RECURSION

classmate
Date
Page

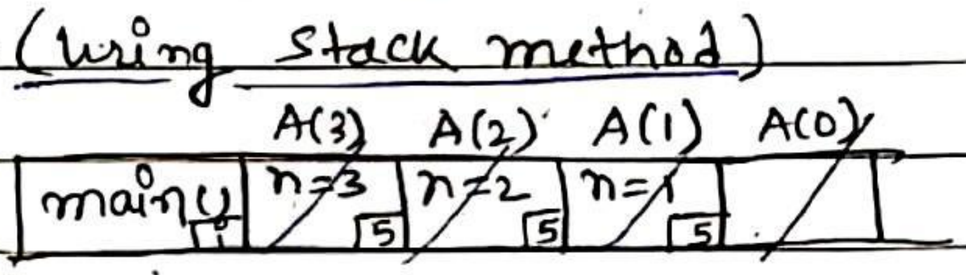
- Tracing the recursion =

ex: 1

```
A(n)
{
1. if (n>0)
2. { printf("%d", n-1);
3.   A(n-1)
4. }
}
```

```
main()
{ :
  :
  A(3);
  :
}
```

- (using stack method)



O/P: 2 1 0

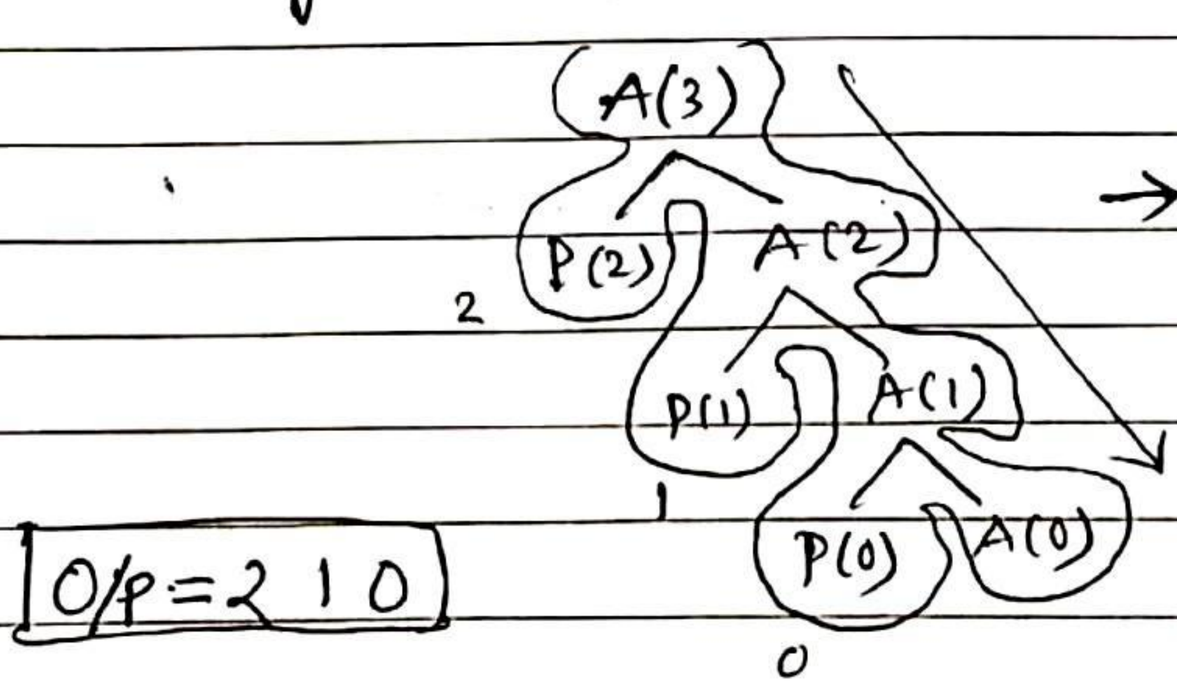
→ (5 activation record are pushed)

- Space complixity, $A(n) = n+1 = O(n)$

$A(3) = 3+1 = 4 = O(3)$.

- Time complixity, $T(n) = C + T(n-1) = O(n)$.

- (using tree method)



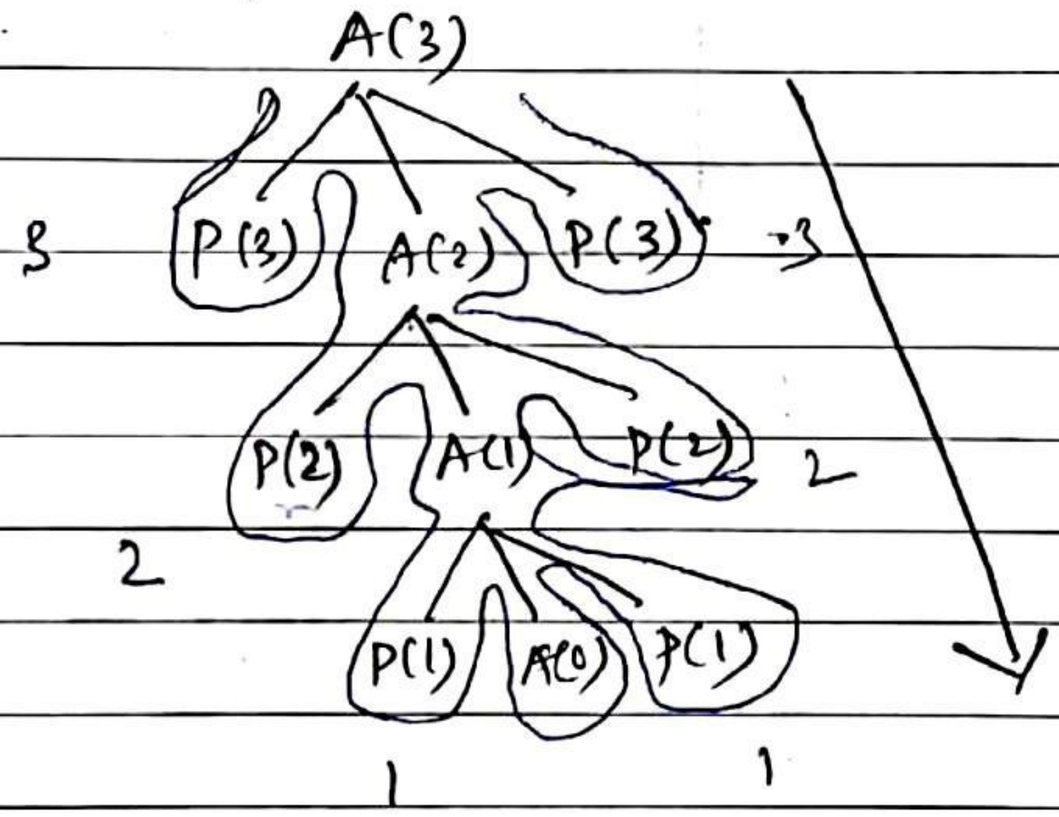
→ max stack record will be creat is = depth of the tree.
(4)

O/P = 2 1 0

ex:2

```
A(n)
{
- if (n>0)
{
- pf(n);
  A(n-1);
- pf(n);
}
}
```

using tree method find output when n=3.



o/p: 3211 2 3

Time complexity: → constant time required.

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

$= O(n)$.

space complexity:

$A(n) = n+1$

$= O(n)$

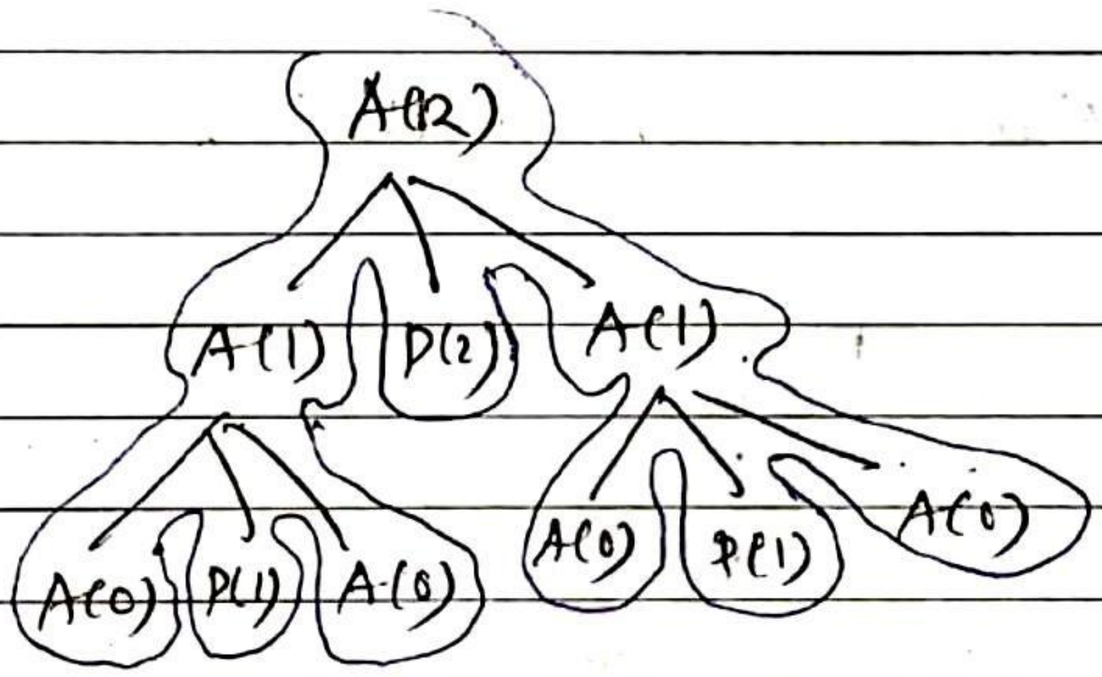
$A(3) = 4$

$= O(3)$.

ex: 3

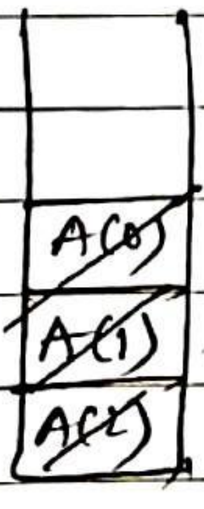
```
A(n)
{
  if(n>0)
    {
      A(n-1);
      pf(n);
      A(n-1);
    }
}
```

What will be ~~co~~ output ?



O/p : 1 2 1 (total record made in stack = depth of the tree)

(3)



when first time visited push and last time visited pop.

→ '3' record are created to execute this.

→ If there are less no. of statement go with tree method.

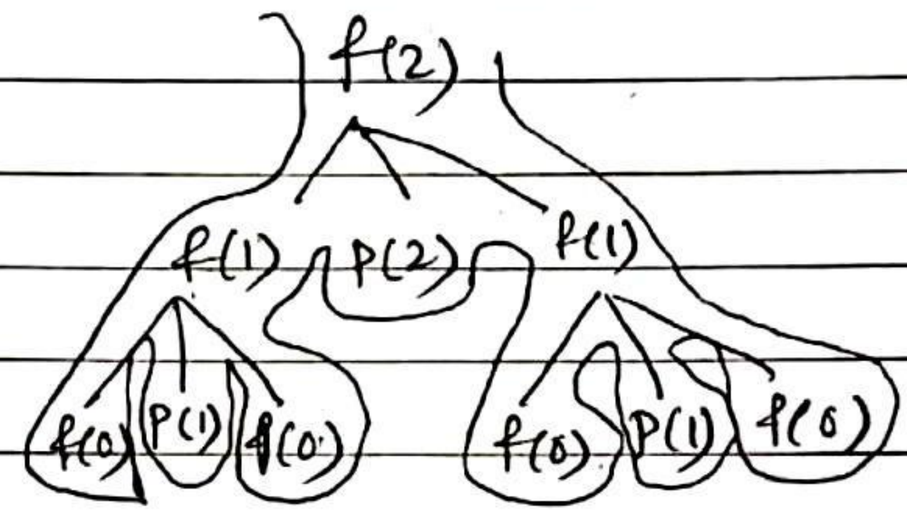
→ " " " more " " " stack method.

Important
Question asked on recursion.
→ Trace and find out output.

- Analysing recursion =

```
f(n)                    n=2
{
  if(n==0)
    return;
  f(n-1);
  pf(n);
  f(n-1);
}
```

tree method -



output : 1 2 1 f(2) = 7 (times function invocated)

- how many time function ~~f(10) = ?~~ invocated when f(10) ?

→

$f(1) = 3 = 2^{1+1} - 1$

$f(2) = 7 = 2^{2+1} - 1$

$f(3) = 15 = 2^{3+1} - 1$

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

$f(10) = 2^{10+1} - 1$

= 2047 (no. of time/functions are invocated).

c(1) = 1
c(2) = 2
c(10) = 10

other way -

$F(n) = 2F(n-1) + 1 \longrightarrow (1)$

$F(n-1) = 2F(n-2) + 1 \longrightarrow (2)$

$n(n-2) = 2F(n-3) + 1 \longrightarrow (3)$

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

$= 2^2 F(n-2) + 2 + 1$

$= 2^2 (2F(n-3)+1) + 2 + 1$

$= 2^3 F(n-3) + 2^2 + 2^1 + 2^0$

$F(n) = 2^i F(n-i) + 2^{i-1} + 2^{i-2} + \cdots 1.$

$F(n) = 1, \; n = 0$ | $n - i = 0, \; i = n$

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

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

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

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

$= 2^{10+1} - 1 \Rightarrow 2^{11} - 1 \Rightarrow 2047.$

(total no. of fun call required).

time complexity,

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

$= O(2^n) \cdot O(2^n).$

span complexity,

$S(n) = O(n).$

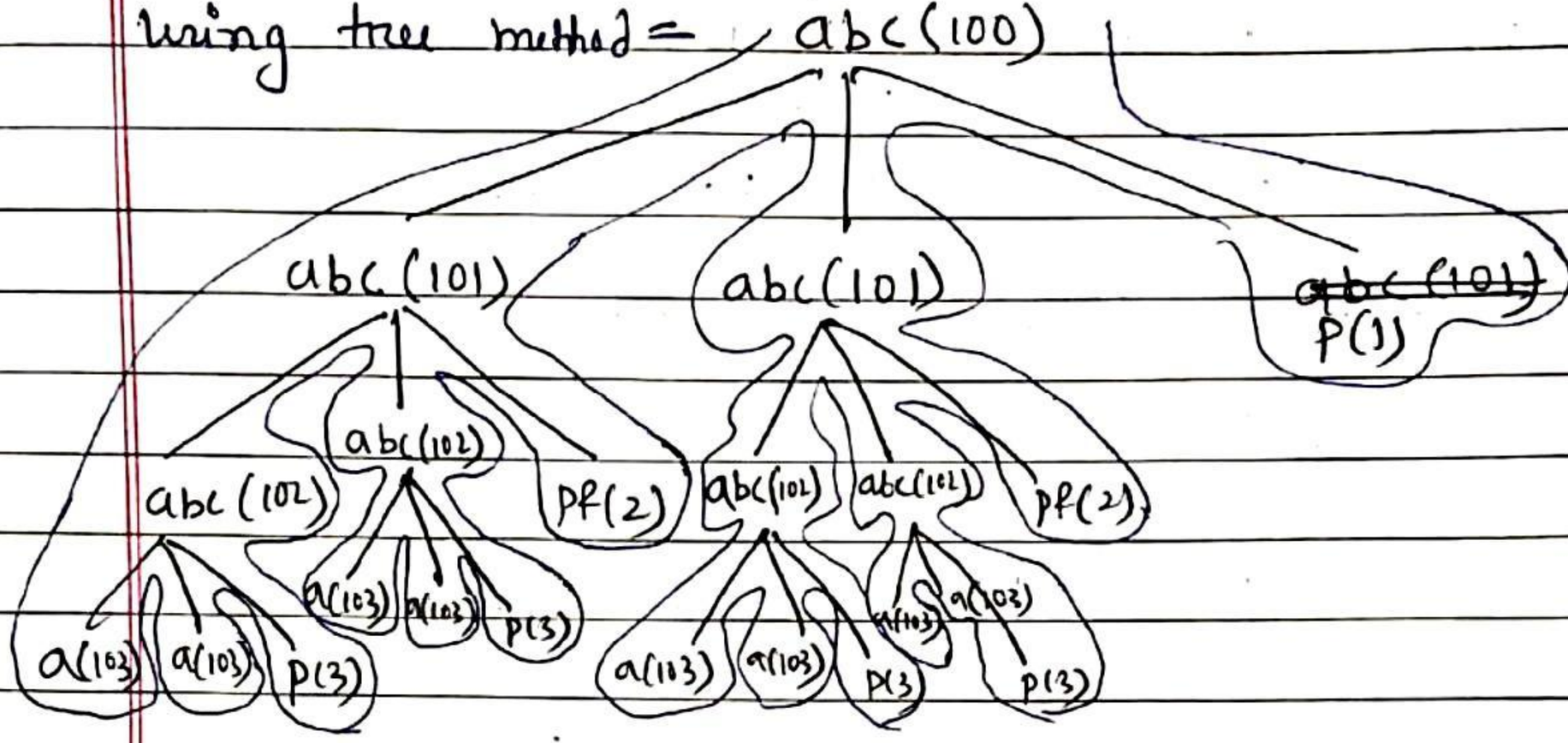
crate-2001

(B1)

```
void abc (char *s)
{
  if (s[0] == '\0')
    return;
  abc (s+1);
  abc (s+1);
  pf ("%c", s[0]);
}
main()
{ abc ("123");
}
```

1	2	3	\0
100	101	102	103

using tree method = abc(100)



(3323321) - Output.
(7-characters printed)

(Q2)

```
void abc (char *s)
{
 if (s[0] == '10') return 0;
 abc (s+1);
 abc (s+1);
 printf ("%c", s[0]);
}
main ()
{
  abc ("123");
}
```

If abc(s) is called with a null-terminated string 's' of length 'n' characters (not counting the null ('10') character), how many character will be printed by abc(s)?

→ C(1) = 1

no. of characters printed.

$C(n) = 2C(n-1) + 1 \longrightarrow (1)$

$C(n-1) = 2C(n-2) + 1 \longrightarrow (2)$

$C(n-2) = 2C(n-3) + 1 \longrightarrow (3)$

$C(1) = 1$

$n - i = 1$

$i = n-1$

$C(n) = 2^3 C(n-3) + 2^2 + 2^1 + 1$

$C(n) = 2^i C(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 1$

$= 2^{n-1} C(1) + 2^{n-2} + 2^{n-3} + \cdots + 1$

$= 2^{n-1} + 2^{n-2} + 2^{n-3} + \cdots + 1$

$= 1 \frac{(2^n - 1)}{2-1}$ (GP)

$C(n) = 2^n - 1$

$= 2^3 - 1$

$= 8 - 1$

$= 7$

Gate 2004 (B3)

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

Time complexity = ?

a) $O(n)$ b) $O(n \log n)$ c) $O(n^2)$ d) $O(2^n)$.

→ $T(n) = 2T(n-1) + 1$ $(n - i = 1)$

$T(n) = 1$ if $n = 1$ $T(1) = 1$

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

$T(n) = 2^{n-1} + 2^{n-2} + 2^{n-3} + \cdots + 1 = 2^n - 1 \Rightarrow O(2^n)$

gate-2005

(84)

```
void foo (int n, int sum)
 {
1. int  K=0, j = 0;
2. if (n==0) return;

3. K = n % 10 , j = n/10;
4. sum = sum + K;
5. foo (j, sum);
6. printf ("%d", K);
 }

int main()
{
  int a = 2048, sum = 0;
  foo (a, sum);
  printf ("%d", sum);
}
```

→

main	foo()	foo()	foo()	foo	foo()
a = 2048	n = 2048	n = 204	n = 20	n = 2	n = 0
sum = 0	sum = 8	sum = 8+4	sum = 12	sum = 12+2	sum = 14
	K = 8	K = 4	K = 0	K = 2	K = 0
	j = 204 [6]	j = 20 [6]	j = 2 [6]	j = 0 [6]	j = 0

bottom stack — pop

Ans 2, 0, 4, 8, 0 — output

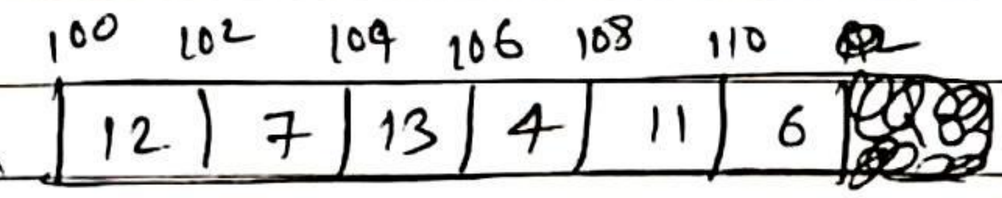
Gate-2005-2010

(Q5)

```
int f(int *a, int n)
  {
    if(n <=0) return 0;
    else
     if (*a %2 ==0)
      return *a+f(a+1,n-1);
    else
      return *a-f(a+1, n-1);
    }

int main() {
  int a[] = {12, 7, 13, 4, 11, 6};
  printf("%d", f(a,6));
  return 0;
}
```

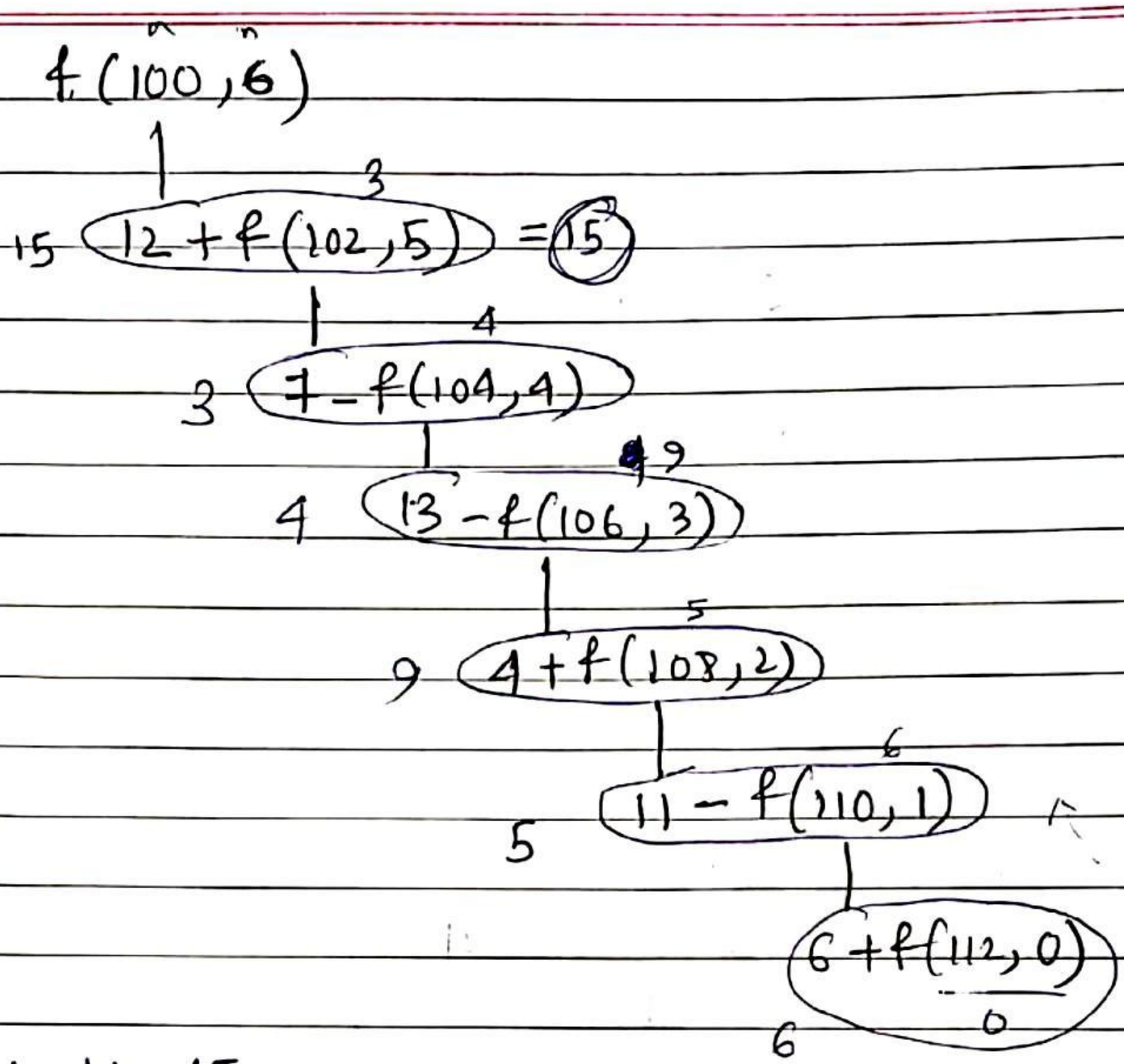
* → When function contain many lines then go with the stack method.
→ When function contain only recursive function call then go with the tree method



f(100, 6)

↓

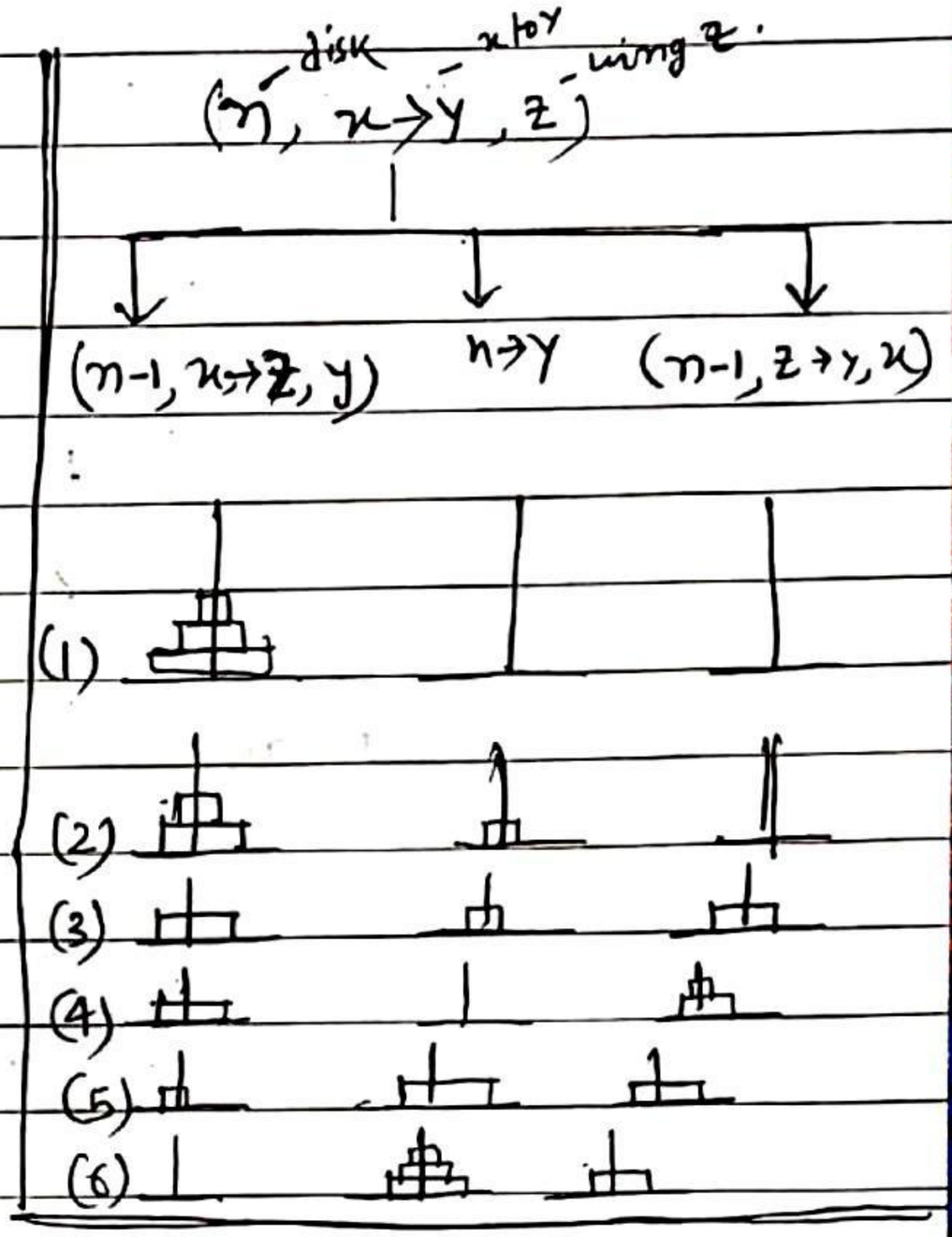
next page.



output: 15

- **Analysing the recursion program of Towers of Hanoi ⇒**

①
```
TOH(n, x, y, z)
{
  if(n>=1)
  {
    TOH(n-1, x, z, y);
    Move top 'x' to 'y';
    TOH(n-1, z, y, x);
  }
}
```



i) No. of function invocation required for 'n' disk =

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

For 3 disk $= 2^{3+1} - 1$

$= 15$

$I(n) = 2I(n-1) + 1 \longrightarrow (I)$

$I(n) = 1 \quad , n = 0$

$I(n-1) = 2I(n-2) + 1 \longrightarrow (II)$

$I(n-2) = 2I(n-3) + 1 \longrightarrow (III)$

$I(n) = 2(2I(n-2)+1)+1 = 2^2 I(n-2) + 2 + 1$

$= 2^2 (2I(n-3)+1) + 2 + 1$

$= 2^3 I(n-3) + 2^2 + 2^1 + 2^0$

$= 2^i I(n-i) + 2^{i-1} + 2^{i-2} \ldots + 1$ $\quad (n-i=0,\ n=i)$

$= 2^n I(0) + 2^{n-1} + 2^{n-2} + \ldots + 1$

$= 2^n + 2^{n-1} + 2^{n-2} + \ldots + 1.$

$Op = 1\,\frac{(2^{n+1}-1)}{2-1} = \boxed{2^{n+1} - 1 = O(2^n)}$

(3) Total disk movement required for n disk = $\boxed{M(n) = 2^n - 1}$

for 3 disk = $2^3 - 1$
= 7 Movement.

~~M(0)~~ $M(n) = 2M(n-1) + 1$

$M(n) = 0 \quad , \quad n = 0$

$M(n-1) = 2^2 M(n-2) + 1$ $\quad n - i = 0$

$M(n-2) = 2M(n-3) + 1$ $\quad n = i$

$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 1$

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

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

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

(3) Time complexity required = $T(n) = (2^{n+1} - 1)$
$= O(2^n)$.

$T(n) = 2T(n-1) + 1$ → constant time

$T(n) = 1 \quad , \quad n = 0$

$n - i = 0$
$i = n$

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

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

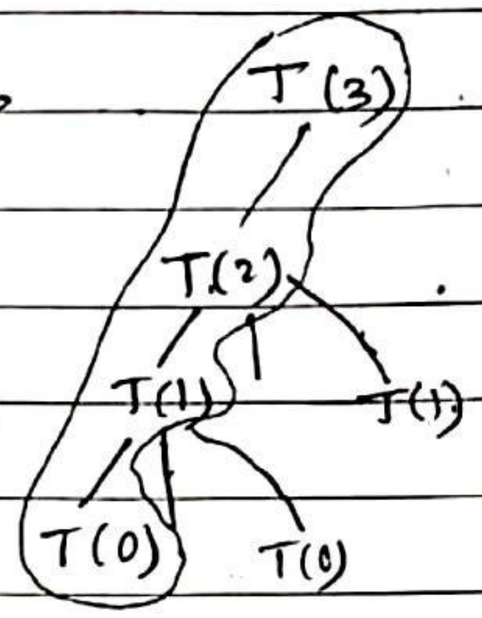
$= 2^{n+1} - 1$

$= 2^n 2^1 - 1$

$= O(2^n)$

(4) space complexity with 'n' disk = $S(n) = (n+1) = O(n)$.

→ depth of the tree.



$S(3) = 3+1$
$= 4$

(2) TOH(n, x, y, z) → modification of previous Algo. to minimize no. of function invocation.

```
{
 if(n > 1)
 { TOH (n-1, x, z, y);
   Move top 'x' to 'y';
   TOH (n-1, z, y, x);
 }
}
else if (n == 1)
   Move "x → y";
}
```

no. of function
now ~~Invocat~~ Invocation required — $I(n) = 2^n - 1 = O(2^n)$

$I(3) = 2^3 - 1 = 7$

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

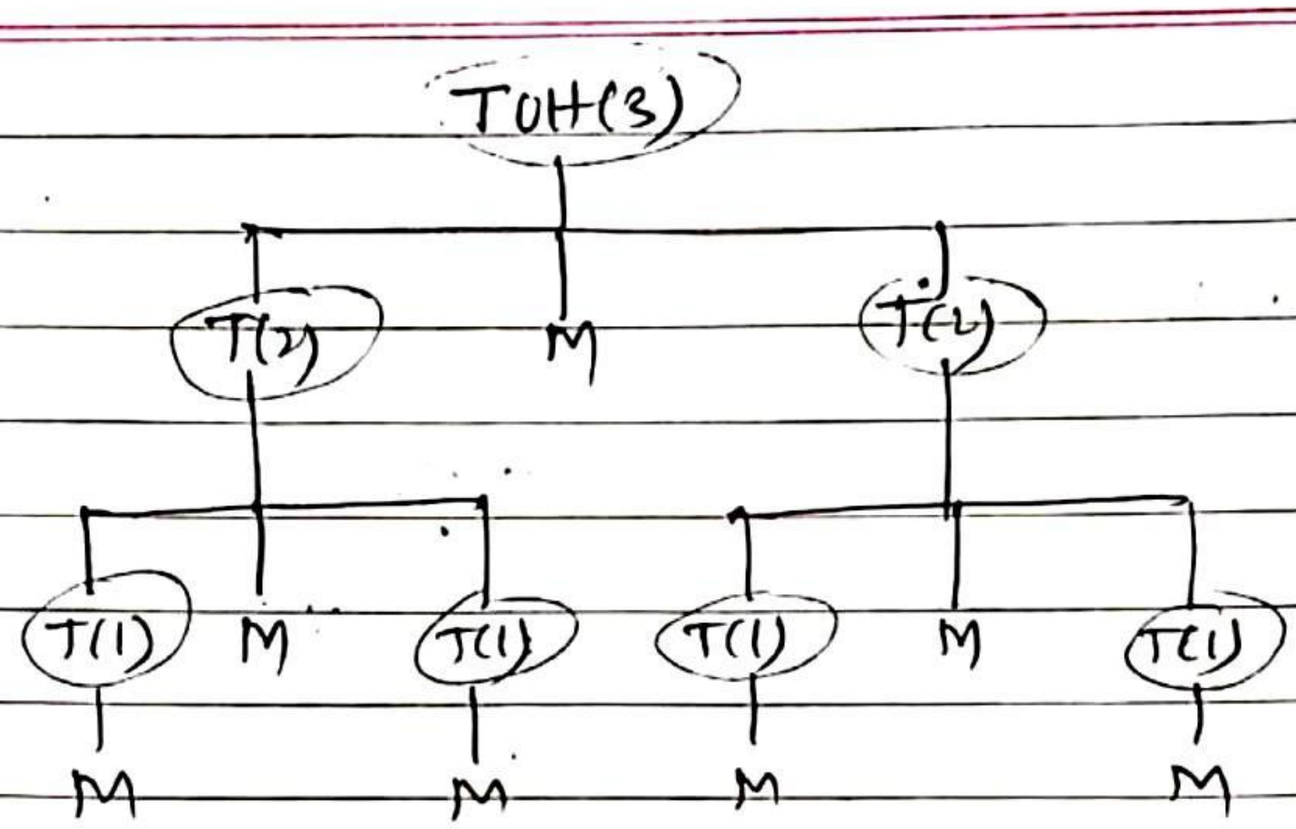
$I(n) = 1, \quad n = 1$ $\qquad n - i = 1$

$I(n) = 2^i I(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 1$

$= 2^{n-1} I(1) + 2^{n-2} + 2^{n-3} + \cdots + 1$

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

$= \frac{1(2^n - 1)}{2-1} = (2^n - 1) = O(2^n)$.



here
total function
invocation is (7).