Notebook
Pg. No.
Date / /

Recursion

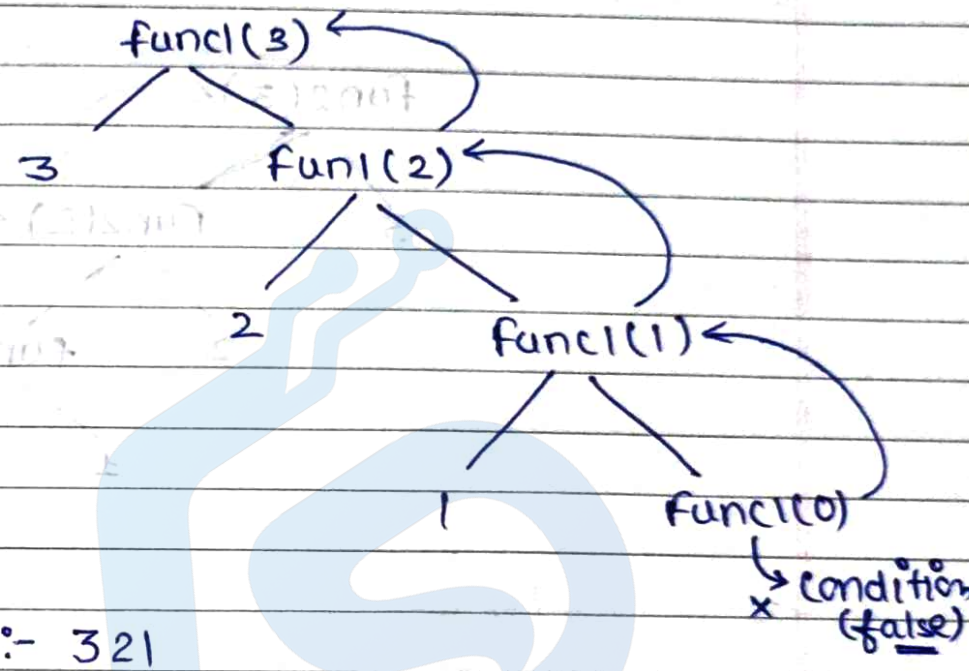
- What is Recursion?
- Example of Recursion.
- Tracing Recursion.
- Stack used in Recursion.
- Time Complexity
- Recurrence Relation.

If a function calling itself again and again, then the process is called Recursion.

Examples:-

```
void func1(int n)
{
    if(n>0)
    {
        printf cout<<n;
        func1(n-1);
    }
}
void main()
{
    int x=3;
    func1(x);
}
```

Output :- (How Recursion is working)

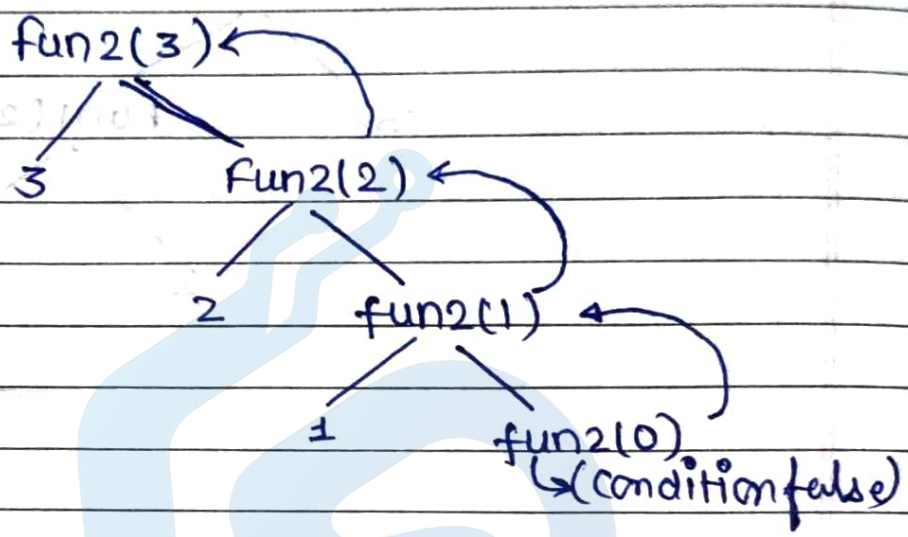


output :- 321

Example 2 :-

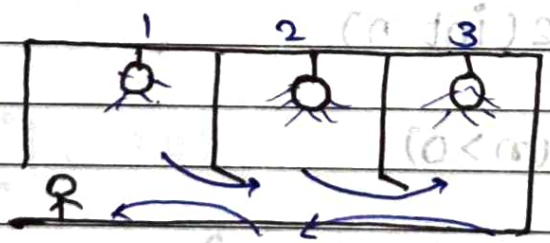
```
void fun2(int n)
{
    if(n>0)
    {
        fun2(n-1)
        printf( cout<<n;
    }

}
void main()
{
    int x=3;
    fun2(x);
}
```

Dry Run of Example 2 :-



O/P :- 1 2 3 (Printing is done ~~after~~ in returing times)

Ex:-



Step 1. Switch on Bulb
Step 2- Go to Next Room (Recursive call)

O/P :- 1 2 3

Step 1: Go to Next Room (Recursive call)
Step 2: Switch on bulb

O/P → 3 2 1

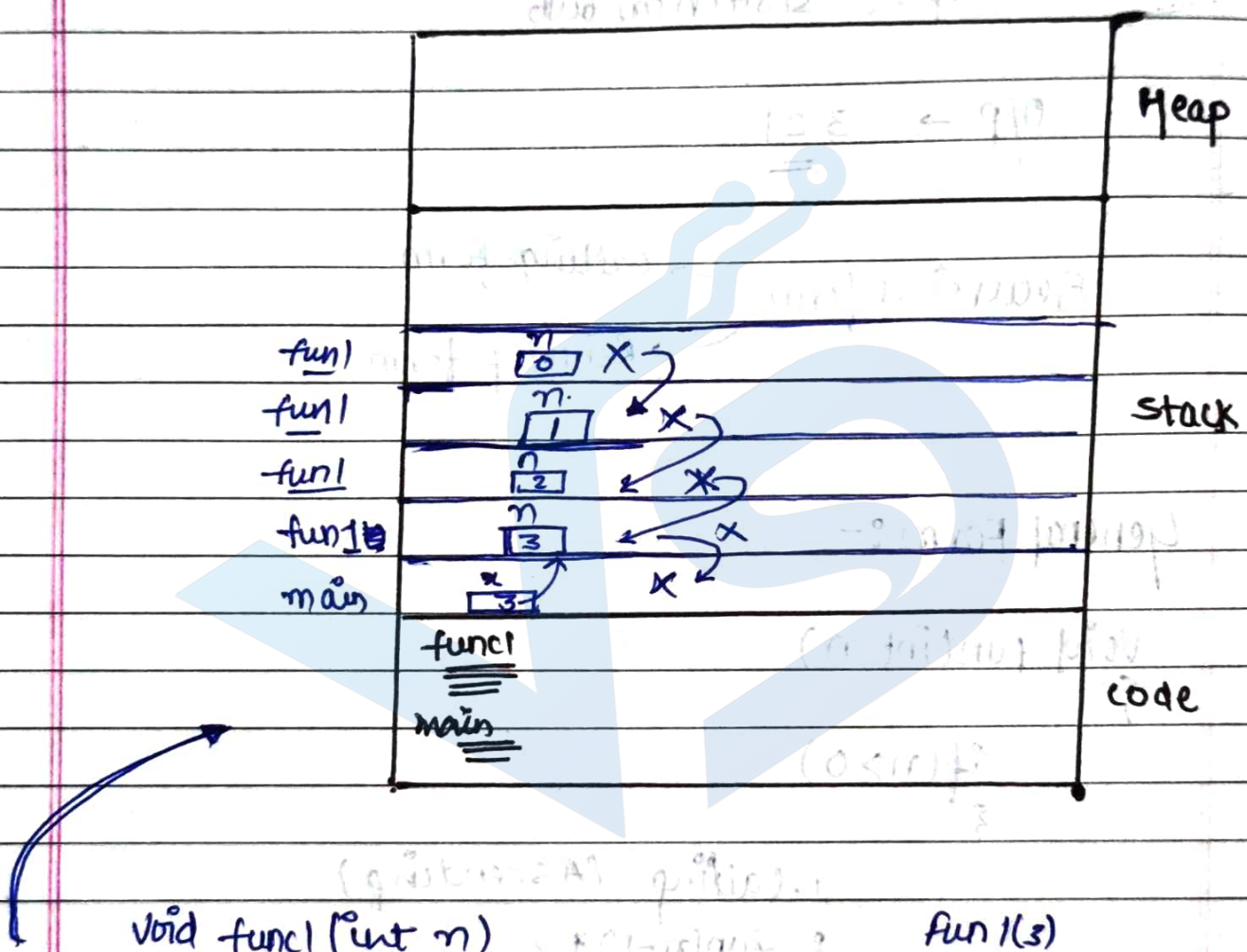
Recursive fun → calling fun
→ Returning fun.

General Form:-

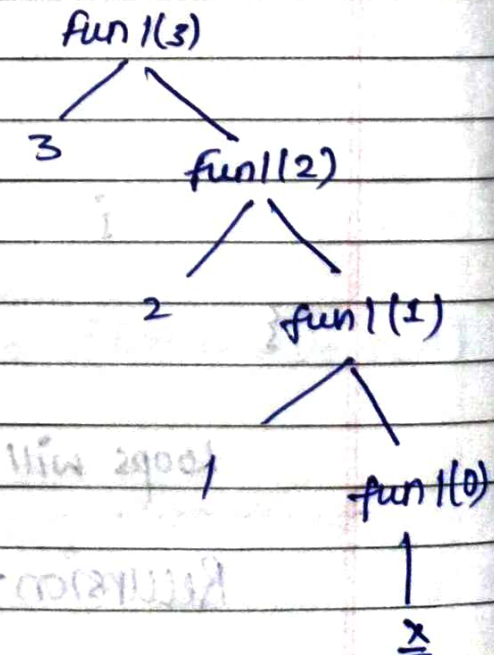
```
Void fun(int n)
{
    if(n>0)
    {
        1. Calling (Ascending)
        2. fun(n-1)*2
        3. Returning (Descending)
    }
}
```

Loops will have only Ascending ~~& Desc.~~ phase.

Recursion → Ascending
→ Descending




```
Void func1(int n)
{
    if(n>0)
    {
        cout << n;
        func1(n-1);
    }
}

Void main()
```



```
{
    int x=3;
    fun1(x);
}
```

o/p :- 321

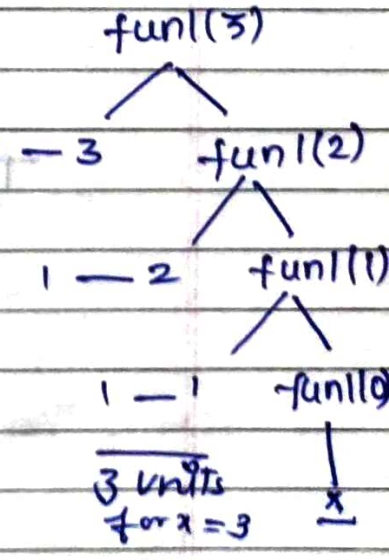
call → n+1

↳ ~~Time~~ O(n) (space complexity)

Recursive function are memory consuming function.

Time complexity (for Recursion)

```
void fun1(int n)
{
    if(n>0)
    {
        cout<<n; ——→ 1
        fun1(n-1);
    }
}
Void main()
{
    int x=3;
    fun1(x);
}
```



Time complexity = O(n)

Notebook
Pg. No.
Date / /

Time complexity using Recurse Relation :-

```
void fun1(int n)          ——— T(n) (Let)
{
    if(n>0)               ——— 1
    {
        cout<< n;         ——— 1
        fun1(n-1);        ——— T(n-1)
    }
}
```

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

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

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+2 & n>0 \end{cases}$$

We can solve that using induction method

By Induction method,

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

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

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

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

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

$\hookrightarrow T(n-3) + 1$

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

⋮

$$T(n) = T(n-K) + K \quad \text{---(4)}$$

Assume $n-K=0$

$n=K$

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

$$T(n) = T(0) + n$$

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

$\hookrightarrow O(n)$

~~STACK~~ STATIC VARIABLES IN RECURSION

```
int fun(int n)
{
    if(n>0)
    {
        return fun(n-1)+n;
    }
    return 0;
}
int main()
{
    int a=5;
    cout<<fun(a);
}
```

fun(5) ⇒ 15

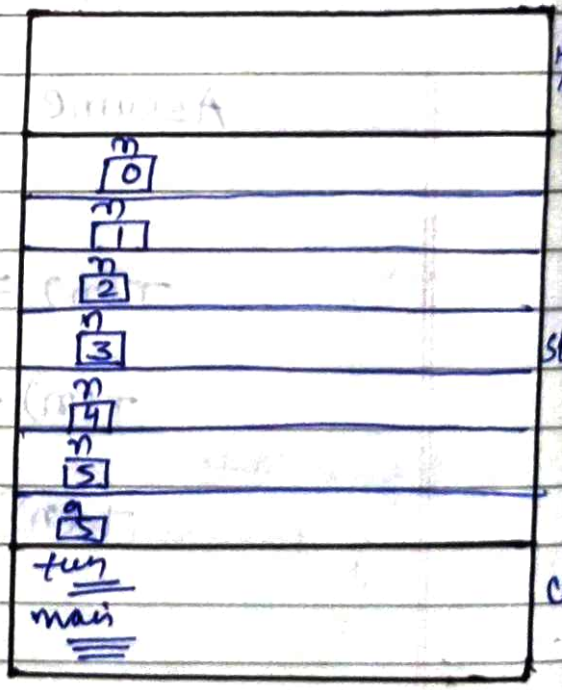
fun(4) + 5 = 15

fun(3) + 4 = 10

fun(2) + 3 = 6

fun(1) + 2 = 3

fun(0) + 1 = 1



Another Example :-

Static

```
int fun(int n)
{
    static int x=0;
    if(n>0)
    {
        x++;
        return fun(n-1)+x;
    }
    return 0;
}
int main()
{
    int a=5;
    cout << fun(a)
}
```

Global

```
static int x=0;
int fun(int n)
{
    if(n>0)
    {
        x++;
        return fun(n-1)
               +x;
    }
    return 0;
}
```

O/P :- 25

fun(5) → 25

fun(4) + 5 = 25

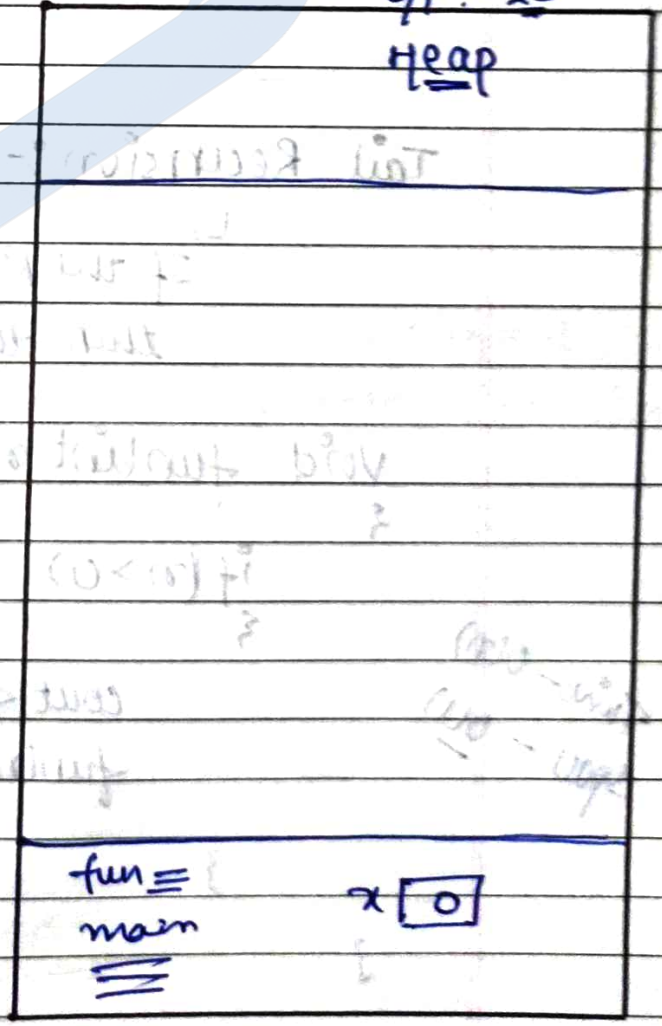
fun(3) + 5 = 20

fun(2) + 5 = 15

fun(1) + 5 = 10

fun(0) + 5 = 5

x: 0 → 1 → 2 → 3 → 4 → 5



Notebook
Pg. No.
Date / /

Types of Recursion:-

1. Tail Recursion
2. Head Recursion
3. Tree Recursion
4. Indirect Recursion
5. Nested Recursion.

Tail Recursion:-

If the Recursive call is last statement, then the Recursion is called Tail Recursion

```
void fun(int n)
{
    if(n>0)
    {
        cout<<n;
        fun(n-1);
    }
}
```

Time - O(n)
Space - O(1)

Everything is performed at calling time.

instead of `fun(n-1);` → `fun(n-1)+n;`

then it is not tail Recursion.

Notebook
Pg. No.
Date / /

Tail Recursion with Loops :-

```
void fun(int n)
{
   while(n>0)
   {
        cout << n;
        n--;
   }
}
```

Time → O(n)
space → O(n)

Head Recursion :-

If the Recursive call is first statement, then the Recursion is called Head Recursion

```
void fun(int n)
{
    if(n>0)
    {
       fun(n-1);
       cout<<n;
       ___
       ___
       ___
    }
```

Notebook
Pg. No.
Date / /

Head Recursion using Loops :-

```
void fun (int n)
{
    int i=1;
    while(n>0)
    {
        cout<<n;
```

```
void fun(int n)
{
    int i=1;
    while(i<=n)
    {
        cout<<i;
        i++;
    }
}
```

Tree Recursion

Linear Recursion:-

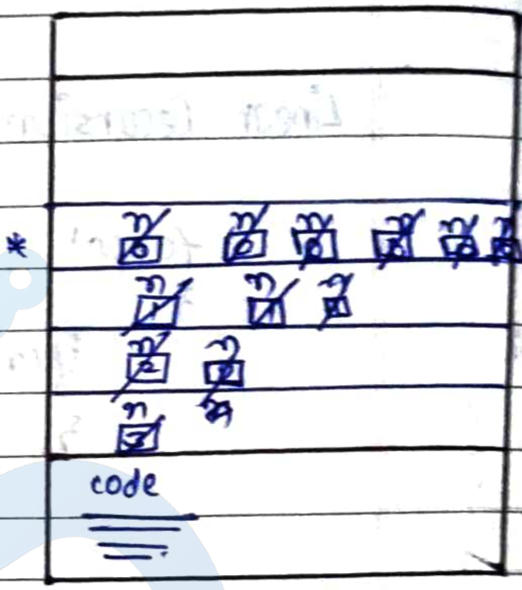
```
fun(n)
{
    if(n>0)
    {
        ===
        ===
        ===
        → fun(n-1);
        ===
        ===
    }
}
```

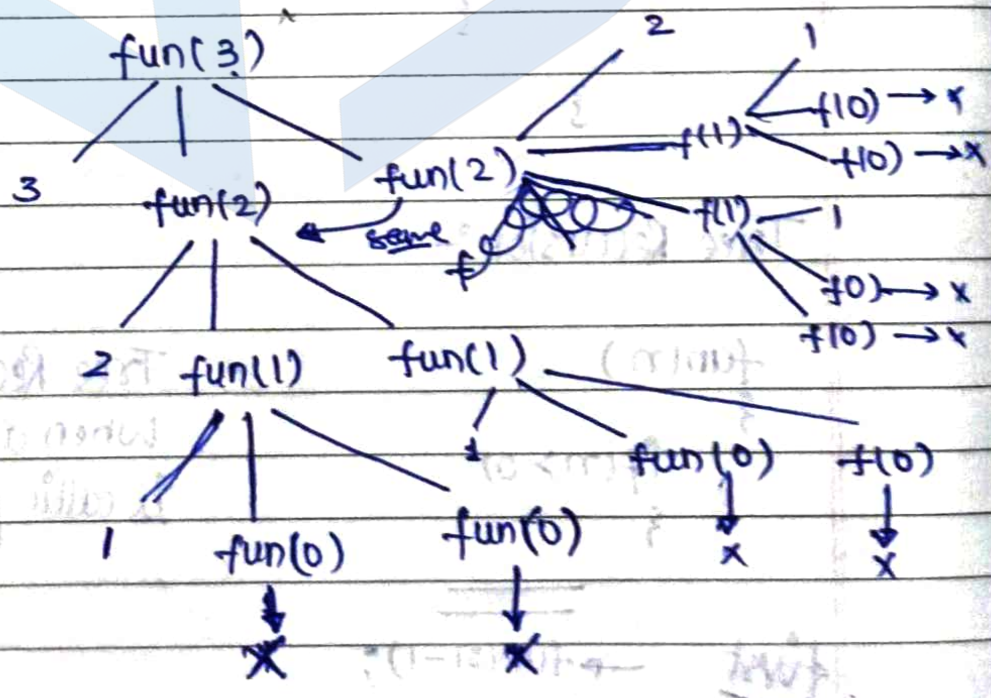
Tree Recursion:-

```
fun(n)
{
    if(n>0)
    {
        ===
        ===
first   → fun(n-1);
        ===
        ===
second  → fun(n-1);
        ===
        ===
    }
}
```

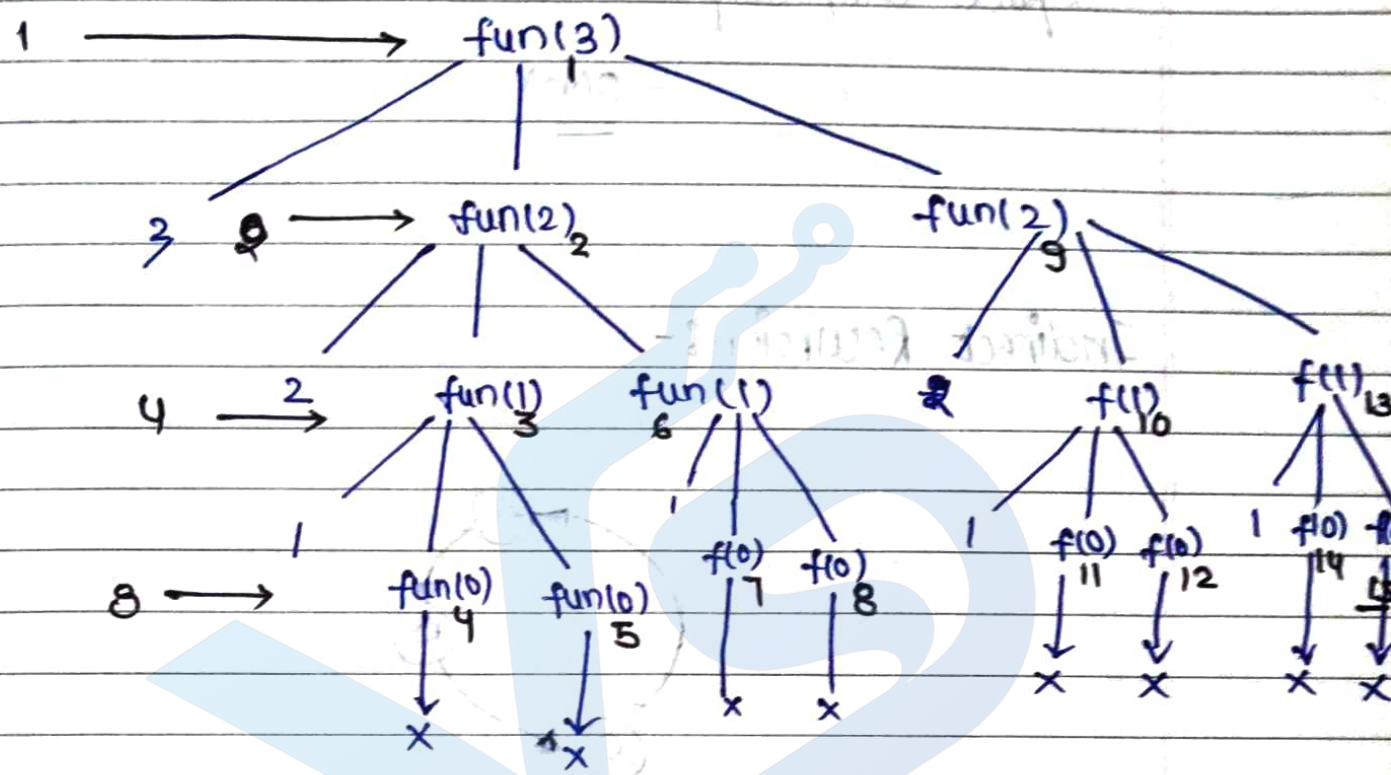
Tree Recursion is a recursion when a ~~function~~ recursive function is calling itself more than one time.

```
void fun(int n)
{
    if(n>0)
    {
        cout<<n;
        fun(n-1);
        fun(n-1);
    }
}
```





O/P: 3211211



15 calls are made

Time complexity →

$= 1+2+4+8 \Rightarrow 15$

$= 2^0 + 2^1 + 2^2 + 2^3 \Rightarrow 2^{3+1} - 1$

$= 2^0 + 2^1 + 2^2 + 2^3 + \text{----} + 2^n$

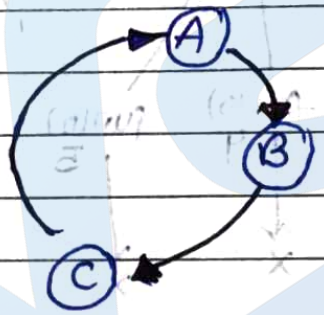
$= 2^{n+1} - 1$

$= O(2^n)$

Space Complexity :- n+1

= O(n)

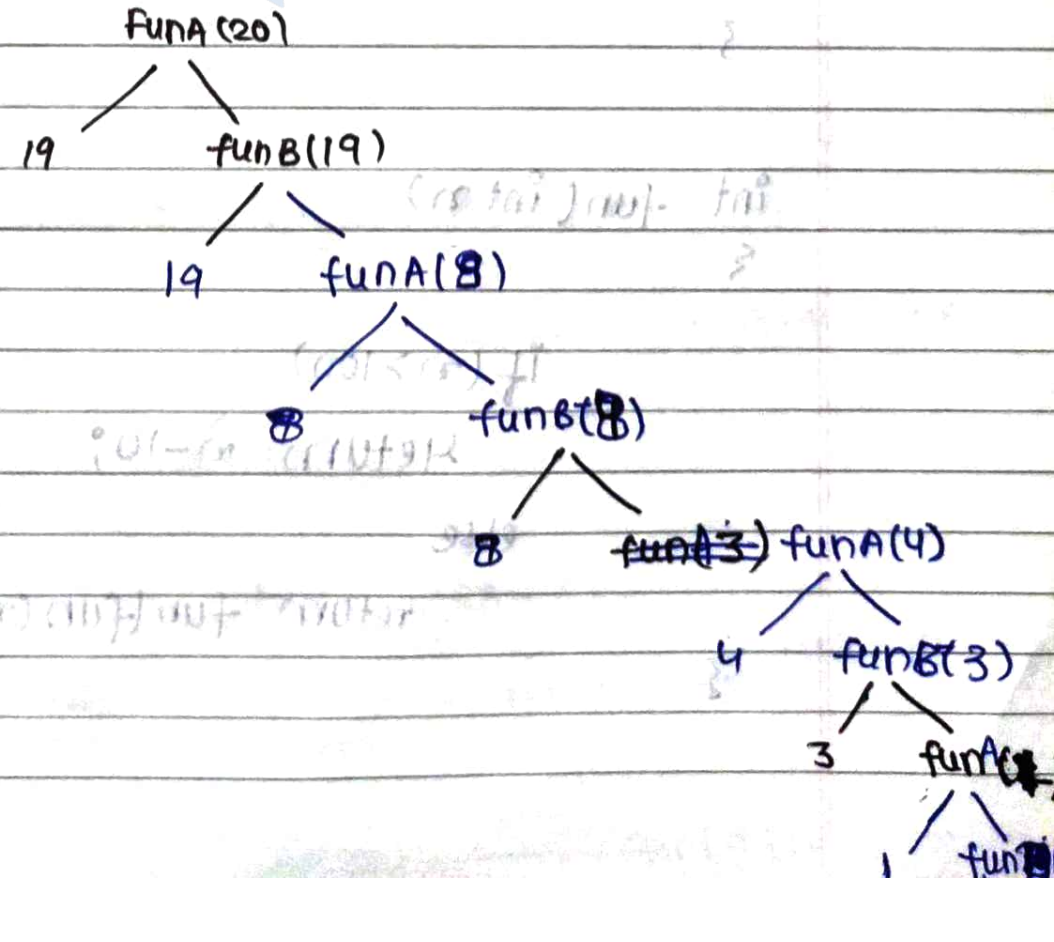
Indirect Recursion :-



If the function f1 calls another function f2 and f2 calls f1 then it is indirect recursion.

```
void A (int n)
{
    if (<——>)
    {
        B(n-1);
    }
}
void B(int n)
{
    if (<——>)
    {
        A(n-3);
    }
}
```

```
void funA(int n)
{
    if(n>0)
    {
        cout<< n;
        funB(n-1);
    }

}
void funB(int n)
{
    if(n>1)
    {
        cout<<n;
        funA(n/2);

    }

}
```

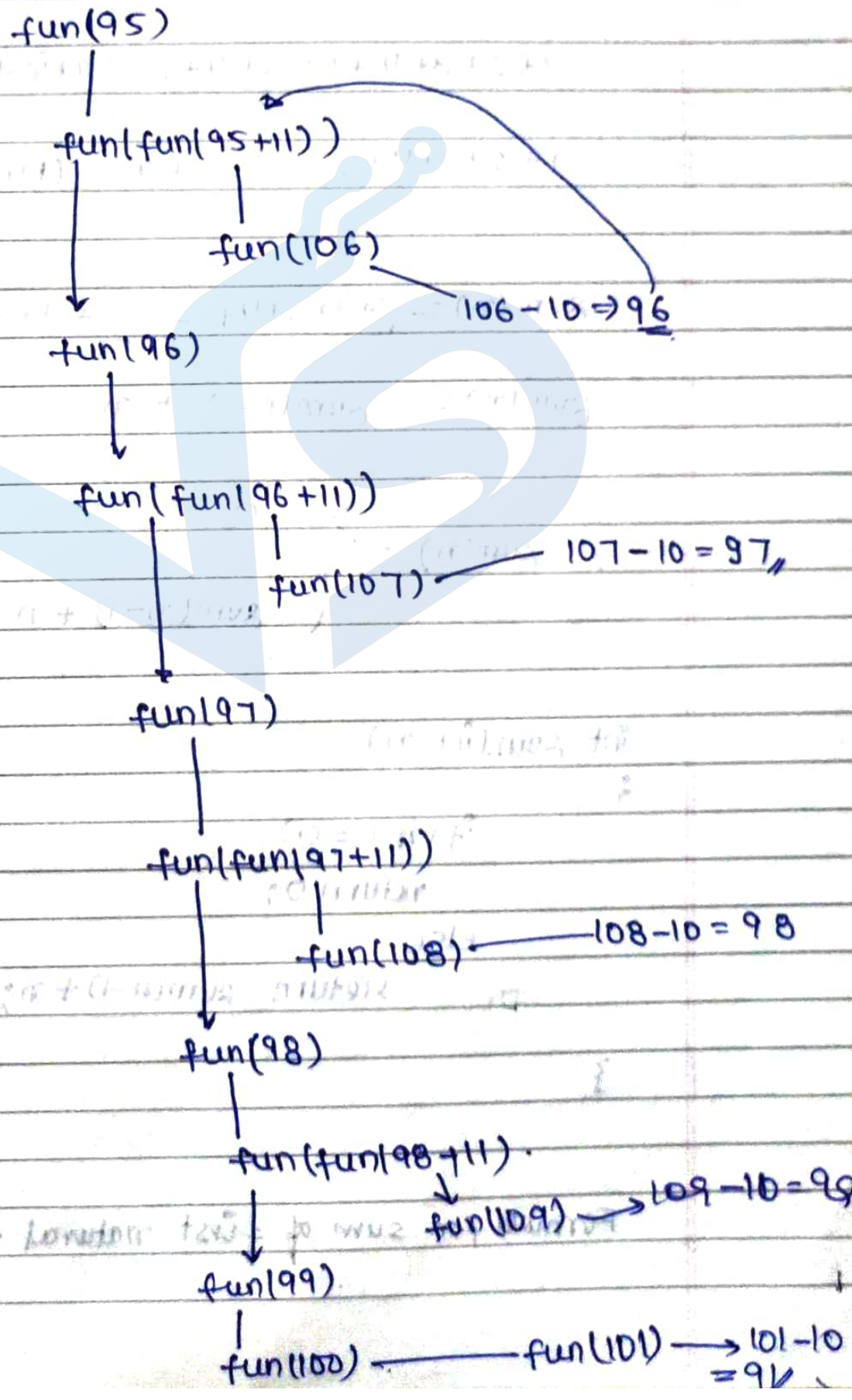


Nested Recursion :-

In nested Recursion, the recursive function will pass the parameter as a recursive call.

```
void fun(int n)
{
    if(<——>)
    {
        ——
        ——
        ——
        fun(fun(n-1));
    }
}

int fun(int n)
{
    if(n>100)
        return n-10;
    else
        return fun(fun(n+11));
}
```

let n = 95



fun(95) = 91

Notebook
Pg. No. _____
Date ____ / / ____

Sum of first n natural numbers :-

1 + 2 + 3 + 4 + 5 + 6 + 7 + ... (first 7)

1 + 2 + 3 + 4 + ... + n (first n)

$$sum(n) = \underbrace{1+2+3+4+\dots+(n-1)}+n$$

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

$$sum(n) = \begin{cases} 0 & n=0 \\ sum(n-1)+n & n>0 \end{cases}$$

```
int sum(int n)
{
    if(n ==0)
      return 0;
    else
      return sum(n-1)+n;
}
```

Formula for sum of first natural numbers $= \frac{n(n+1)}{2}$

```
int sum(int n)
{
    return n*(n+1)/2;
}
```

$O(1)$

```
int sum(int n)
{
    int i, sum = 0;
    for(i=0; i<=n; i++)
    {
        sum = sum + i;
    }
    return sum;
}
```

$O(n)$

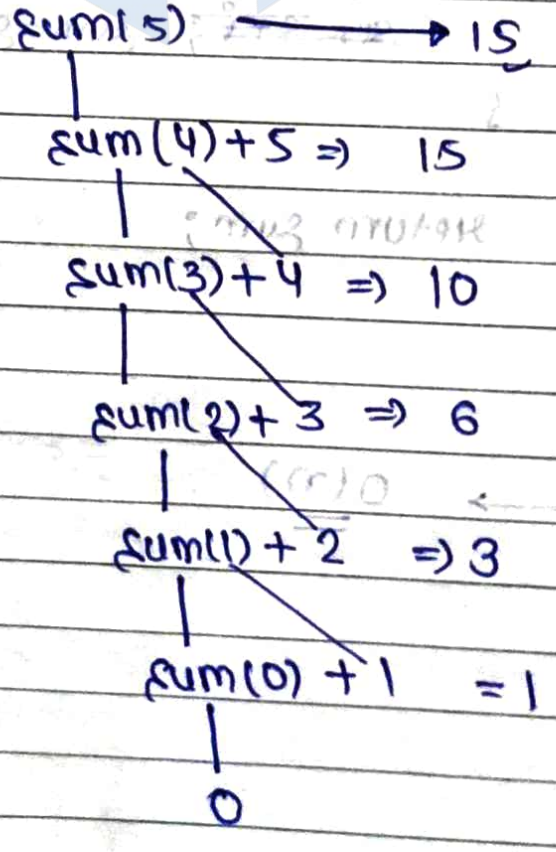
```
int sum(int n)
{
    if (n == 0)
        return 0;

    else
        return sum(n+1) + n;
}
```

O(n+1)

Time C. = O(n)

space = n+1
= O(n)

sum(5)



Notebook
Pg. No.
Date / /

Factorial of a number

factorial representation (n!)

$n! = 1 \times 2 \times 3 \times \dots \times n$

$5! = 5 \times 4 \times 3 \times 2 \times 1 \Rightarrow 120$

$0! = 1$

$1! = 1$

$fact(n) = 1 * 2 * 3 * \dots (n-1) * n$

$fact(n) = fact(n-1) * n$

$$\text{int } fact(n) = \begin{cases} 1 & n = 0 \\ fact(n-1) * n & n > 0 \end{cases}$$

```
int fact(int n)
{
    if(n==0)
        return 1;
    else
        return fact(n-1)*n;
}
```

space ⇒ O(n)
time ⇒ O(n)

Power using Recursion

$2^5 = \underbrace{2 \times 2 \times 2 \times 2}_{4} \times 2$

$m^n = m \times m \times m \times m \times \ldots$ for n times

$pow(m,n) = m \times m \times \ldots \times (n-1)$ times $\times m$

$pow(m,n) = pow(m,n-1) * m$

$$pow(m,n) = \begin{cases} 1 & n = 0 \\ pow(m,n-1) * m & n > 0 \end{cases}$$

```
int Pow(int m, int n)
{
    if(n==0)
    {
        return 1;
    }
    else
    {
        return pow(m,n-1)*m;
    }
```

for

pow(2, 9) → 512

pow(2,8) * 2 ⇒ 512

pow(2,7) * 2 = 256

pow(2,6) * 2 ⇒ 128

pow(2,5) * 2 = 64

pow(2,4) * 2 = 32

pow(2,3) * 2 = 16

pow(2,2) * 2 = 8

pow(2,1) * 2 = 4

pow(2,0) * 2 = 2

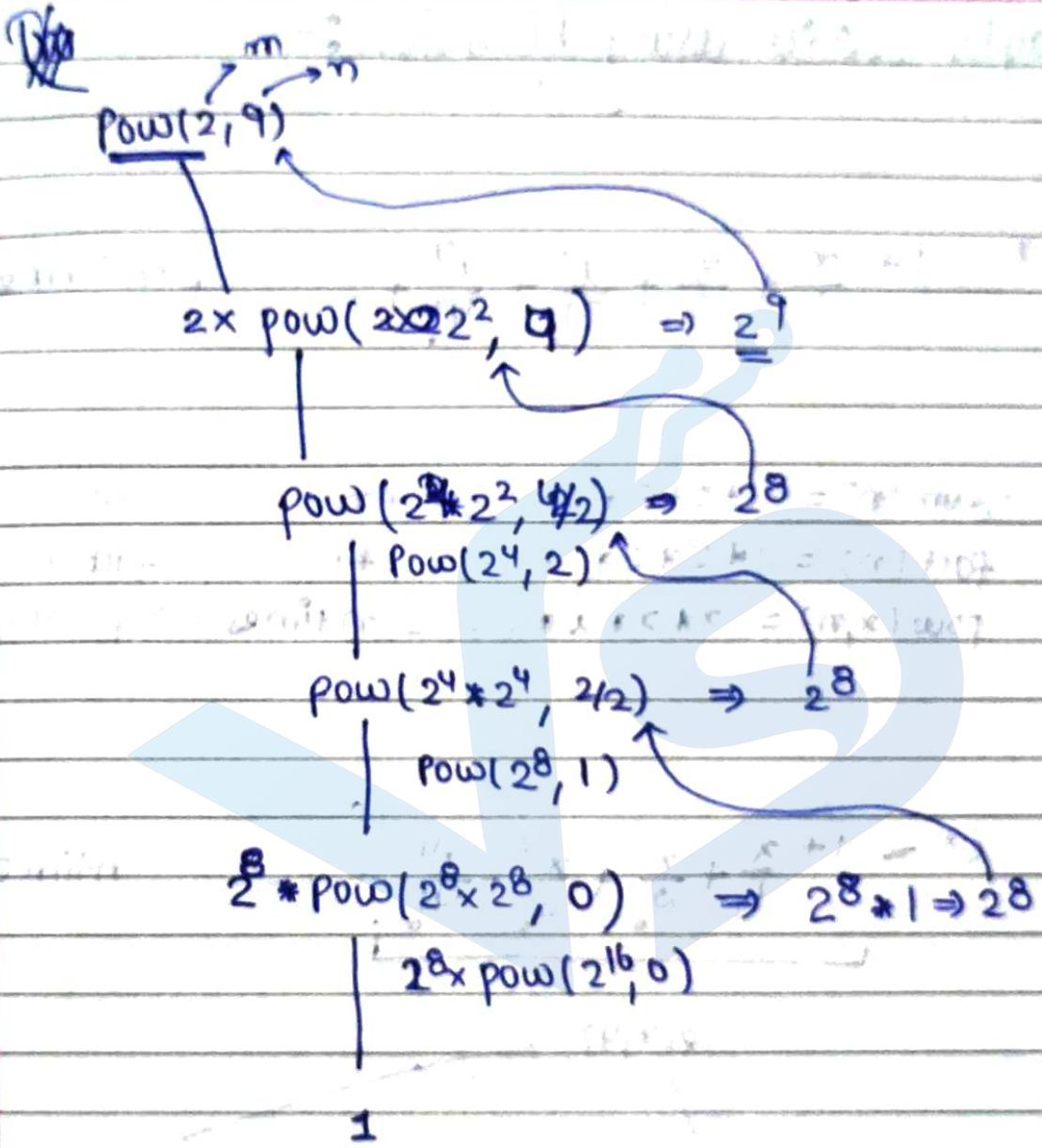
$2^8 = (2^2)^4$

$= (2*2)^4$

$= (4)^4$

$2^9 = 2^1 \times (2^2)^4$

```
int pow(int m, int n)
{
    if(n==0)
    {
        return 1;
    }
    else if (n%2==0)
    {
        return pow(m*m, n/2);
    }
    else
    {
        return m* pow(m*m, (n-1)/2));
    }
}
```

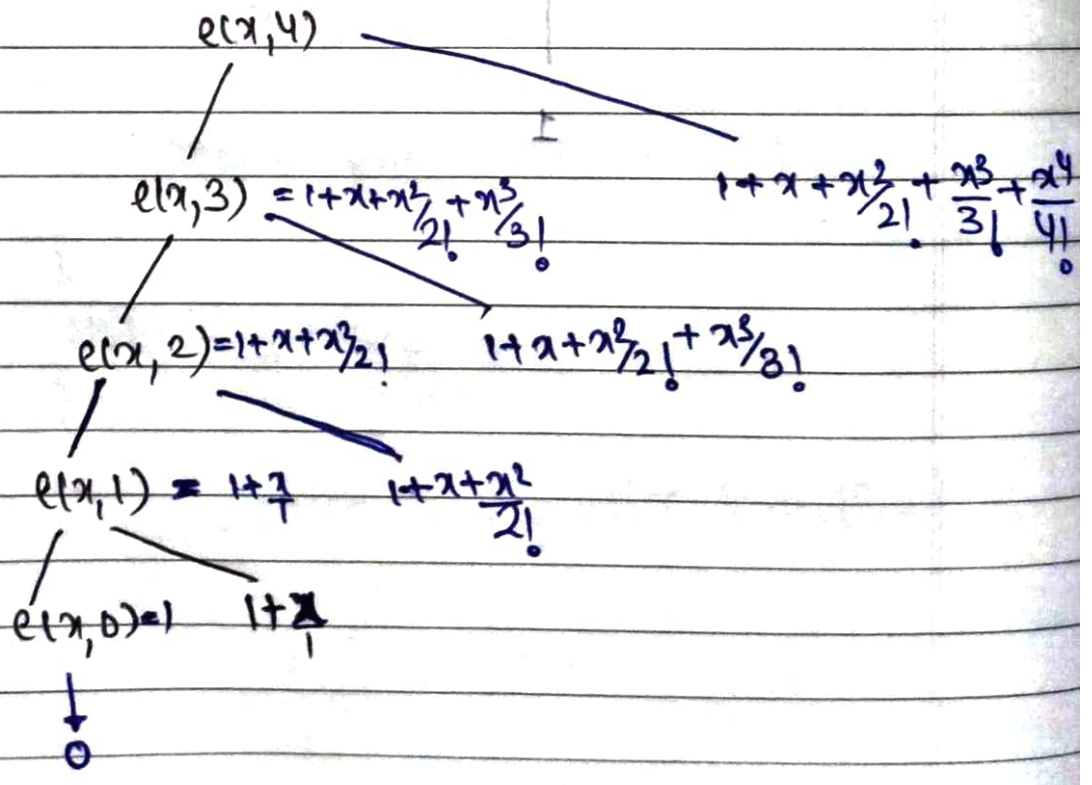

m
n
pow(2, 9)
2 × pow(2*2^2, 4)
⇒ 2^9
pow(2^4 * 2^4, 2/2)
⇒ 2^8
pow(2^4, 2)
pow(2^8, 1)
2^8 * pow(2^8 × 2^8, 0)
⇒ 2^8 * 1 ⇒ 2^8
2^8 × pow(2^16, 0)
1

Taylor Series using Recursion :-

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + n \text{ terms}$$

- sum(n) = 1+2+3+ ... +n → sum(n-1)+n
- fact(n) = 1*2*3* ... *n → fact(n-1)*n
- pow(x,n) = x*x*x* ... n times → pow(x,n-1)*x

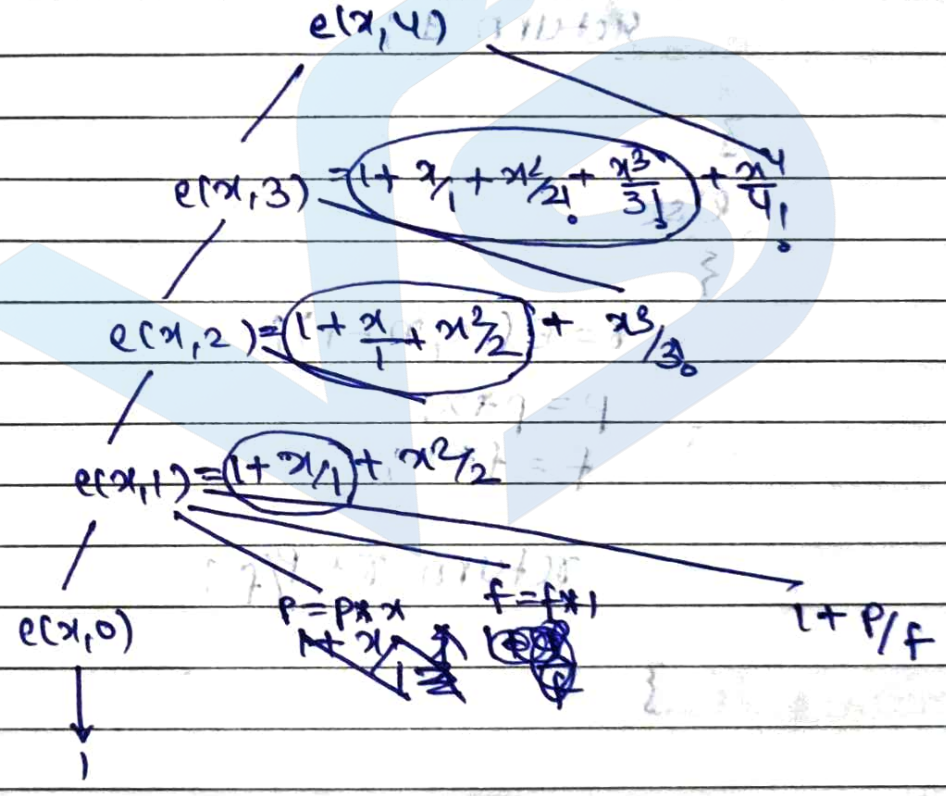
$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \text{ n times}$$



P F

x^0 1 1 01

x*x*x*x*1 * 2 * 3 * 4



```
int e(int x, int n)
{
    static int p=1, f=1;
    int r;

    if (n==0)
    {
        return 1;
    }
    else
    {
        r = e(x, n-1);
        p = p*x;
        f = f*n;

        return r + p/f;
    }

}
```

Notebook
Pg. No.
Date / /

Taylor Series using Horner's Rule :-

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + n \text{ terms}$$

Multiple By: 0 0 2 4 6

$$\frac{x \times x}{1 \times 2} \quad \frac{x \times x \times x}{1 \times 2 \times 3} \quad \frac{x \times x \times x \times x}{1 \times 2 \times 3 \times 4}$$

$$2 + 4 + 6 + 8 + 10 \ldots$$

$$2[1 + 2 + 3 + 4 \ldots]$$

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

$$= n^2 + 1$$

$$= O(n^2) \rightarrow n^2 \text{ Multiplication required}$$

Reducing Multiplication.

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{1*2} + \frac{x^3}{1*2*3} + \frac{x^4}{1*2*3*4}$$

$$= 1 + \frac{x}{1}\left[1 + \frac{x}{2} + \frac{x^2}{2*3} + \frac{x^3}{2 \times 3 \times 4} + \cdots\right]$$

$$1 + \frac{x}{1}\left[1 + \frac{x}{2}\left(1 + \frac{x}{3} + \frac{x^2}{3*4}\right)\right]$$ $O(n^2)$ Quadratic

$$1 + \frac{x}{1}\left[1 + \frac{x}{2}\left(1 + \frac{x}{3}\left(1 + \frac{x}{4}\right)\right)\right]$$ $O(n)$ ↳ Linear

Recursive function

```
int e(int x, int n)
{
    int s=1;

    for(n>0; n--)
    {
        s=1+ x/n * s;
    }
    return s;

}
```

```
int e(int x, int n)
{
    static int s=1;
    if(n==0)
    {
        return s;
    }
    else
    {
        s = 1+x/n*s;
        return e(x,n-1);
    }
}
```

Fibonacci Series Using Recursion - Memorization

fibonacci = 0 1 1 2 3 5 8 13 .....

index 0 1 2 3 4 5 6 7

$$fib(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2) + fib(n-1) & n>1 \end{cases}$$

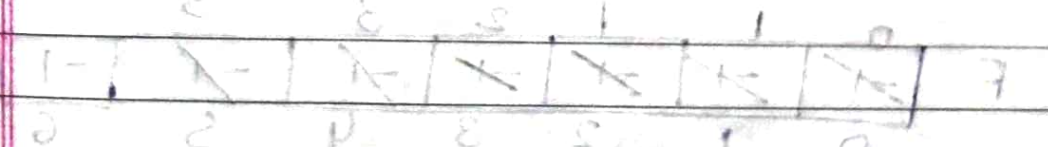
```
int fib(int n)
{
    if (n <= 1)
    {
        return n;
    }
    else
    {
        return (fib(n-2) + fib(n-1));
    }
}
```

Iterative version

```
int fib(int n)
{
    int a=0, b=1, sum;          —1
    if(n<=1)
    {
        return n;               —1
    }
    else
    {
        for(i=2; i<=n; i++)     —n
        {
            s = to + t1;        — n-1
            to = t1;            —— n-1
            t1 = s;             —— n-1
        }
        return s;               —1
    }
```

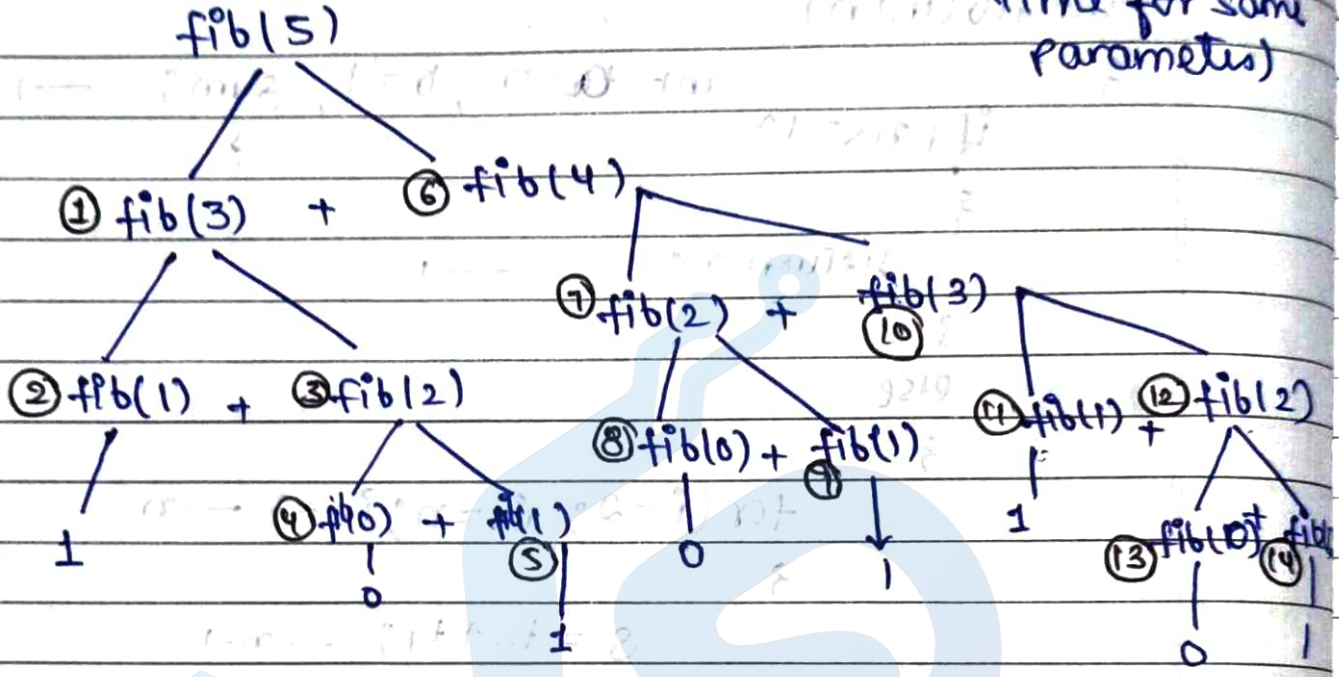
$T(n) = 4n - 1$

$O(n)$



Notebook
Pg. No.
Date / /

Excessive Recursion (Call multiple time for same parameters)

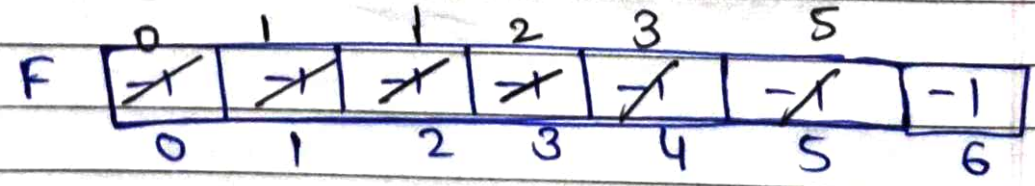


fib(5) → Number of calls 15

fib(4) → Number of calls 9

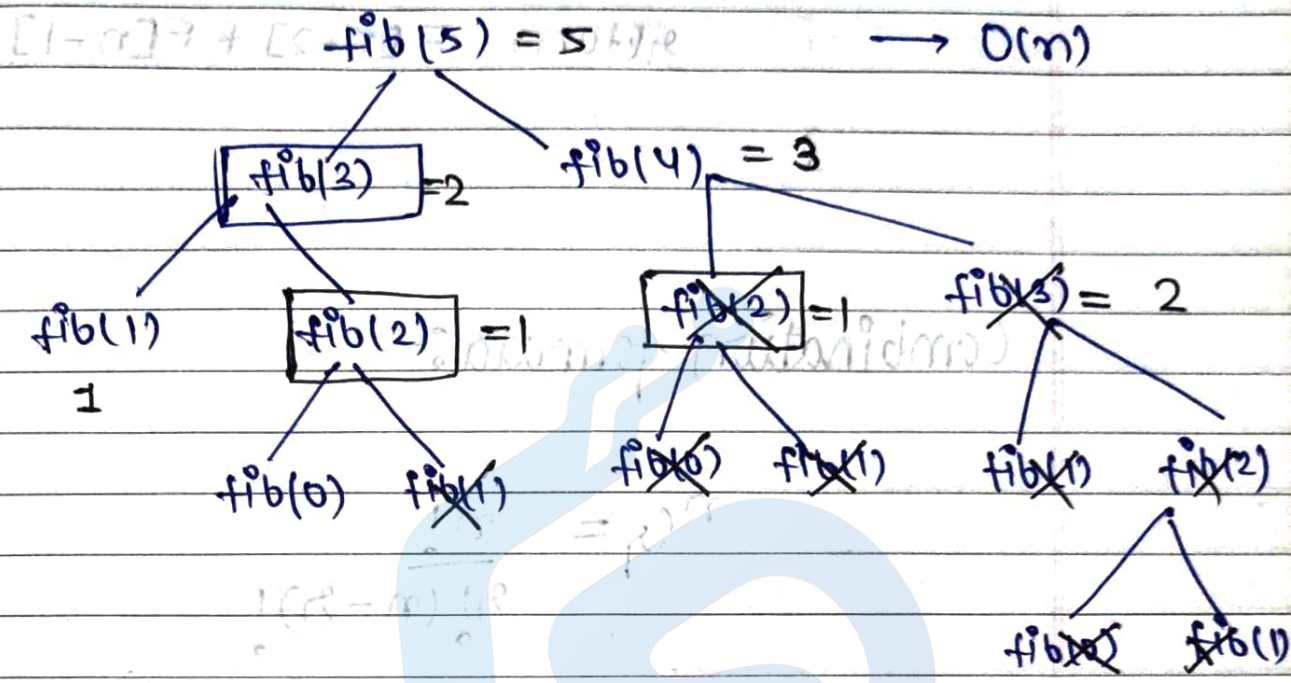
fib(3) → Number of calls 5

$O(2^n) \rightarrow 2fib(n-1)$



↓

Approaches known as Memorization



function for memorization:-

```
int f[10]
int fib(int n)
{
    if (n<=1)
    {
        f[n] = n;
        return n;
    }
    else
    {
        if (f[n-2] == -1)
        {
            F[n-2] = fib(n-2);
        }
        if (F[n-1] == -1)
        {   F[n-1] = fib(n-1);
        }
```

return F[n-2] + F[n-1];

}

Combination formula:-

$$^{n}C_{r} = \frac{n!}{r!(n-r)!}$$

A B C D E F G

↑

select 3 item

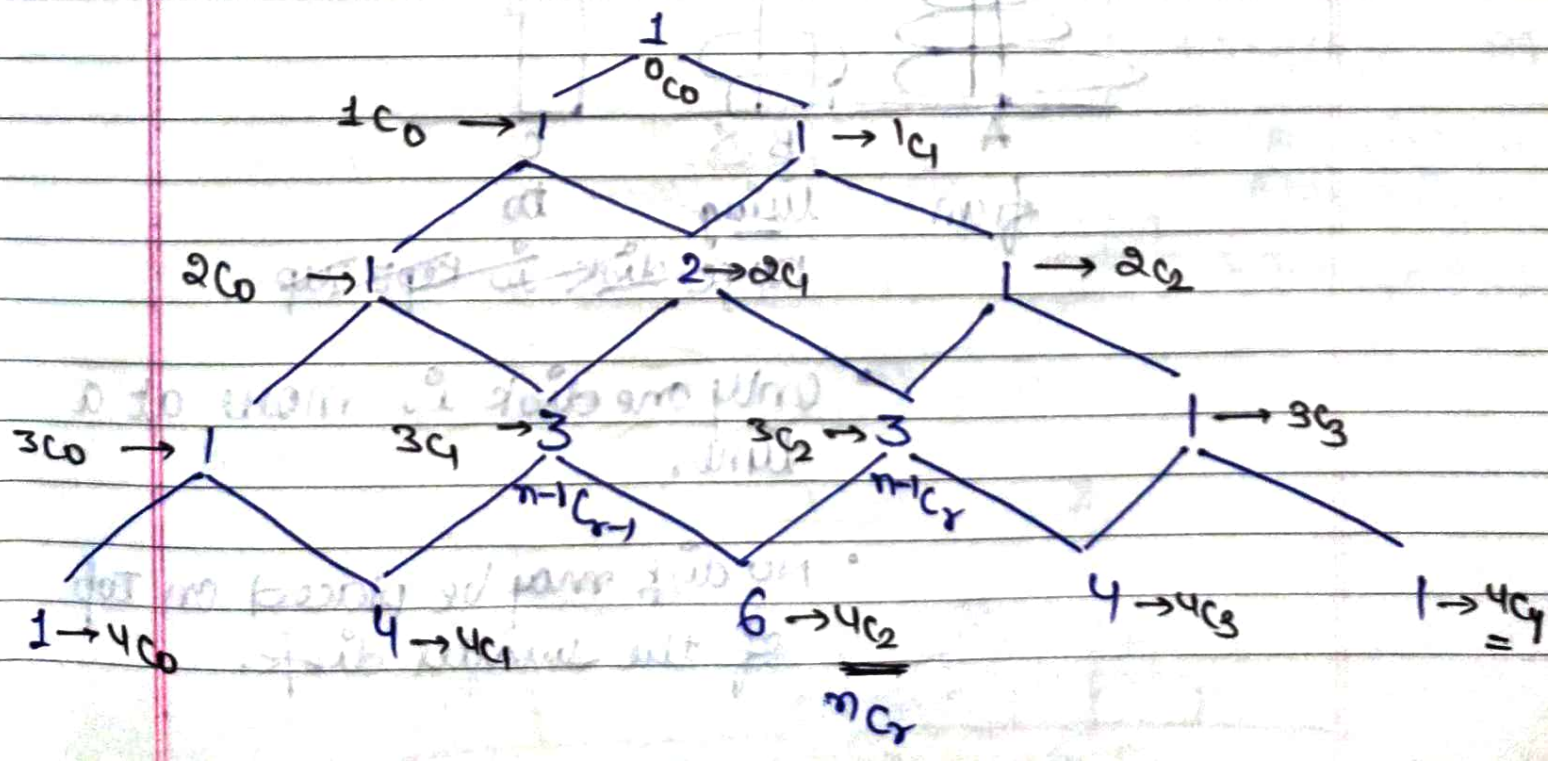
ABC
ABD
~~ACB (same)~~

$^{5}C_{0-5} = {}^{5}C_{0}\ {}^{5}C_{1}\ {}^{5}C_{2} \ldots\ {}^{5}C_{5}$

```
int C (int n, int r)
{
    int t1, t2, t3;

    t1 = fact(n);   ——— n
    t2 = fact(r);   ——— n
    t3 = fact(n-r); ——— n

    return t1/(t2*t3); ——— 1

}
```

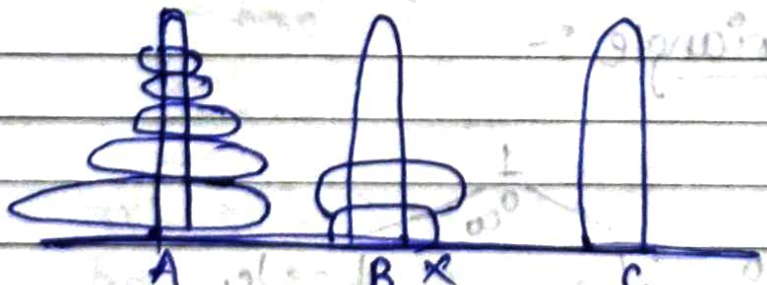
$O(n)$

Pascal triangle :-



```
int C (int n, int r)
{
    if (r == 0 || n == r)
    {
        return 1;
    }
    else
    {
        return C(n-1, r-1) + C(n-1, r);
    }
```

Towers of Hanoi



from using to

~~Large disk is kept top~~

- Only one disk is move at a time.
- No disk may be placed on top of the smaller disk.

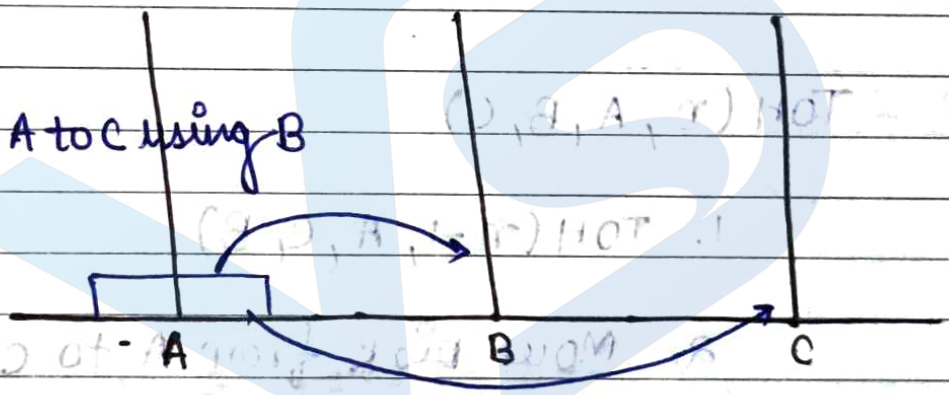
Notebook
Pg. No.
Date / /

- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

1 disk :-

TOH(1, A, B, C)

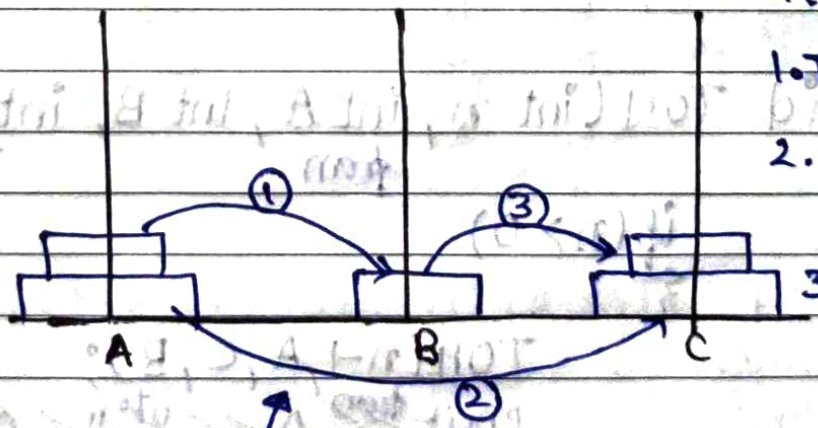
Move Disk from A to C using B



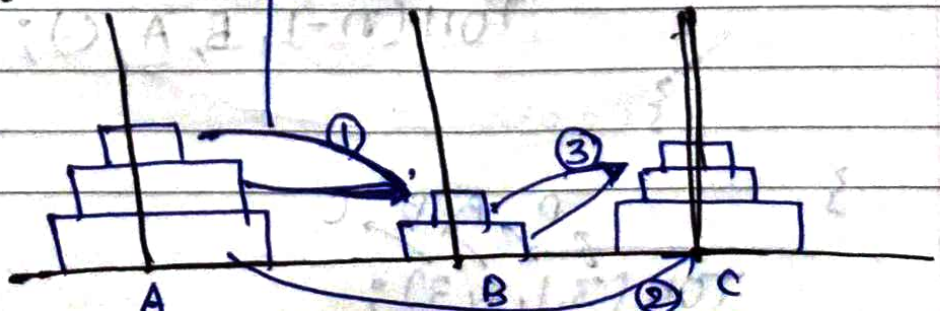
2 Disk :-

TOH(2, A, B, C)

1. TOH(1, A, C, B)
2. Move Disk from A to C using B
3. TOH(1, B, A, C)



3 disk :-



TOH(3, A, B, C)

1. TOH(2, A, C, B)
2. ~~TOH~~ Move Disk from A to C using B
3. TOH(2, B, A, C)

TOH(n, A, B, C)

1. TOH(n-1, A, C, B)
2. Move Disk from A to C using B.
3. TOH(n-1, B, A, C);

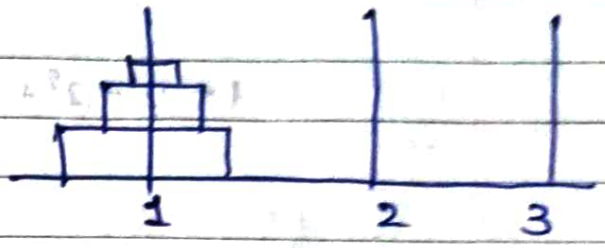
```
void TOH(int n, int A, int B, int C)
{                    from        to
    if(n>0)
    {
        TOH(n-1, A, C, B);
        cout << A << "to" << C;
        TOH(n-1, B, A, C);
    }
}
```

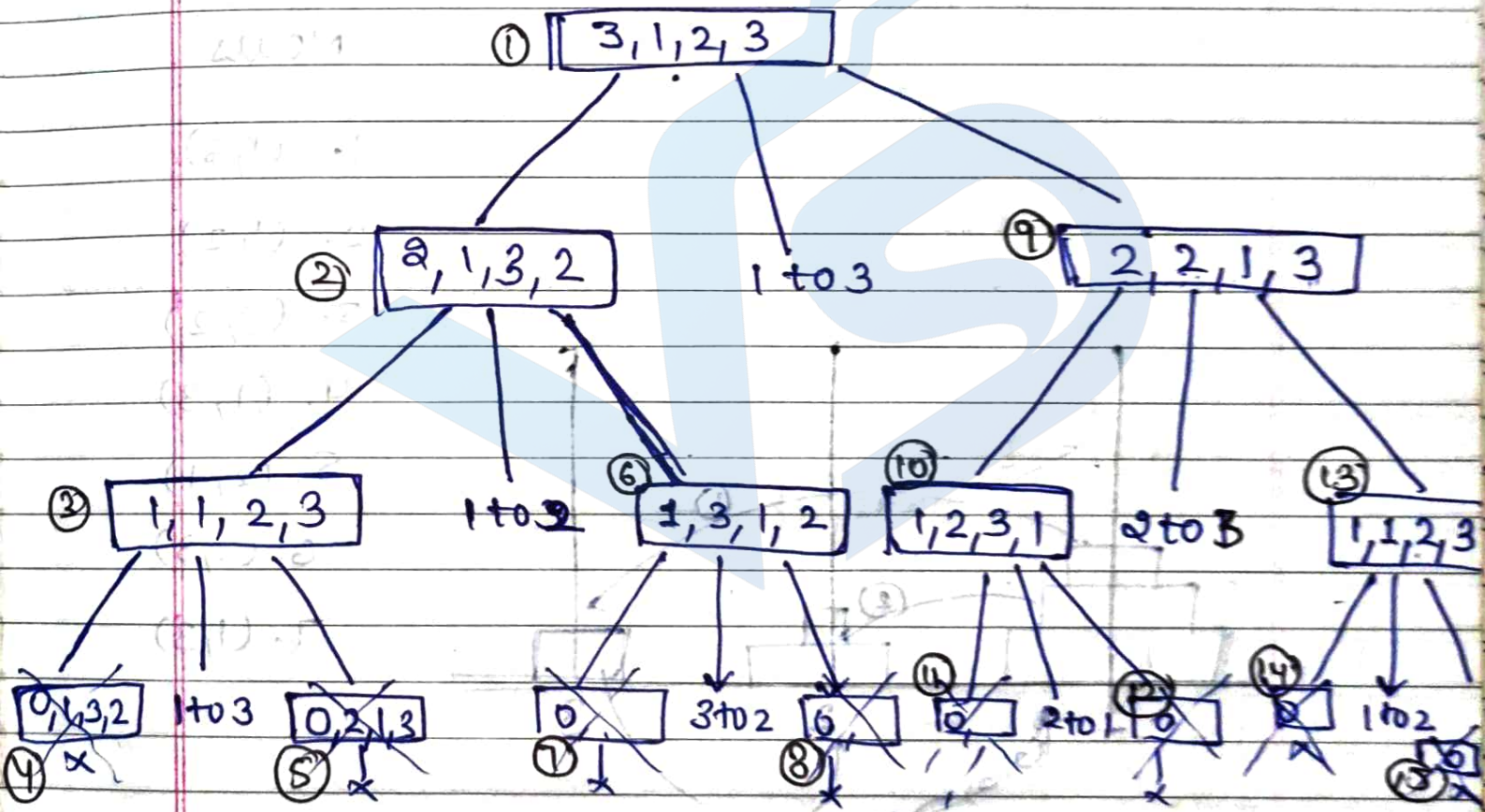
n A B C

TOH(3, 1, 2, 3);

Pg. No.
Date / /

n = 3
A = 1
B = 2
C = 3





(1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,2)

Disk = 3 → No. of calls = 15 → $1 + 2 + 2^2 + 2^3 \Rightarrow 2^4 - 1$

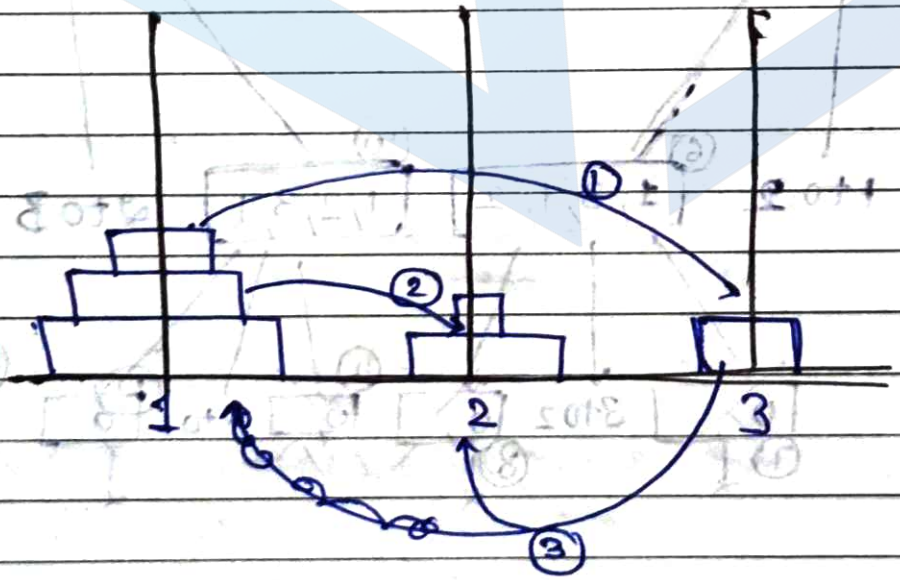
Disk = 2 → No. of calls = 7

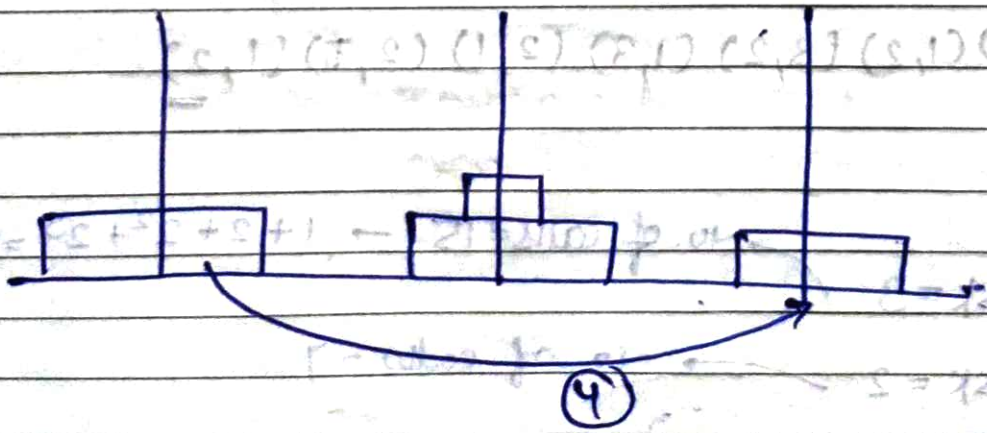
$$1 + 2 + 2^2 + \cdots 2^n = 2^{n+1} - 1$$

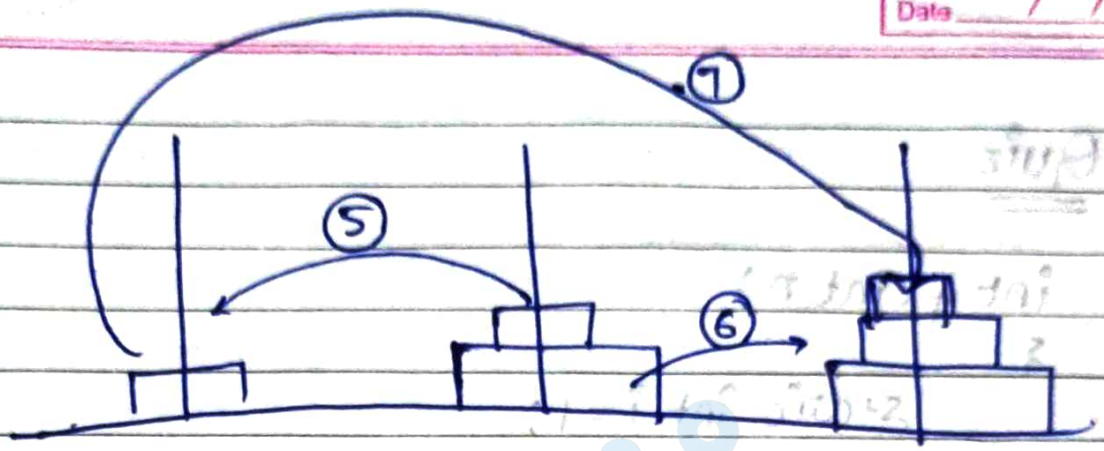
$$\therefore O(2^n)$$

MOVES

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






7
5
6

Notebook
Pg. No.
Date / /

Quiz

1.
```
int f(int n)
{
    static int i=1;
    if(n>=5)
    {
        return n;
    }
    else n = n+i
    *   i++;
    return f(n);
}   }
```

i=2
n=4
i=3
n=7

the value returned by f(1) is

→ Answer is 7

2.
```
void foo(int n, int sum)
{
    int k=0, j=0;
    if(n==0) return
    k = n%10
    j = n/10
    sum = sum+k
    foo(j, sum)
    cout<<k;
}
```

k=8
j=0
sum=8

```
int main()
{
    int a=2048, sum=0;
    foo(a,sum);
    cout<<sum;

}
```

what does the above program print?

→ First Iteration

↳ a=2048

n=2048, sum=0

k=8 → 8

j=204

sum = 8

foo(204, 8) → 4

n=204 sum=8

k=4 j=20

sum = 8+4 = 12

foo(20, 12) → 2

n=20 sum=12

k=2 j=2

sum = 12+2 = 14

foo(2, 14)

n=2 sum=14

k = 0
j = 0

sum = 14

foo(0, 14)

n=0 14

~~return~~ return;

2, 0, 4, 8, 0

- what is the return value of f(P, P), if the value of p is initialised 5 before the call? Note that the first parameter is passed by reference, whereas the second parameter is passed by value.

```
int f(int x, int c)
{
    c = c-1;
    if(c==0) return 1;
    x = x+1;
    return f(x,c)*x;
}
```

→ f(5,5)

c=4

x=5+1

x=6

return f(6,4)*6; ⇒ 30240

x=6 c=4

c=3

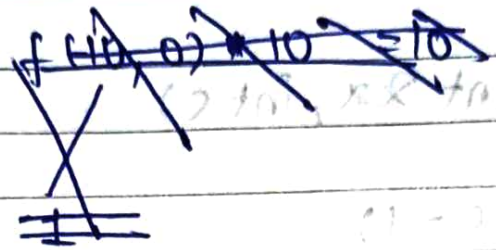
x=7

return f(7,3)*7; ⇒ 5040

f(8,2)*8 → 720

f(9,1)*9 = 90

Answer



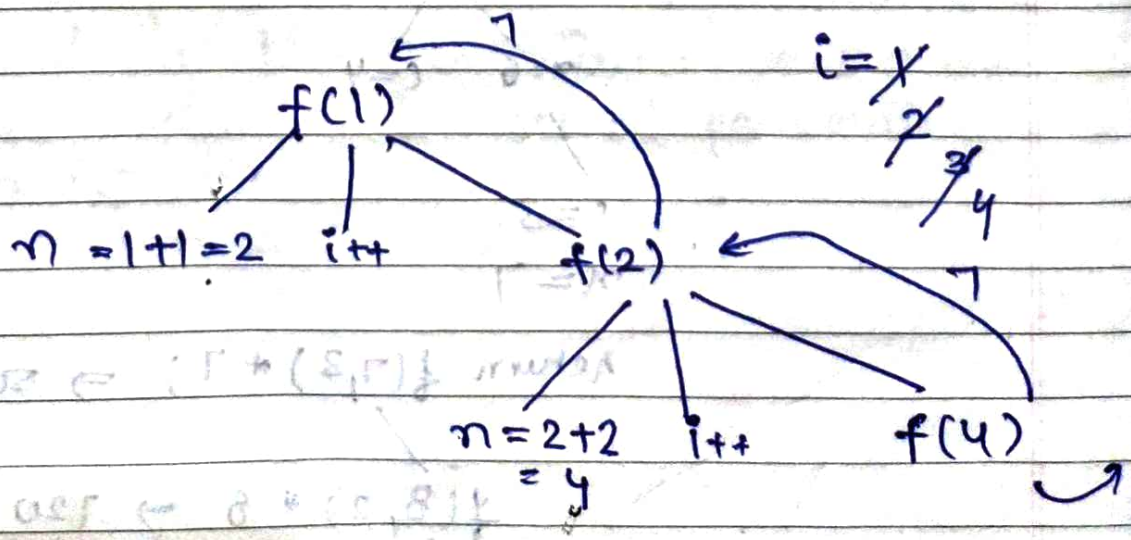
Section 15 (Recursion) → Quiz

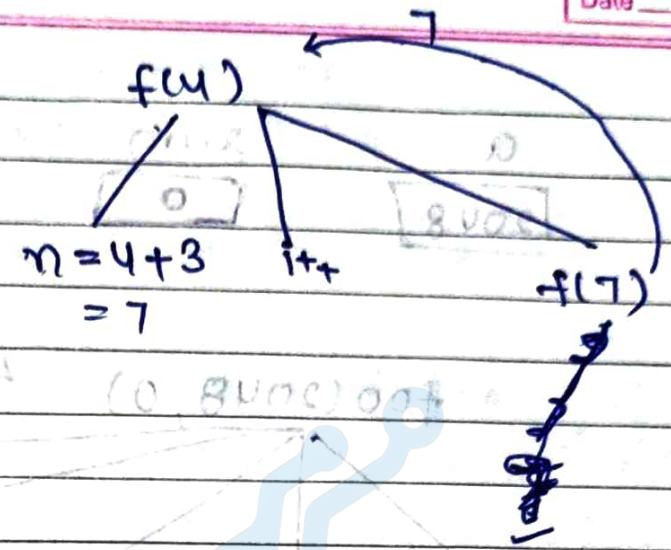
Q.

```
int f(int n)
{
    static int i=1;
    if(n>=5)
    {
        return n;
    }
    n = n+i;
    i++;
    return f(n);
}
```

the value of returned f(1) = ?

→

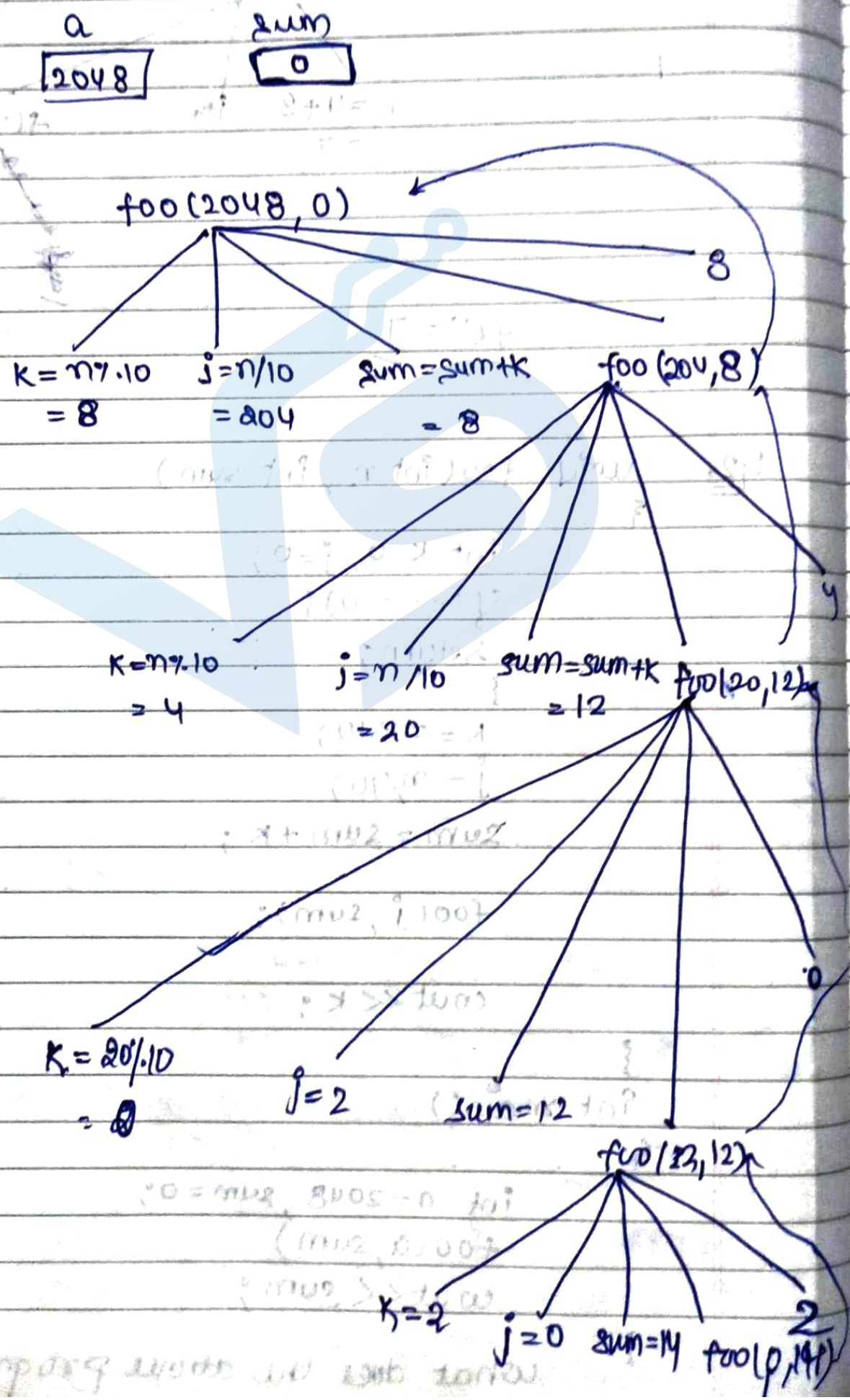




f(1) = 7

Q2.
```
void foo(int n, int sum)
{
    int k=0, j=0;
    if(n==0){
    return;
    }
    k = n%10;
    j = n/10;
    sum = sum + k;

    foo(j, sum);

    cout << k;
}
int main()
{
    int a=2048, sum=0;
    foo(a, sum)
    cout<< sum;
}
```

What does the above program print?


a
2048
sum
0
foo(2048, 0)
8
K = n%.10
= 8
j = n/10
= 204
sum = sum+k
= 8
foo(204, 8)
4
K = n%.10
= 4
j = n/10
= 20
sum = sum+k
= 12
foo(20, 12)
0
K = 20%.10
= 0
j = 2
sum = 12
foo(2, 12)
K = 2
j = 0
sum = 14
foo(0, 14)
2

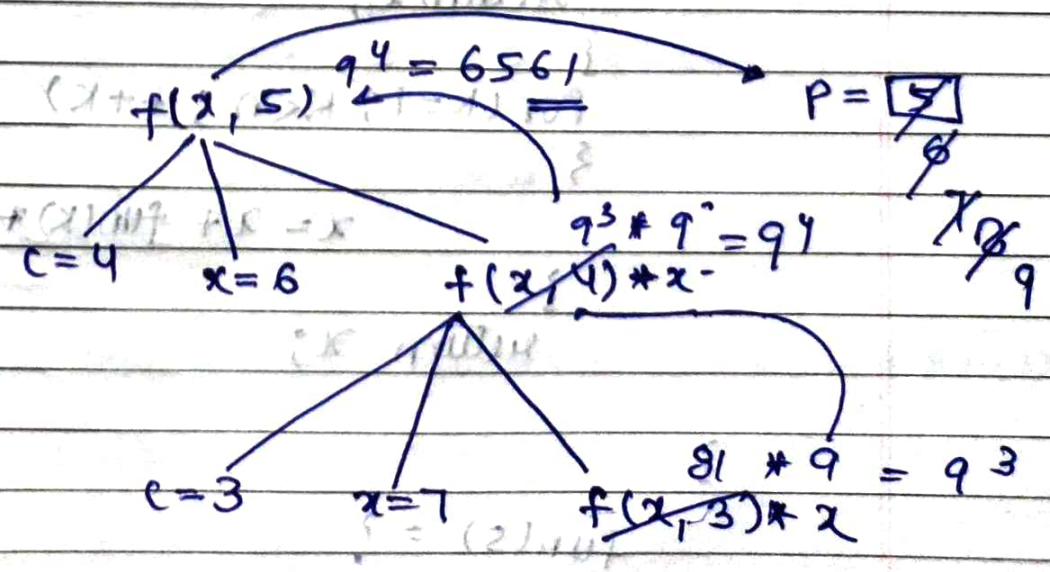
Output 20480

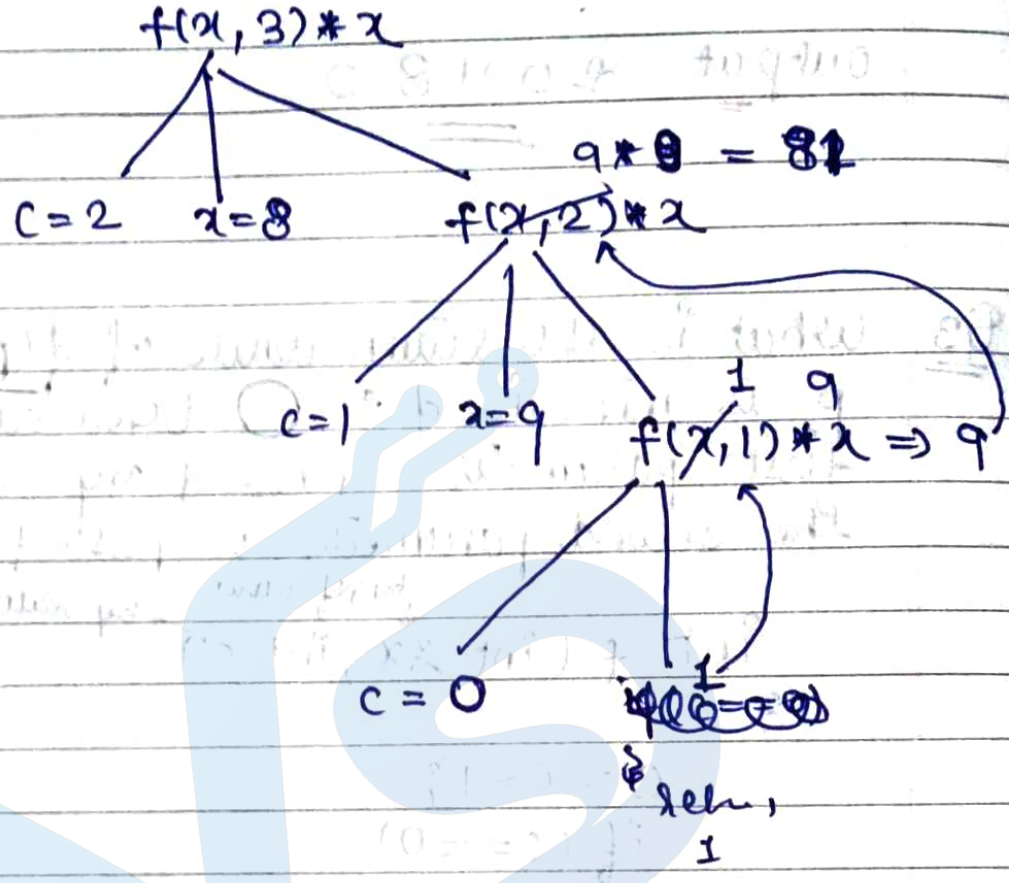
Q3 What is the return value of f(p,p), if the value of p is initialised to (5) before call? Note that the first parameter is passed by reference, whereas the second parameter is passed by value.

by reference ↓ , by value →

```
int f(int &x, int c)
{
    c = c-1;
    if (c==0)
    {
        return 1;
    }
    x = x+1;
    return f(x,c)*x;
}
```

→





Q4.
```
int fun(int n)
{
    int x=1, k;
    if(n==1){
    return x;
    }
    for(k=1; k<n; ++k)
    {
        x= x+ fun(k)* fun(n-k);
    }
    return x;
}
```

fun(5) = ?

fun(5) = 51

Notebook
Pg. No.
Date / /

fun(5)

x = x + fun(1) * f(4) + fun(2) * fun(3) + fun(3) * fun(2)
+ fun(4) * fun(1)

= 1 + 1 * 5 + 2 * 5 + 5 * 2 + 5 * 1

= 1 + 5 + 10 + 10 + 5 ⇒ 51

32
20
52

n	1	2	3	4	5
fun(n)	1	2	5	16	51

fun(2)

x = x + fun(1) * fun(1)

= x + 1 * 1

= x + 1

= 1 + 1

= 2

fun(3)

x = x + fun(1) * fun(2) + fun(2) * fun(1)

= 1 + 1 * 2 + 2 * 1

= 5

fun(4)

x = x + fun(1) * fun(3) + fun(2) * fun(2) + fun(3) * fun(1)

= 1 + 1 * 5 + 2 * 2 + 5 * 1

= 1 + 5 + 4 + 5

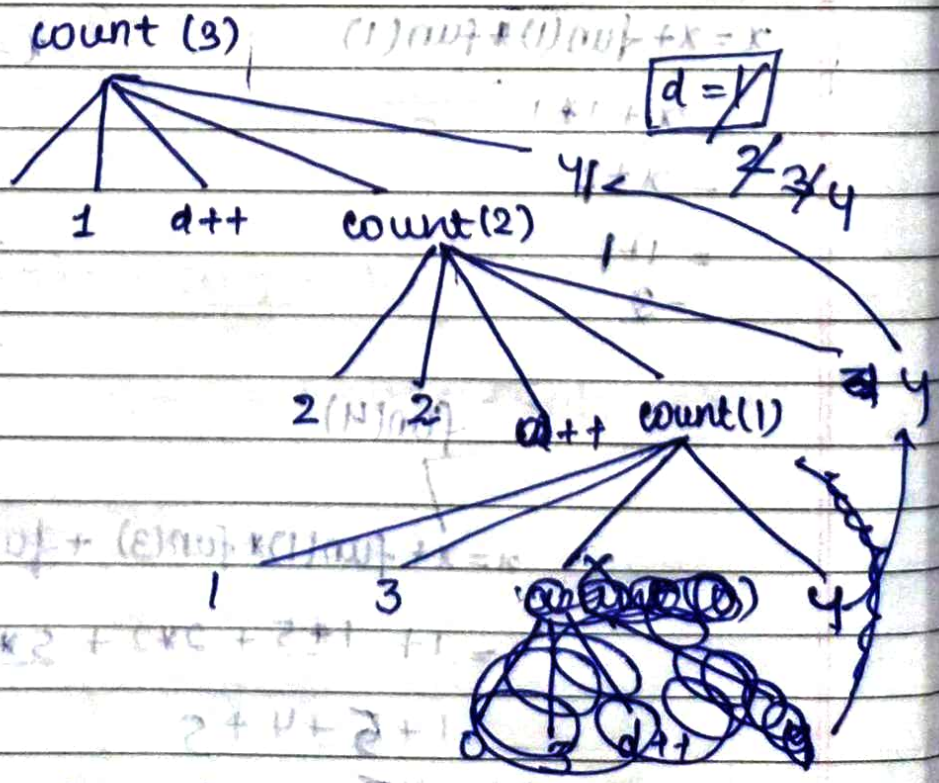
= 15

Q5.

```
void count(int n)
{
    static int d=1;
    cout<<n;
    cout<<d;
    d++;
    if(n>1)
    {
        count(n-1);
    }
    cout<<d;
}
```

count(3) = ?

→



Output :- ~~3122044~~ 3123444