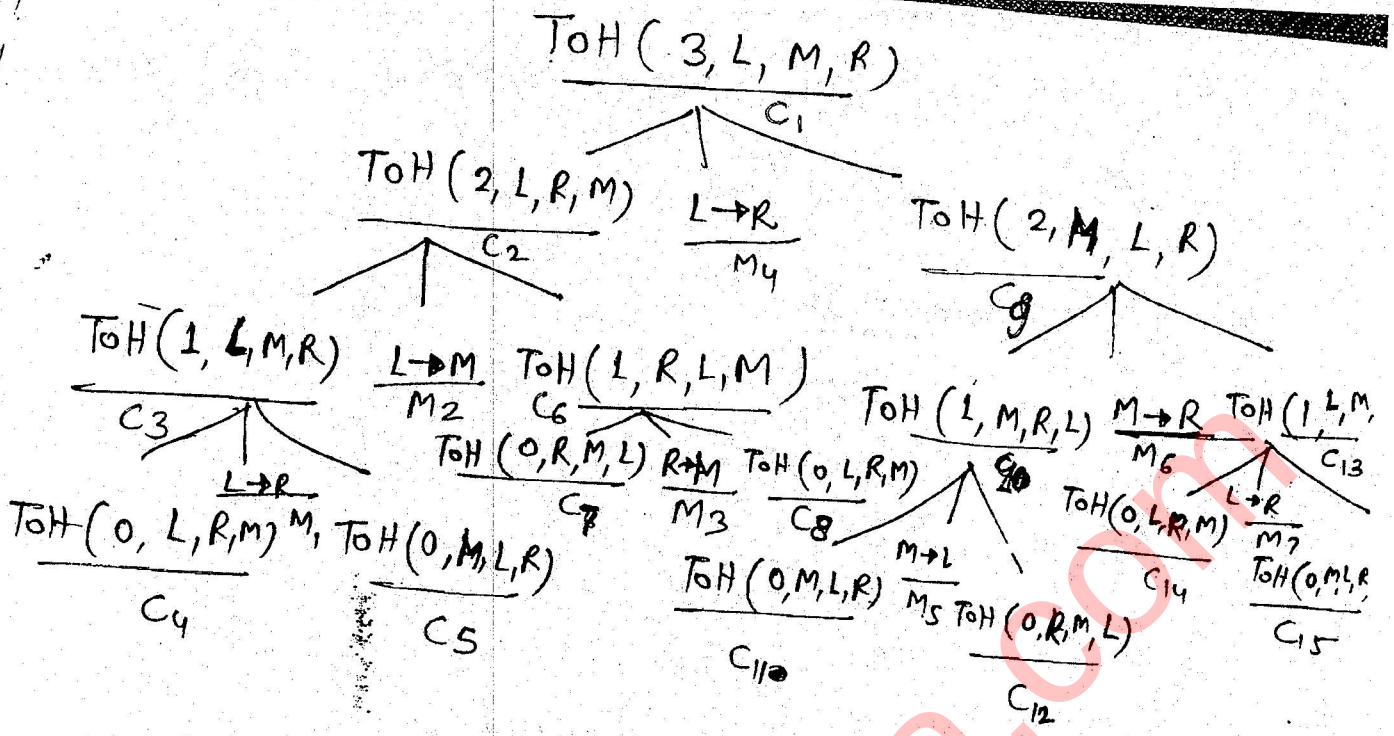
Recurren Rel^n

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

$= 2^n$

Stack space = n level tree

$= O(n)$



★ i) In ToH (n) After How many function call first move will be taken place = n+1

{ After C_4 there is a M_1 }.

★ ii) In ToH (n) Is there any function call After last move yes

{ After M_7 there is C_{15} }.

★ iii) In ToH (n) Is there any move After last function call No

[After C_{15} in Back Tracking, No work]

★ iv) In ToH (3) After How many function calls there is move from M → R After 12 function call.

[After C_{12} there is M_6 (M → R)]

v) In ToH(n) How many function calls are these ⇒ $2^{n+1}-1$

# of function call

$T(0) = 1$

$T(1) = 3$

$T(2) = 7$

$T(3) = 15$

$T(4) = 31$

$T(50) = 2T(49)+1$

∴ $\boxed{T(n) = 2T(n-1)+1}$ [Recurrence Rel^n for value

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

vi) In ToH(n) How many moves are these ?

$T(0) = 0$

$T(1) = 1$

$T(2) = 3$

$T(3) = 7$

$T(4) = 2T(3)+1 \Rightarrow 15$

∴ $\boxed{T(n) = 2T(n-1)+1}$

$T(50) = 2T(49)+1$