UDEMY ALGO

Section - 5 = RECURSION

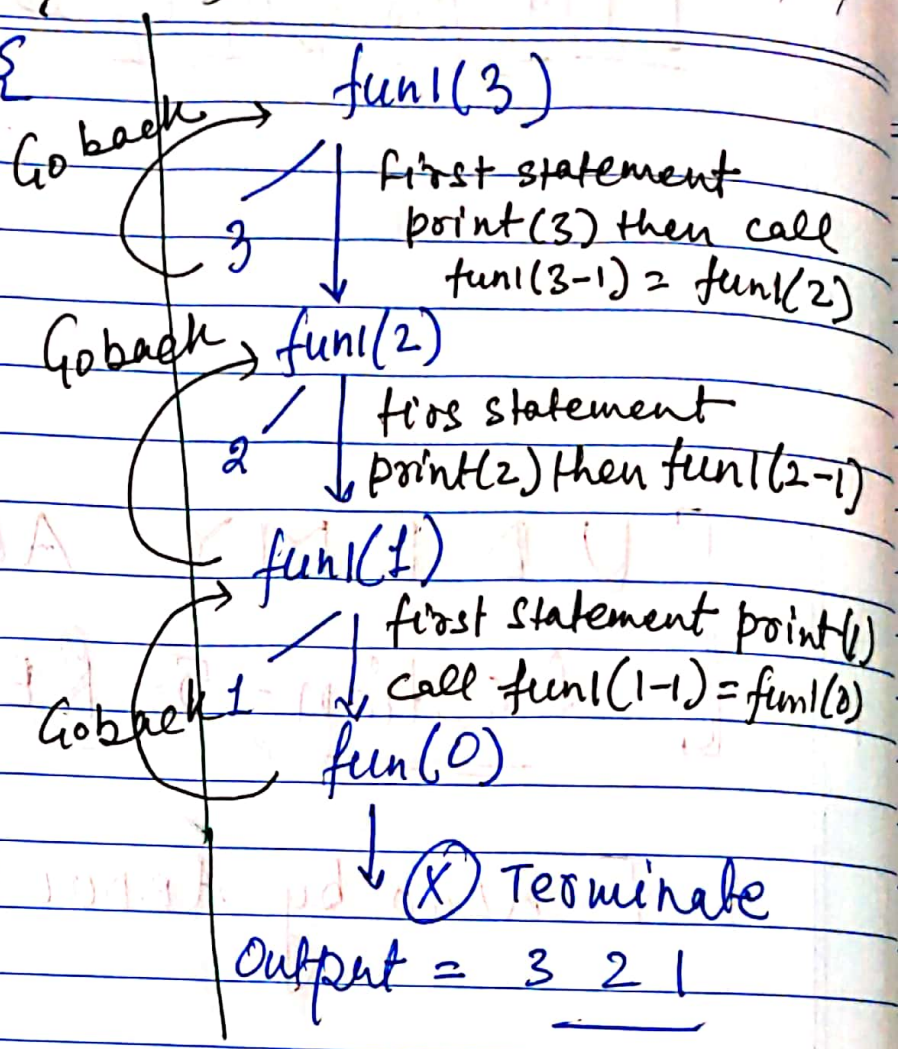
[ALGO by ABDUL BARI]

P1

```
void fun1(int n) {
  if (n>0)
  {
    print(n);
    fun1(n-1);
  }
}
void main() {
  int x=3;
  fun1(x);
}
```

Trace Tree for P1

Date: / /



output = 3 2 1

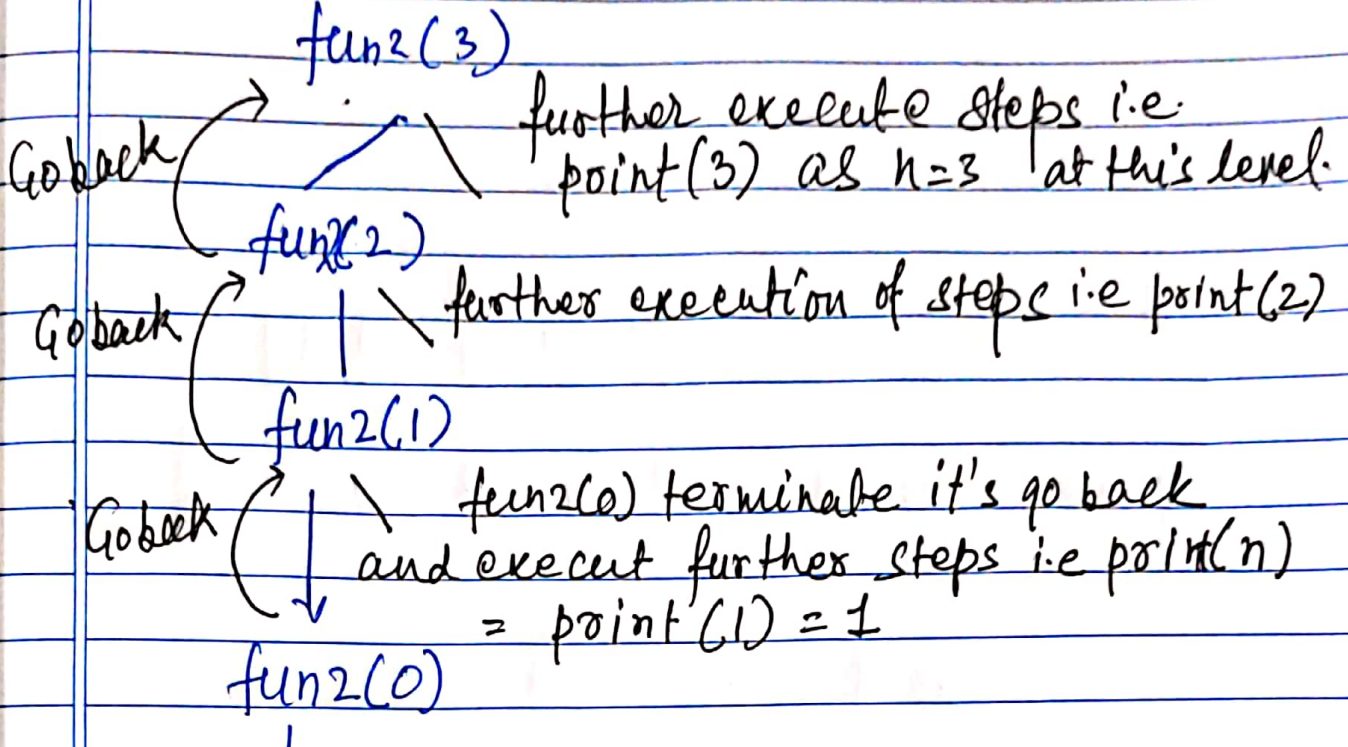
P2

```
void fun2(int n){
  if(n>0) {
    fun2(n-1);
    print(n);
  }
}
```

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

Date / / /

TraceTree P2 :-



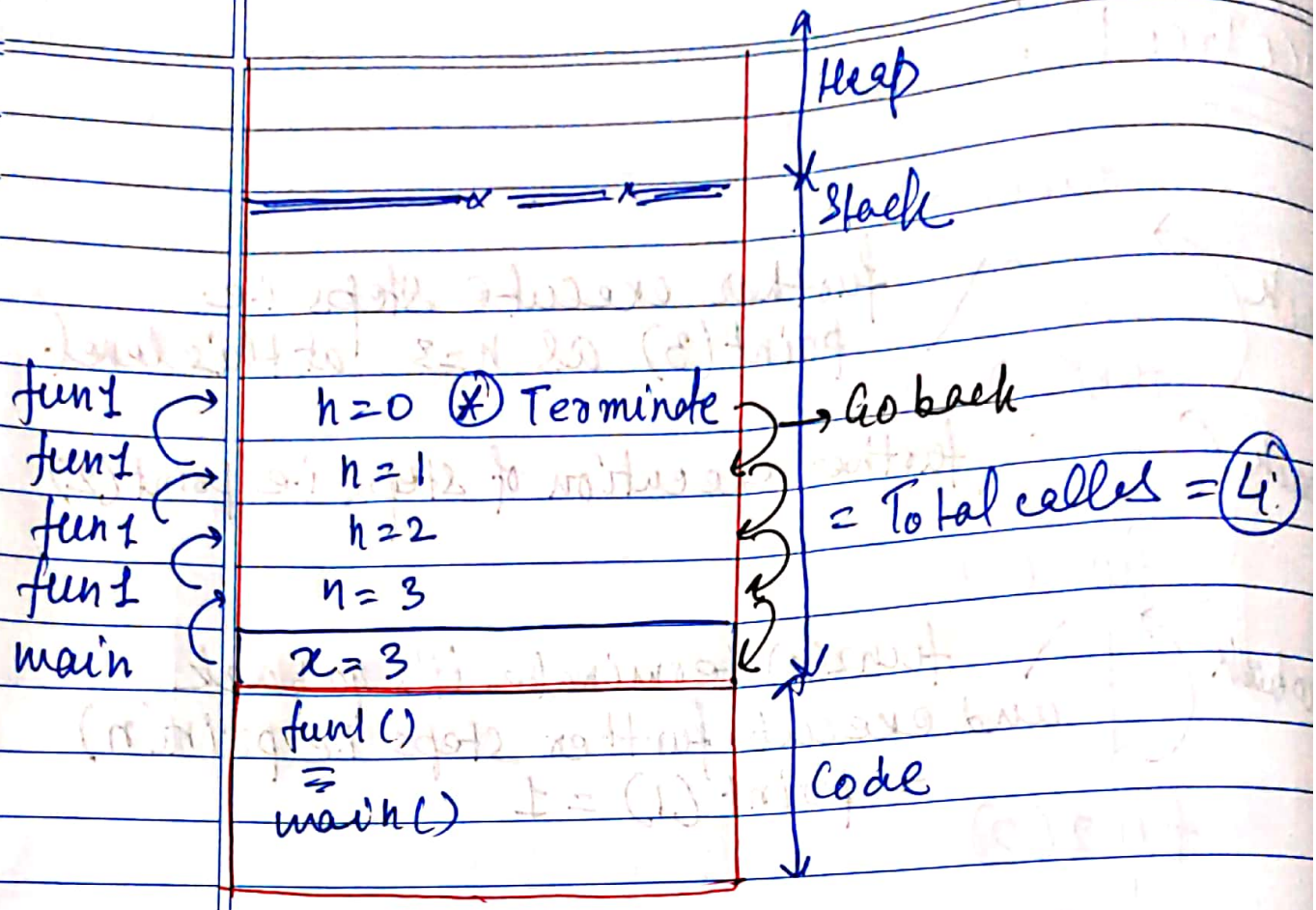

```
void fun(int n) {
  if(n>0) {
    statement 1
    statement 2
    ...
    fun(n-1)*2
    statement 6
    statement 7
    statement 8
  }
}
```

calling time

Returning time

In Recursion any statement before function call executed while CALLING. any statement after it executed returning time

Ex: fun(n-1)*2; multiply will execute after Return time.



(*) Memory view of P1 program

Time Complexity of Recursion :-

(P1)
```
void fun1(int n){ == assume T(n)
if(n>0) { → 1
print(n); → 1
fun1(n-1); → T(n-1)
{
}
```

Date: / / /

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

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

(as constant is treated as same)

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

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

$= T(n-2) + 2$

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

$= T(n-3) + 3$

$=$ ⋮

$= T(n-k) + k$

assume $n - k = 0$

$\hookrightarrow n = k$

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

$= T(0) + n$

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

Order of $O(n)$.

Date / /

Static Variable in Recursion

(P2)

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

```
main() {
int a = 5;
printf(fun(5));
}
```

Tracing Tree P3

fun(5) n=5, 10+n = 10+5 = 15

fun(4) + ___ n=4, 4+n = 4+6 = (10)

fun(3) + ___ n=3; 3+n = (6)

fun(2) + ___ n=2; 1+n = 1+2 = 3

fun(1) + ___ n=1; = 0+1 = 1

fun(0) = 0

f(4) or f(4) = 10 to goes back

6 Goes Back

3 Goes Back

1 Goes Back

0 zero Go back

Date: / / /

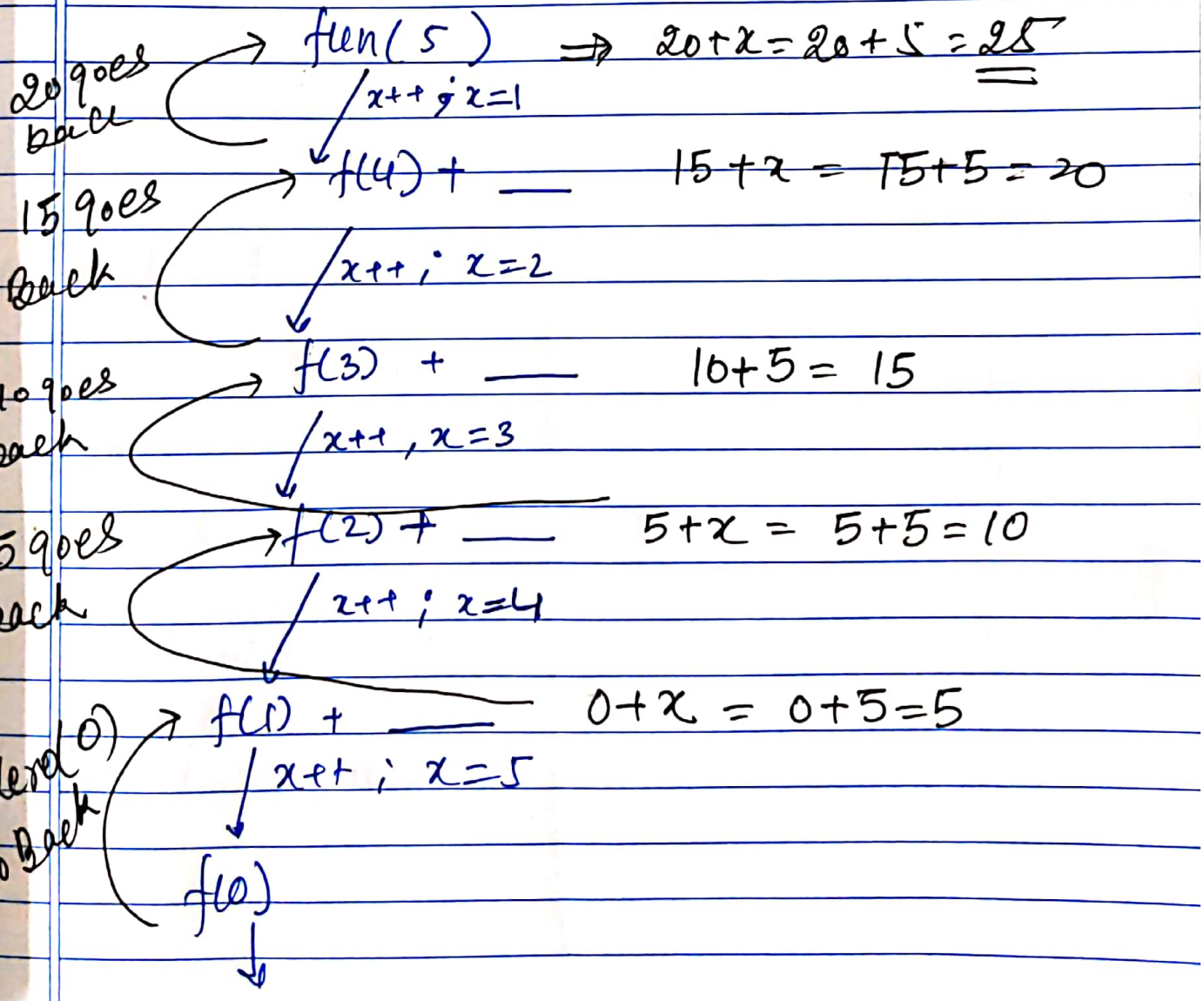
(4)

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

```
main(){
  int a=5;
  print(fun(5));
}
```

As x is static variable it will have single copy of x.

x = 0



Date / /

Types of Recursion :-

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

Tail Recursion :- Recursive call is last statement in the function

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

Recursive call is last statement.

Head Recursion :-

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

First statement is Recursive Call.

Nothing comes first.

Date: / / /

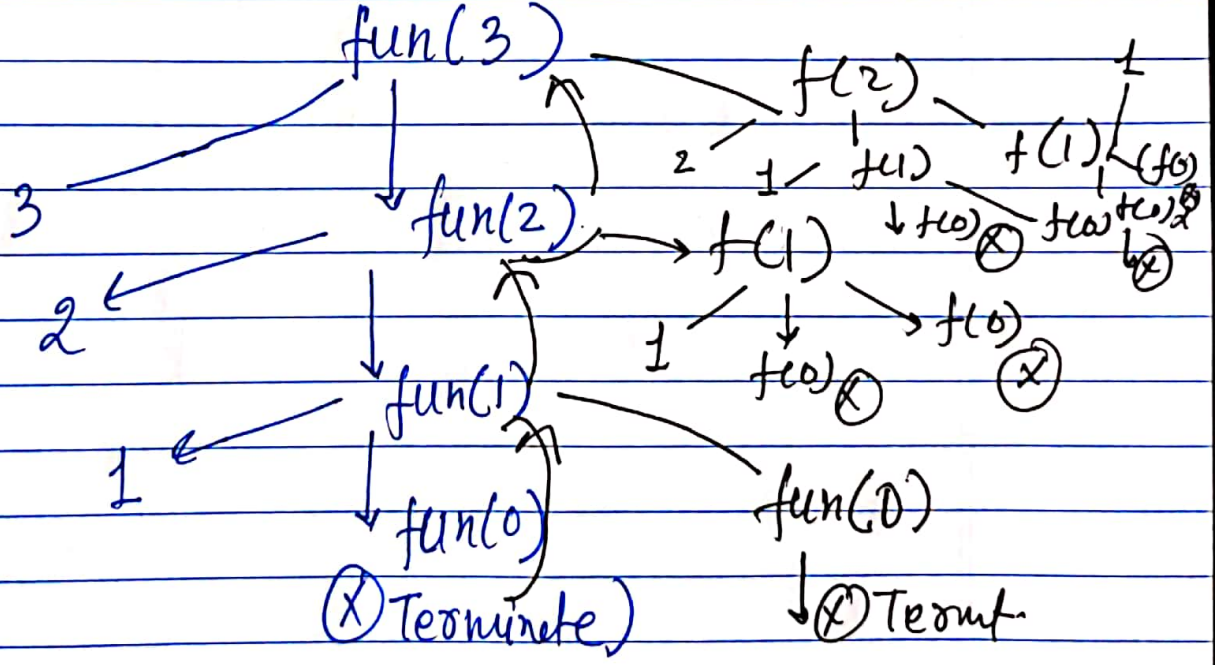
Tree Recursion:- If the call is more than one time

```
fun(n){
if(n>0){
  =
  - fun(n-1);
  =
  fun(n-2);
}=
```

More than one recursive calls.

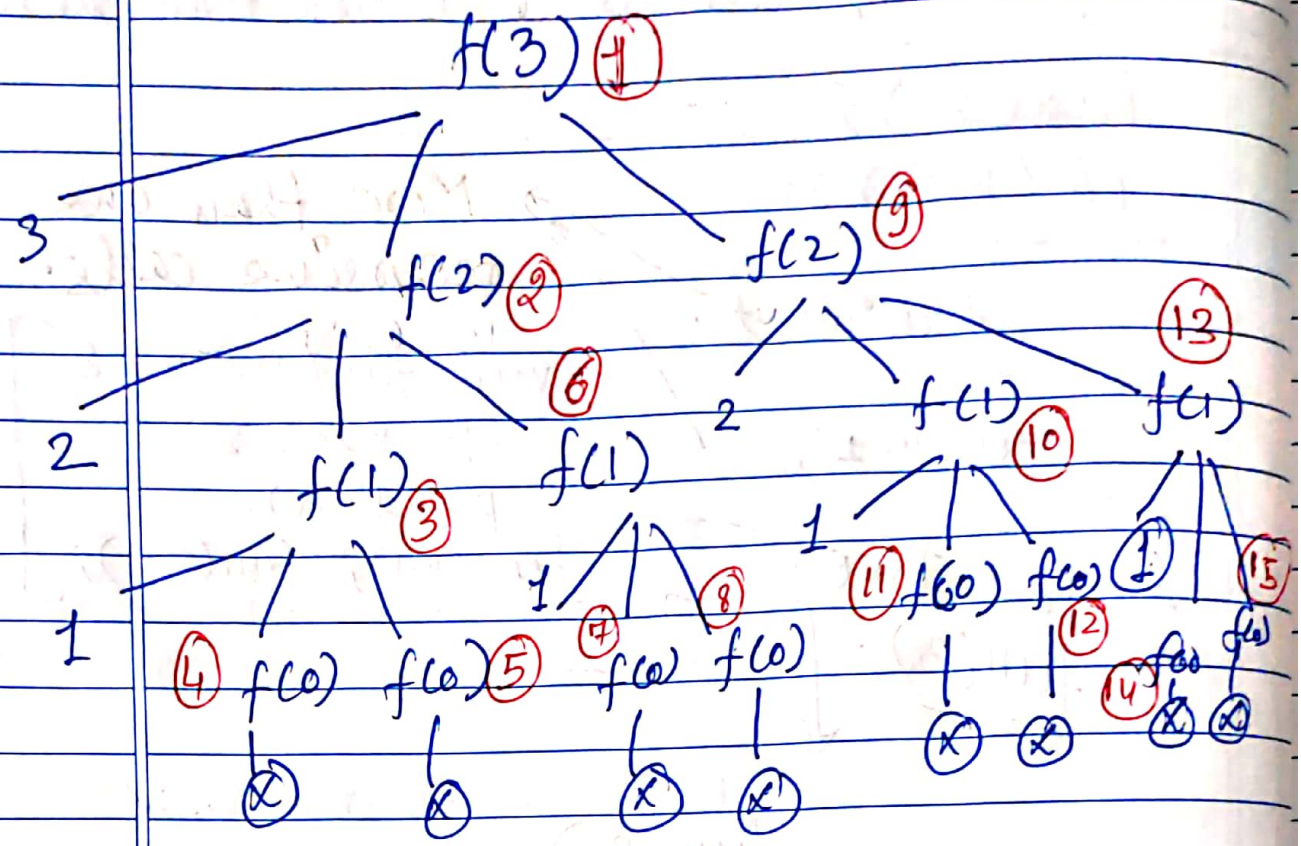
```
- void fun(int n){
  if(n>0){
  print(n)
  fun(n-1); fun(n-1);
  }
```

Trace Tree fun(3)



Output ⇒ 3, 2, 1, 1, 2, 1, 1

Date : / /



Detail Tree of P5. & calling sequence.

⑧ Sum of First "N" natural number :-

$S = 1 + 2 + 3 + \cdots (N-1) + N$

$S = sum(N-1) + N$

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

Date / / /

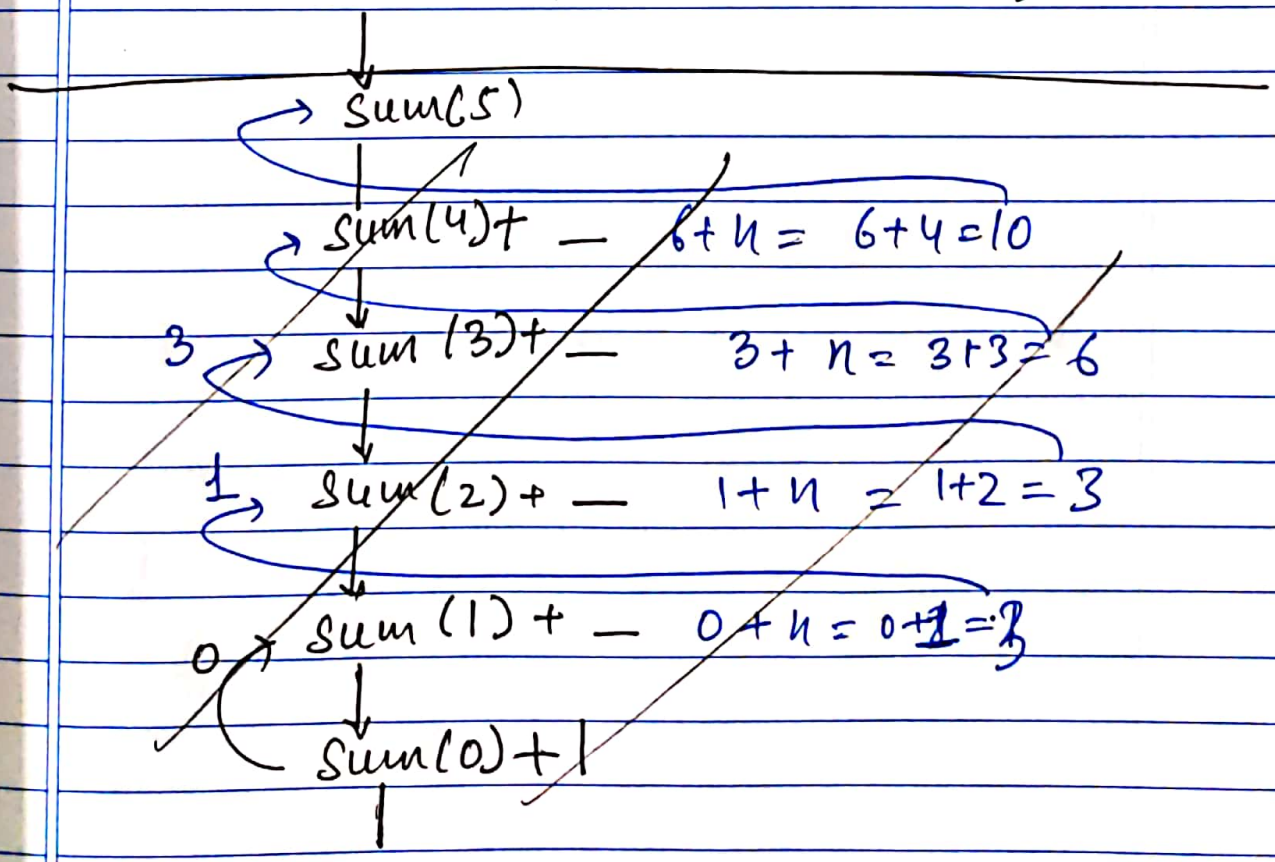
(8) Imp

We say for large value of "n" what is recurrence relation & for smaller value direct value is given

Using relation we can write program.

```
int sum(int n) {
    if (n==0)          condition for small value
        return 0;
    else
        return sum(n-1)+n;     } put recurrence relation
}
```

↓ Trace Tree sum(5)



Date: /

Trace Tree Sum(5)

Sum(5) → 15 =

10 → Sum(4) + 5 = 10 + 5

6 goes → Sum(3) + 4 = 6 + 4 = 10

3 goes → Sum(2) + 3 = 3 + 3 = 6

1 goes → Sum(1) + 2 = 1 + 2 = 3

back → Sum(0) + 1 = 0 + 1 = 1

0

Factorial of given Number :-

$n! = n \times n-1 \times n-2 \cdots 1$

$= 1 \times 2 \times 3 \times 4 \times \cdots (n-1) \times (n)$

Recursive Defn

$fact(n) = 1 \times 2 \times 3 \times \cdots (n-1) \times n$

$= fact(n-1) \times n$

Math Relationship

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

Date: / / /

Putting Recurrence relation in ~~was~~ program.

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

② Power Using Recursion :- $(m)^n$ example 2^5

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

$p(m,n) = m \times m \ldots (n-1)$ 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;
  }
  return p(m,n-1)*m
}
```

Trace Tree pow(2,3)

pow(2,3) ← 4×2=8

↓

pow(2,2)*2 ← 2×2=4

↓

pow(2,1)*2

↓

pow(2,0)*2 = 1×2

↑ 1

program

Date: / /

Taylor Series Recursion :-

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

Exploring Approach :-

Let say we want taylor series up to $x^4/\underline{|4}$ i.e total FIVE Terms. So Trace tree should look like below:-

Trace Tree $e(x,4)$:-

Trace tree should like this →

$e(x,4)$

↓

$e(x,3) \rightarrow \frac{x^3}{\underline{|3}} + \left(1 + \frac{x}{\underline{|1}} + \frac{x^2}{\underline{|2}}\right)$

↓

$e(x,2) = \frac{x^2}{\underline{|2}} + (1 + \frac{x}{1})$ from below

|

$e(x,1) \rightarrow \frac{x}{1}$ (It should return val$x$) +1 (from below add)

|

$e(x,0) = 1$ (It shoul return 1)

Continue

Date / /

Take 2 static variable

initial values of static var.
trace the tree again.

```
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);
}
```

Fibonacci Series :

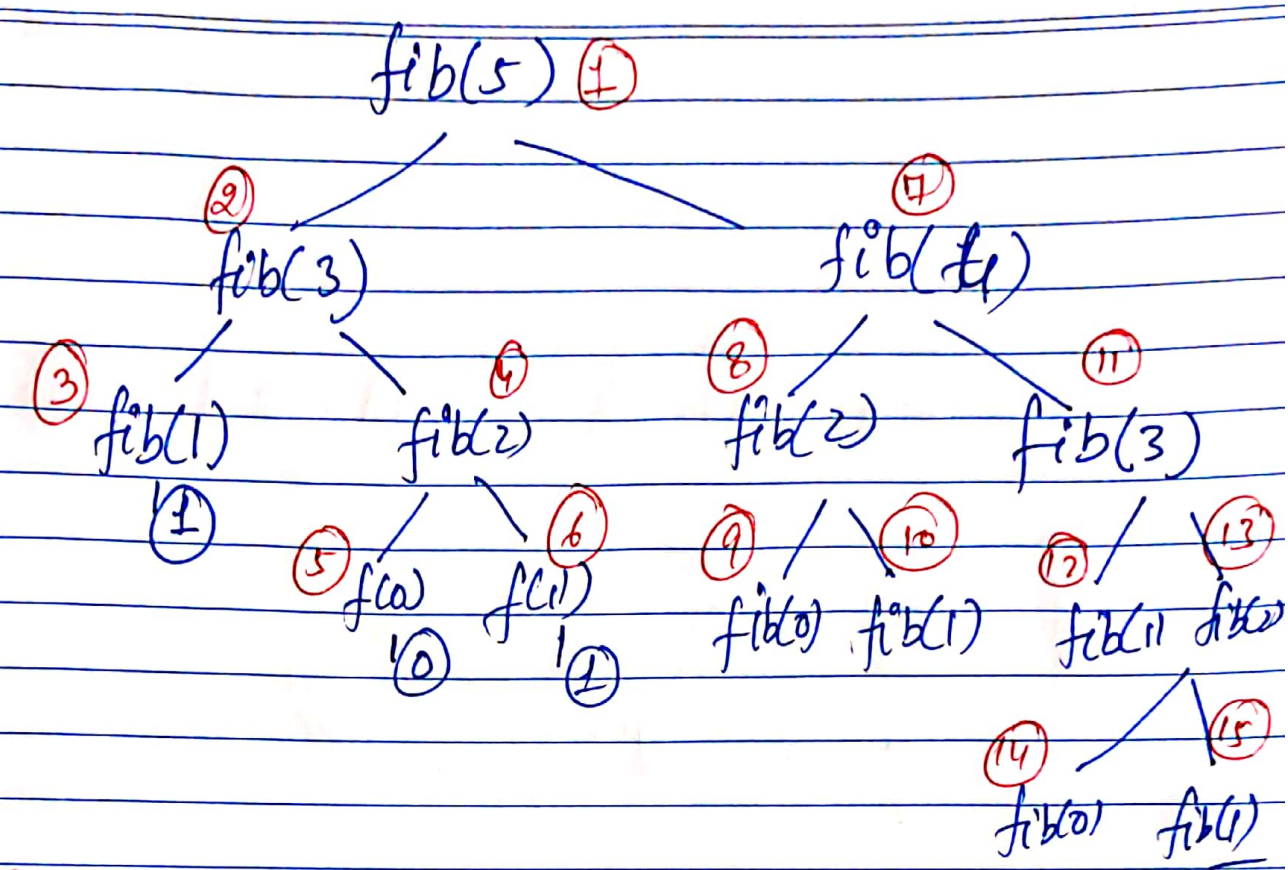
0, 1, 1, 2, 3, 5, 8 ....

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

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

Q Trace Tree
fib(5)

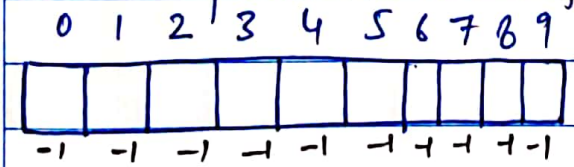
Date: / / /



○ → Denote calling sequence.

✱ Memoization Approach for Fibonacci :-

In above Recursive example we see lots of funtion call repeat ex— fib(2), fib(3), fib(4) :-

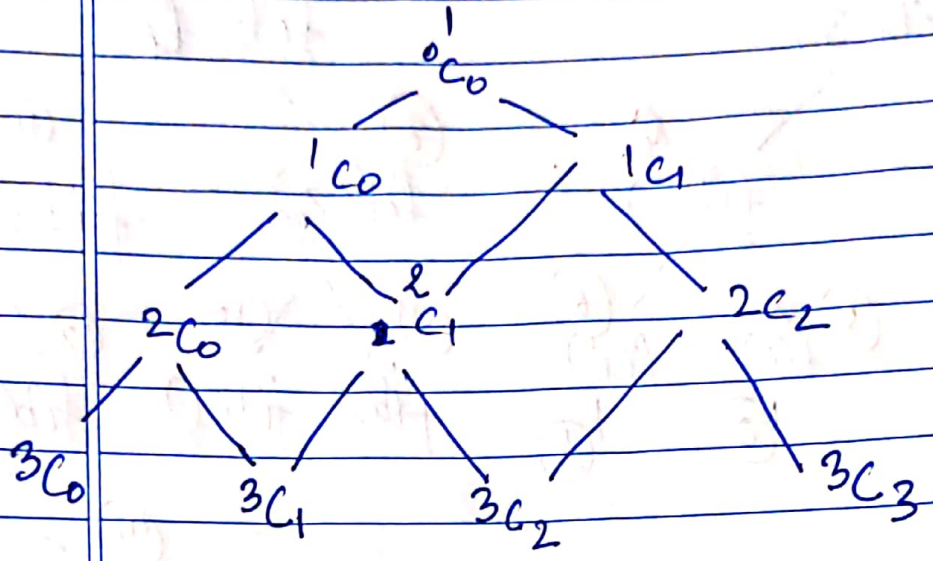



```
int F[10];
int fib(int n) {

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])
}
```

Date: / /

${}^{n}C_{r}$ Using Recursion := (Pascal Triangle)



Any element is sum of above 2 elements :-

ex :- $3C_1 = 2C_0 + 2C_1$

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