Step 1: Learn the Basics

Lec 1: Things to know in C++ | Java | Python or any language

① User Input/output

→ Scanner for input | Sopln for output.

→ Ex:

```
import java.util.Scanner;
Class Demo
{
    psvm(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        Sopln("Enter your name: ");
        String name = sc.nextLine();          // Ramesh
        Sopln("Enter your age: ");
        int age = sc.nextInt();               // 28
        Sopln(name + " " + age);
    }
}
```

output:: Ramesh 28

② Data types

→ Data types are categorized into primitive type and reference type.

→
- (a) Primitive Data type
- (b) Reference Data type.

ⓐ Primitive Data types

→ There are the most basic data types in Java.

→ Ex:

Data type	Size	Example
byte	1 byte	byte b = 100;
short	2 bytes	short s = 10000;
int	4 bytes	int x = 50000;
long	8 bytes	long l = 150006;
float	4 bytes	float f = 5.75f;
double	8 bytes	double d = 19.99
char	2 bytes	char c = 'A';
boolean	1 bit	boolean b = true;

ⓑ Reference Data type

→ These store references to objects.

→ Ex:-

Type	Example
String	String name = "Alice";
Arrays	int[] arr = {1, 2, 3};
custom class	Person p = new Person();
wrapper	Integer i = 5;

③ if else statements

→ Ex:-

```
class Demo
{
    psvm(String[] args)
    {
        int number = -10;

        if (number>=0)
            Sopln(".Positive number");
        else
            Sopln("Negative number");
    }
}
```

output: Negative number

④ Switch

→ The switch statement is used to execute one block of code among many options based on the value of a variable.

→ Ex:-

```
String fruit = "Apple";
switch(fruit)
{
    case "Apple":
        Sopln("You choose Apple");
        break;
    case "Mango":
        Sopln("You choose Mango");
        break;
    default:
        Sopln("unknown fruit");
}
```

output:- You choose Apple

⑤ Arrays, Strings

→ Array: An array is a collection of elements of the same data type stored in a contiguous memory location.

Ex:-

```
int[] num = new int[5];
int[] arr = {90, 80, 70, 85};
```

→ String: A string in java is an Object that represents a sequence of characters.

Ex:-

```
String name = "Alice";
String city = new String("New York");
```

⑥ for loops

→ used when you know how many times you want to loop.

→ Ex. print 1 to 5

```
class Demo
{
    psvm(String[] args)
    {
        for(int i=1; i<=5; i++)
        {
            sopln(i);
        }
    }
}
```

output:

1
2
3
4
5

⑦ while loop

→ Ex. Print 1 to 5

```
class Demo
{
    psvm(String[] args)
    {
        int i=1;
        while (i<=5)
        {
            sop(i);
            i++;
        }
    }
}
```

output: 1 2 3 4 5

⑧ functions

→ A function/method is a block of code that performs a specific task, and can be called multiple times.

→ Ex: Add two numbers

```
class Demo
{
    static int add(int a, int b)
    {
        return a+b;
    }
    psvm(String[] args)
    {
        int result = add(5, 3);
        sop(result);
    }
}
```

output: 8

⑨ Time & Space Complexity

ⓐ Time complexity

→ It tells you how fast an algorithm runs as the input size increases.

→ Common time complexities.

Notation	Name	Example
O(1)	constant time	Accessing an array element: arr[0]
O(log n)	logarithmic time	· Binary search
O(n)	linear time	loop through Array.
O(n log n)	log-linear time	merge sort, Quick sort
$O(n^2)$	Quadratic time	· Nested loop (Bubble sort)
$O(2^n)$	Exponential time	Recursive fibonacci
O(n!)	factorial time	Brute-force permutation

ⓑ Space complexity

→ It tells you how much memory an algorithm need based on input size.

→ includes.

- variables
- Data structure (arrays, Lists, maps etc)
- function call stack (especially in recursion)

Lec-2: Build up logical Thinking

(a) Pattern -Done

Lec 3: Java collections : Done

Lec 4: Basic Maths

(1) Count Digits

→ N = 7789 : extraction of digits

7789 [%10] = 9

↳ It will give the remainder.

7780 ↔ 9

7789/10 = 778.9 = 778

778 %10 = 8 (⇒ 778/10 = 77

77 %10 = 7 (⇒ 77/10 = 7

7%10 = 0

→ while (N>0)

{ lastDigit = N%10;

N = N/10;

count++;

}

Sop(count);

→ TC → O(log10(N))

log10(7789) + 1

3.81... + 1

4

② Reverse a number

→ Given an integer N return the reverse of given number.

Note: If a number has trailing zeros, then the reverse will not include them, for eg Reverse of 10400 will be 401 instead of 00401.

Ex:- I/P : N = 12345

O/P : 54321

→

```
rev=0;
while(N>0)
{ int rem = N%10;
  N = N/10;
  rev = rev*10 + rem;
}
System.out(rev)
```

③ Palindrome number

→ Given an integer N, return true if it is a palindrome else return false.

A palindrome is a number that reads the same backward as forward.

for Example : 121, 1331 & 4554

Ex:-

I/P : 4554
O/P : true

I/P = 7789
O/P = false

→ int n = 1991

int rev = 0;
int num = n;
while (n > 0)
{
int rem = n % 10;
n = n / 10;
rev = rev * 10 + rem;
}
return num == rev;

(4) Armstrong number

→ Given an integer N, return true if it is an armstrong number otherwise return false.

An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the numbers of digits.

Ex:- 153

$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

true

→ int n = 153;
int count = (int) Math.log10(n) + 1;
int ans = 0;
while (n > 0)
{
int rem = n % 10;
n = n / 10;
ans = ans + Mathpow(rem, count);
}
Sopln (num == ans);

⑤ Print all divisors:

→ Given an integer N, return all divisors of N.

A divisor of an integer N, is a positive integer that divides N without leaving a remainder.

Ex:: n = 36

o/p:: [1, 2, 3, 4, 6, 9, 12, 18, 36]

→ Ex::

```
n = 36
for(i=1; i<=n; i++)
{
    if(n%i==0)
        cout(i);
}
```

→ Tc :- O(n)

⑥ find GCD of two numbers

→ Given two integers N1 and N2. find their greatest common divisor.

The GCD of any two integers is the largest number that divides both integers.

Ex:: n1 = 9
n2 = 12

o/p:- 3

→

```
int gcd=1
for(int i=1; i<=Math.min(n1,n2); i++)
{ if(n1%i==0 && n2%i==0)
      gcd=i;
}
```

Tc :- O(min(n1, n2))

→

```
for( i= min(n1,n2); i>=1; i--)
{ if(n1%i==0 && n2%i==0)
  { gcd=i;
    break;
  }
}
```

Tc : O(min(n1, n2))

→ Euclidean Algorithm :-

---

n1, n2

gcd(n1, n2) = gcd(n1 - n2, n2) = = where n1 > n2

n1 = 20
n2 = 15

gcd(20, 15) = gcd(5, 15)
gcd(15, 5) = gcd(10, 5)
gcd(10, 5) = gcd(5, 5)
gcd(5, 5) = gcd(0, 5) → gcd when zero

gcd(a, b) = gcd(a + b, b)

```
while(a>0 && b>0)
{ if(a>b) a=a%b;
  else    b=b%a
}
if(a==0) sop'in(b)
else     sop'in(a)
```

Tc :- O(logϕ (max(a, b)))

(7) Check Prime number

→ Given an integer n, check whether it is prime or not.

A prime number is a number that is only divisible by 1 and itself, and the total number of divisors is 2.

Ex:- n=2 | n=10
o/p:- True | o/p = false

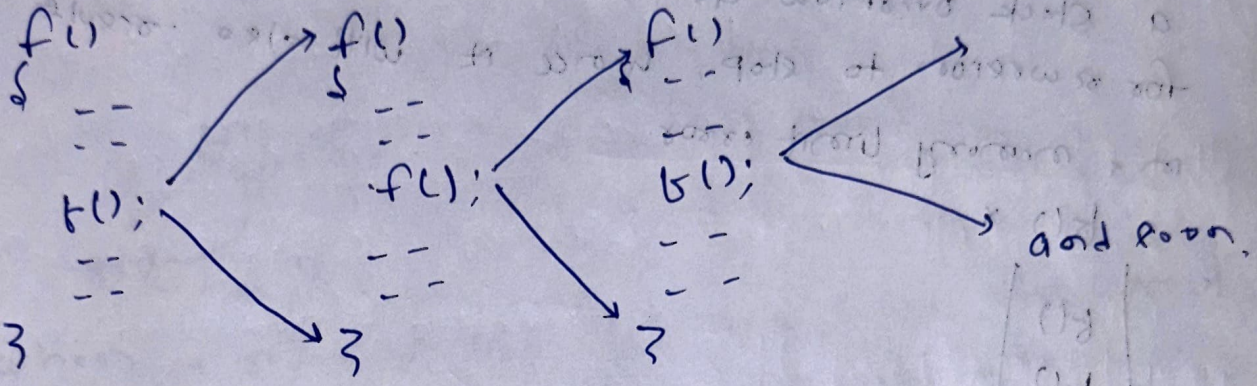
→
```
int cnt = 0;
for(i=1; i*i<=n; i++)
{
    if(n%i==0)
    {
        cnt++;
        if((n/i) != i)
            cnt++;
    }
}
if(cnt == 2) return true;
else return false;
```

Lec 5: Learn Basic Recursion

(1) understanding recursion by printing something N times

→ what is Recursion?

- It is a phenomenon when a function calls itself indefinitely until a specified condition is fulfilled.

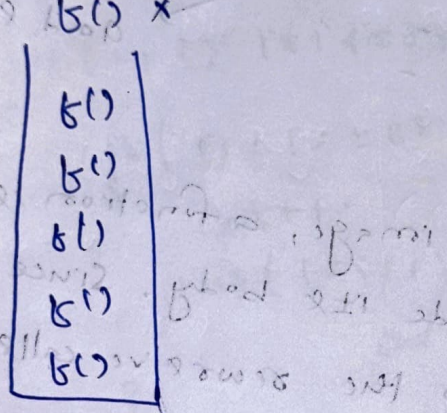


- As we can see in the above image, a function is calling the same function inside it's body. Since there is no condition to stop the recursive calls the calls will run indefinitely until the stack runs out of memory (stack overflow).

→ what is ~~Recursive~~ stack overflow in Recursion?

- Whenever recursion calls are executed, they are simultaneously stored in a recursion stack where they wait for the completion of the recursive function. A recursive function can be only be completed if a base condition is fulfilled and the control returns to the parent function.

- But, when there is no base condition given for a particular recursive function, it gets called indefinetly which results in a stack overflow. i.e, exceeding the memory limit of the recursion stack and hence the program terminates giving a segmentation fault error.
- The illustration above also represents the case of a stack overflow as there is no terminating condition for recursion to stop. hence it will also result in a memory limit error.



→ Base condition

- It is the condition that is written in a recursive function in order for it to get completed and not to run infinitely. After encountering the base condition, the function terminates and returns back to its parent function simultaneously.

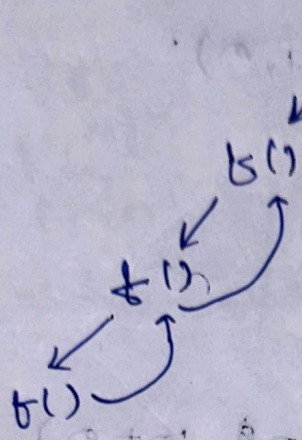
- Ex:

```
int count=0;
f()
{ if(count==3)
      return;
  cout<<count;
  count++;
  f();
}
```

o/p: 0 1 2

→ Recursive Tree

- A recursion tree is basically a representative form of recursion which depicts how functions are called and returned as a series of events happening consecutively. It is a pictorial description of the process of recursion as below



- When a recursive call gets completed, the control returns back to its parent function which is then further executed until the last function waiting in the recursive stack returns.

② Print name n times using Recursion

```
void fun(i, n)
{
  if(i > n)
    return;
  cout("Java");
  fun(i+1, n);
}
main()
{
  int n = 3;
  fun(1, n);
}
```

fun(1,3) → fun(2,3) → fun(3,3) → fun(4,3)

Java Java Java

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

f(1,3) f(2,3) f(3,3) f(4,4) Next the base condition.

TC :- O(n)
SC :- O(n)

③ Print linearly from 1 to N.

→ n = 4, O/P :- 1, 2, 3, 4

```
public static void main(String[] args)
{
    int n = 4;
    fun(1, n);
}
```

```
void fun(int i, int n)
{
    if (i > n)
        return;
    System.out.println(i);
    fun(i+1, n);
}
```

④ Print from N to 1.

→ n = 4, O/P :- 4 3 2 1

```
public static void main(String[] args)
{
    int n = 4;
    fun(n, n);
}
```

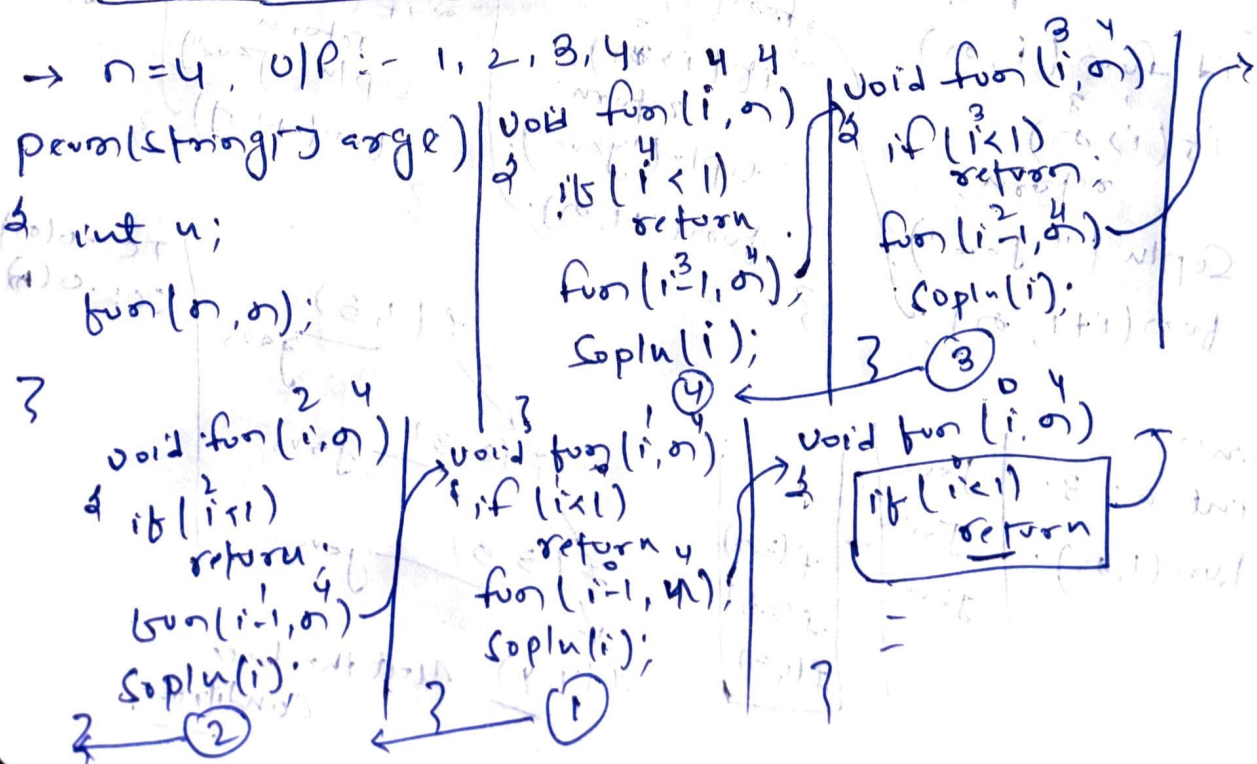
```
void fun(int n, int n)
{
    if (n < 1)
        return;
    System.out.println(n);
    fun(n-1, n);
}
```

⑤ Print linearly from 1 to N (But by Backtracking)

→ n = 4, O/P :- 1, 2, 3, 4

```
public static void main(String[] args)
{
    int n;
    fun(n, n);
}
```

```
void fun(i, n)
{
    if (i < 1)
        return;
    fun(i-1, n);
    System.out.println(i);
}
```



⑥ Print n → 1 (using backtrack)

→ n = 3, · O/P :- 3, 2, 1

```
prvoid(String[] args)
{
    int n;
    fun(1, n);
}
```

```
void fun(i, n)
{
    if(i > n)
        return;
    fun(i+1, n);
    sopln(i)
}
```

```
fun(i, n)
{
    if(i > n)
        return;
    fun(i+1, n);
    sopln(i);
}
```

```
fun(i, n)
{
    if(i > n)
        return;
    fun(i+1, n);
    sopln(i);
}
```

```
fun(i, n)
{
    if(i > n)
        return;
    fun(i+1, n);
}
```

3
2
1

⑦ Sum of first n numbers

ⓐ Parameter
ⓑ functional

ⓐ Parameterized way

N = 3 O/P :- 6

```
prvoid(String[] args)
{
    int n = 3;
    Soplnfun(n);
}
```

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

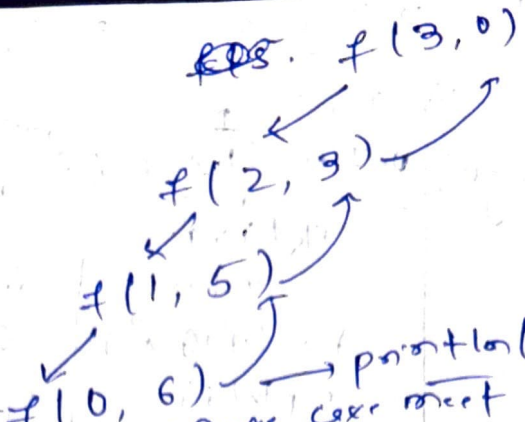
```
prvoid(String[] args)
{
    int n = 3;
    sopln(0);
    fun(n, 0);
}
```

```
fun(n, sum)
{
    if(n < 1)
        sopln(sum);
        return;
    fun(n-1, sum+n);
}
```

```
fun(n, sum)
{
    if(n < 1)
        sopln(sum);
        return;
    fun(n-1, sum+n);
}
```

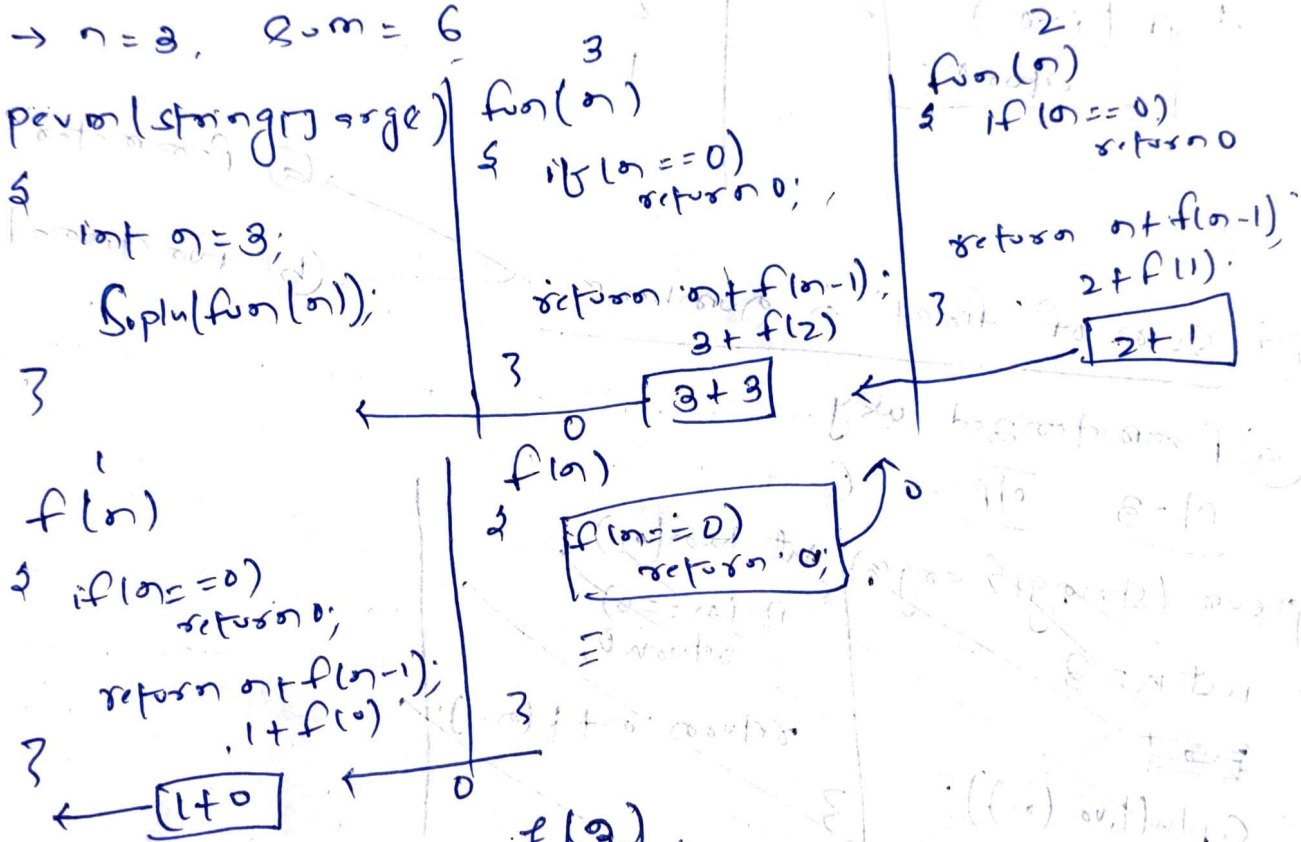
```
fun(n, sum)
{
    if(n < 1)
        sopln(sum)
        return;
    fun(n-1, sum+n);
}
```

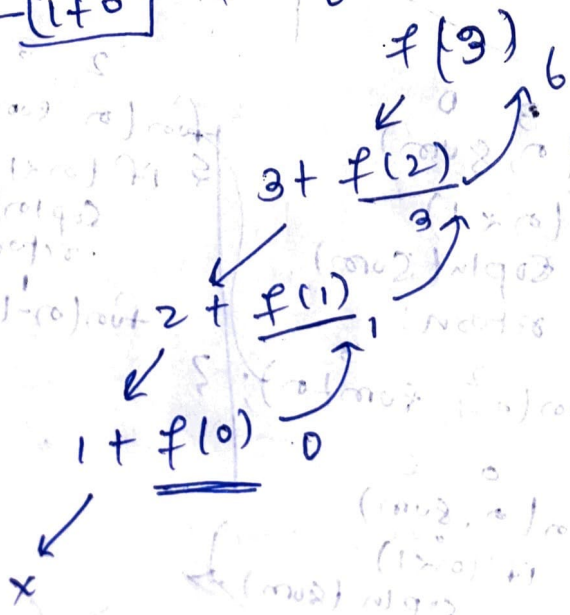
⇒ 6



ⓑ functional

→ n = 3, sum = 6





⑧ factorial of N

→ n = 3, o/p :- 1*2*3 = 6

```
→ psvm(String[] args)
{
  int n = 3;
  Sop(fun(n));
}
```

```
int fun(n)
{ if(n==0)
    return 1;
  return n*f(n-1);
}
```

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

3 → fun(3) → 3*f(2) → 3*2 → 6

2 → f(2) → 2*f(1) → 2*1

1 → f(1)

```
f(1)
{ if(n==1)
    return 1;
}
```

⑨ Reverse a given array

ⓐ Using two pointers:

- n = 5, arr[] = {5, 4, 3, 2, 1} (indices 0, 1, 2, 3, 4)

o/p :- {1, 2, 3, 4, 5}.

```
→ psvm(String[] args)
{ int n = 5;
  int[] arr = {1,2,3,4,5};
  Sop(arr);
  fun(0, n-1, arr);
  Sop(arr);
}
```

```
f(l, r, arr)
{ if(l>=r)
    return;
  swap(l, r, arr);
  fun(l+1, r-1, arr);
}
```

f(0, 4, arr) → f(1, 3, arr) → f(2, 2, arr)

```
f(l, r, arr)
{ if(l>=r)
    return;
  swap(l, r, arr);
  fun(l+1, r-1, arr);
}
```

(b) Using single pointer

I/P: n=5 ; arr = {5, 4, 3, 2, 1}

```
prevn(S[]arg1)
{
int n=5;
arr={5,4,3,2,1};
fun(n-1, arr);
Sop1n(arr);
}
```

```
fun(i, arr)
{
 if(i>=arr.length-i-1)
    return;
 Swap(i, arr.length-i-1)
 fun(i+1, arr);
}
```

(10) Check if a string is palindrome:

→ A string on reversal reads the same.

→ I/P :- s = "MADAM"
0 1 2 3 4

```
prevn()
{
String s="MADAM";
Sop1n(fun(0, s));
```

```
fun(int i, String s)
{
 if(i>=n/2)
    return true;
 if(s[i] != s[n-i-1])
    return false;
 return f(i+1, s);
}
```

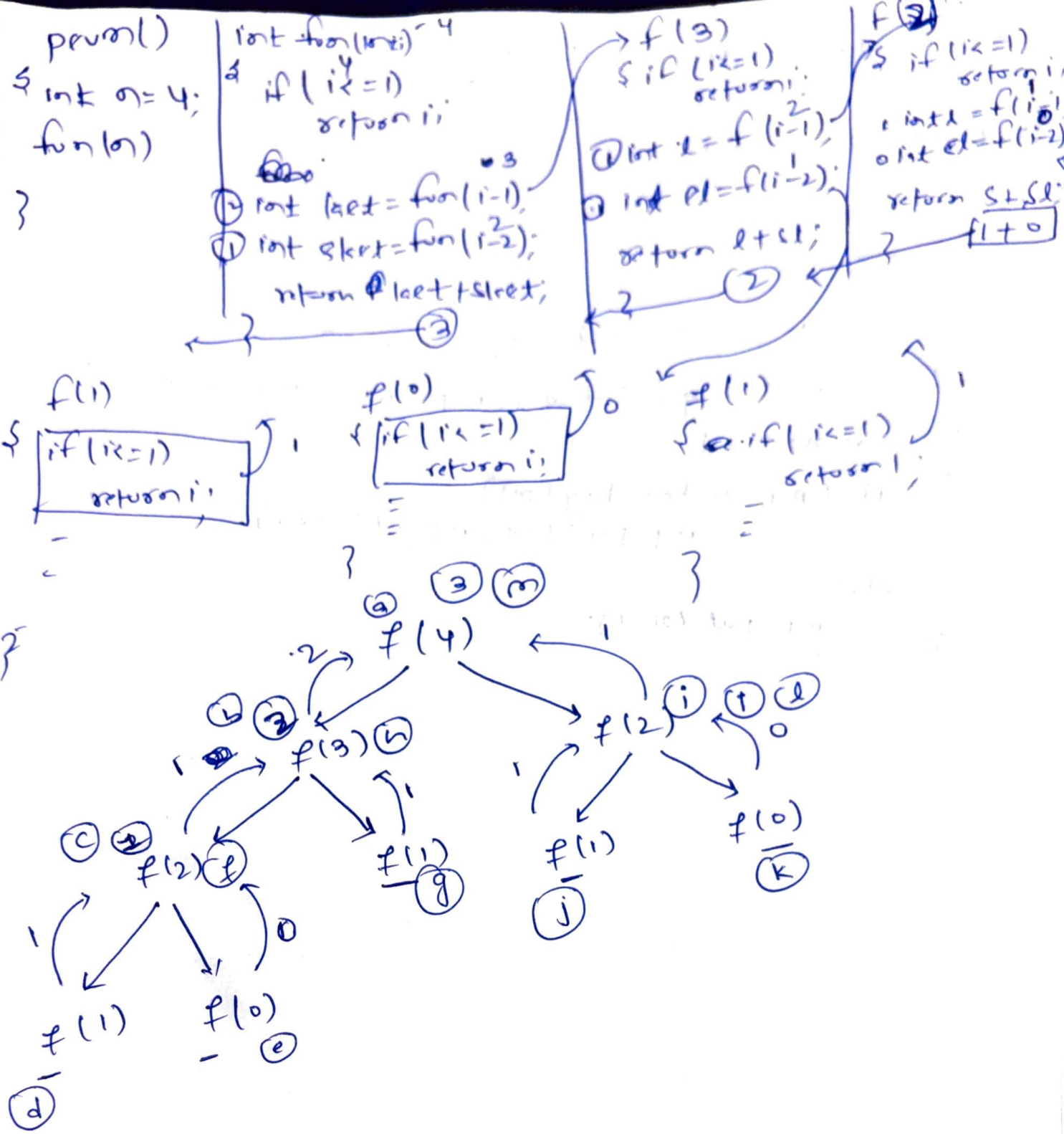
(11) Fibonacci number

→ Given An integer N. Print the fibonacci series upto the Nth term.

N=5

O/P: 0, 1, 1, 2, 3, 5

fib(i) = fib(i-1) + fib(i-2);


pevon()
int n=4;
fun(n)
int fun(int i)
if(i==1)
return i;
int left=fun(i-1);
int sleft=fun(i-2);
return left+sleft;
f(3)
if(i==1)
return i;
int l=f(i-1);
int sl=f(i-2);
return l+sl;
f(2)
if(i==1)
return i;
int l=f(i-1);
int sl=f(i-2);
return s+sl;
f+0
f(1)
if(i==1)
return i;
f(0)
if(i==1)
return i;
f(1)
if(i==1)
return 1;
f(4)
f(3)
f(2)
f(1)
f(2)
f(1)
f(0)
f(1)
f(0)

Lec6: Learn Basic Hashing

① Hashing Theory:-

→ ~~arr[] =~~

→ arr = {1, 5, 2, 9, 7, 2, 3, 1}

```
HashMap<Integer, Integer> mp = new HashMap<>();
for(int x: arr)
{   if (mp.containsKey(x))
        mp.put(x, mp.get(x)+1);
    else
        mp.put(x, 1);
}
```