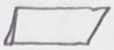
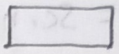
JAVA

Flowchart:- Diagram to represent solutions of problems.
↓
Small parts → logically arrange.

Components:-

Start/Exit 

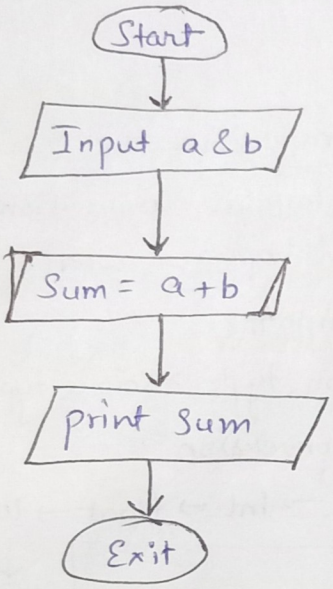
Input/Output 

Process 

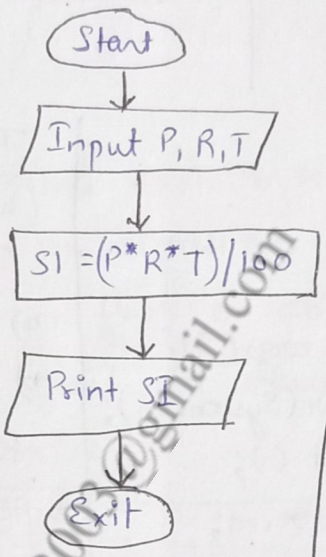
Decision 

Arrows ↓

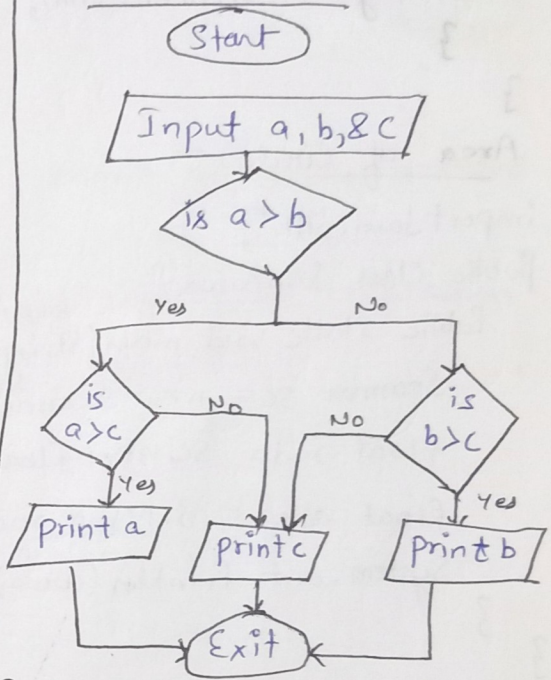
Calculate Sum:-



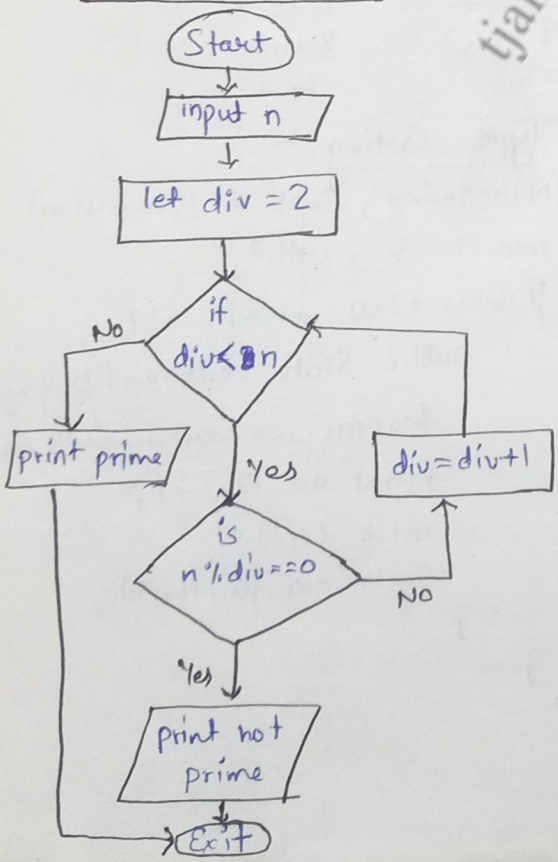
Calculate SI:-



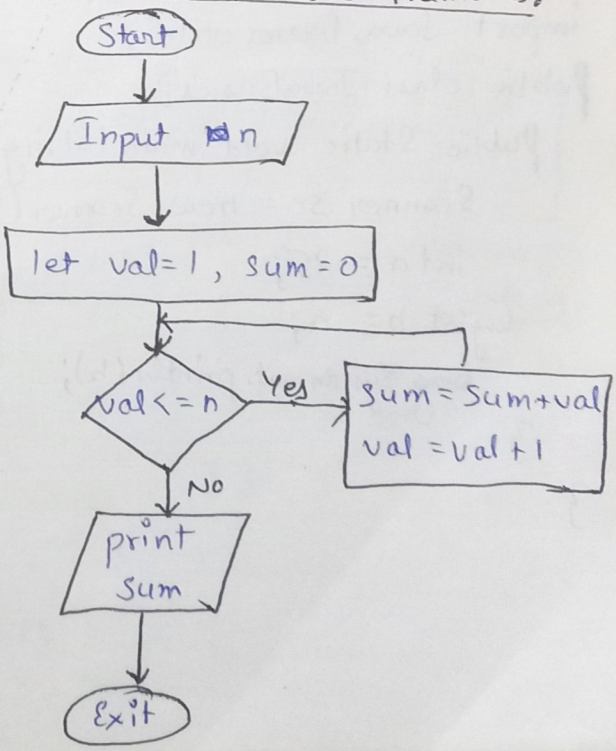
Max of 3 No.:-



No. is Prime or Not:-



Sum of first N Natural numbers:-



Sum of a & b:-

```
import java.util.*;
Public class JavaBasics{
    Public Static Void main(String args[]){
        Scanner sc= new Scanner (System.in);
        int a = sc.nextInt();
        int b= sc.nextInt();
        int sum= a+b;
        System.out.println(sum);
    }
}
```

Product of a & b:-

```
import java.util.*;
Public class JavaBasics{
    Public Static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        int product = a* b;
        System.out.println(product);
    }
}
```

Area of Circle:-

```
import java.util.*;
Public Class JavaBasics{
    Public Static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        float rad = sc.nextfloat();
        float area = 3.14f * rad * rad;
        System.out.Println(area);
    }
}
```

```
import java.~~Basics~~ util.*;
Public class JavaBasics{
    Public Static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int a = 25;
        long b= a;
        ~~long~~ System.out.println(b);
    }
}
```

Type Conversion:-

(Winding, Implicit Conversion)

Conversion happens when

a) Type Compatible

b) Destination type > Source type

Possible Conversion

byte → short → int → float → long → double

25 → 25

4 byte int → 8 byte long

Type Casting:-

(Narrowing, Explicit Conversion)

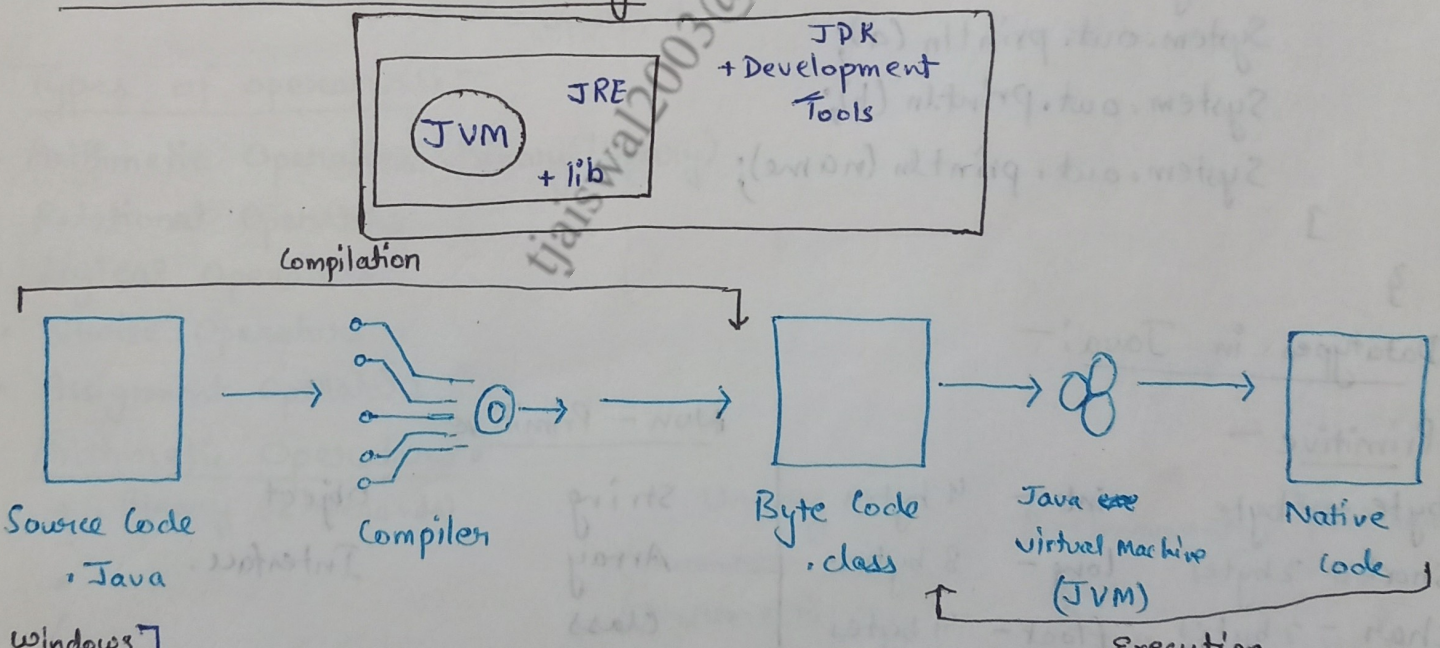
```
import Java.util.*;
Public Class JavaBasics{
    public Static void main(String args[]){
        Scanner sc= new Scanner(System.
        float a= 12.25f;
        int b= (int)a;
        System.out.println(b);
    }
}
```

```
Public class JavaBasics{
    Public Static void main (String args[]){
        char ch = 'a';
        char ch = 'b';
        int number = ch;
        int number = ch2;
        System.out.println(number);
        System.out.println(number2);
    }
}
```

Type promotion in Expression :—

1. Java automatically promotes each byte, short or char operand to int when evaluating an expression.
2. If one operand is long, float or double the whole expression is promoted to long float or double respectively.

How is our code Running :—



Windows, Mac, Linux → Portable language

Byte code se kisi bhi Machine me run ho Jayega.

Boilerplate Code :—

```
Public class JavaBasics{
    Public static void main (String args[]){
        . . . . . .
    }
}
```

Output in Java :-

```
System.out.print("Hello World");
```

System.out.print → Function; "Hello World" → output

```
Public class JavaBasics {
    Public Static void main (Staring args[]){
        System.out.println ("Hello World");
                          └→ line ka space dega.
    }
}
```

Variables in Java :-

2 * (a+b) ⟶ Variable

↓

Literal

JavaBasics
Main, print, println } Identifiers

```
Public Class JavaBasics {
    Public Static void main(String args[]){
        int a = 10;
        int b = 5;
        String name = "Tushar Jaiswal";
        System.out.println (a);
        System.out.println (b);
        System.out.println (name);
    }
}
```


Datatypes in Java :-

Primitive –

byte – 1 byte	int – 4 bytes
Short – 2 bytes	long – 8 bytes
char – 2 bytes	float – 4 bytes
boolean – 1 bytes	double – 8 bytes

Non – Primitive –

String	object
Array	Interface.
class	

Sum of a & b :-

```
Public Class JavaBasics
    Public class void main (String args[]){
        int a = 10;
        int b = 5;
        int sum = a+b;
        System.out.println(sum);
    }
}
```

Comments :-

```
// Hi  → Single line comment
/*
  Hi  → Multi line Comment
*/
```

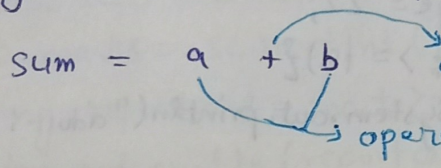
Inputs in Java:-

```
import Java.util.*;
Public class JavaBasics{
    Public static void main (String args[]){
      Scanner Sc= new Scanner (System.in);
      String input = Sc.next ();
      System.out.println (input);
    }
}
```

- next - Space ke pahle Tak capture karta hai.
- nextline - Pura capture karta hai.
- next Int - Number Input ke liye leta hai
- next byte -
- next float - float no. store karne ke liye.
- next double
- next boolean
- next short
- next long

Operators in Java:-

System Symbol that tell compiler to perform some operation.



Types of operators:-

- Arithmetic Operators (Binary / Unary)
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Arithmetic Operators :-

\+ Binary (2 operands)
\-
*
/
%

Unary
(Increment)++
(Decrement)--

Pre-Increment
++a
↓
change than use
--a

Post Increment
a++
↓
use than change
a--

```
import Java.util.*;
Public class JavaBasics{
  Public static void main (String args[]){
      int A= 10;
      int B = 5;
      System.out.println ("Add = " +(A+B));
  }
}
```

Relational Operators :-

==	check two value equal or not
!=	Not equal to
>	Greater
<	Smaller
>=	Greater than equal to
<=	lesser than equal to

Logical Operators :-

&&	(logical AND)	(If Both statements True then returns True)
||	(logical OR)	(If both stat: false then false otherwise True)
!	(logical Not)	True → False False → True

Assignment Operator :-

=		
+=	A+B =	A += B
-=	A-B =	A -= B
*=	A* B =	A *= B
/=	A/= B	A / B

Conditional Statement ; -

If, else
else if
ternary Operator
Switch

If - else statement -

```
if (condition) {
    - - -
    - - - .
}
else {
    - - -
    - - -
}
```

```
import Java.util.*;
Public Class JavaBasics{
    Public Static void main (String args[]){
        int age = 22;
        if (age >= 18){
            System.out.println("adult: drive, vote");
        }
        else{
            System.out.println("not adult");
        }
    }
}
```


Else if Statements :-

```
if (condition 1){

}
else if (condition 2){

}
else{

}
```

Income Tax Calculator :-

```
import Java.util.*;
Public class JavaBasics {
    public Static void main (String args[]){
        Scanner sc= new Scanner ( System.in);
        int income= sc.nextInt();
        int tax;
        if (income < 50,000){
            tax=0;
        }
        else if (income >= 50000 && income <= 1000000){
            tax = int (income * 0.2);
        }
        else{
            tax = (int)(income * 0.3);
        }
```

```
        System.out.println("Your tax is: " + tax);
    }
}
```

Ternary Operator :-

```
Variable = Condition ? Statement1 : Statement2;
                       (True)      (False)
                       (otherwise)
```

```
int larger = (5>3) ? 5 : 3;
String type = (5%2==0) ? "even" : "odd";
```

Check if a student will pass or fail -

marks >= 33 PASS
marks < 33 fail

```
import java.util.*;
public class JavaBasics{
    public static void main (String args[]){
    Scanner sc = new Scanner (System.in);
     int marks = sc.nextInt();
     String reportcard = marks>=33 ? "PASS" : "FAIL";
     System.out.println (reportcard);
    }
}
```

```
import java.util.*;
public class JavaBasics{
    public static void main
        (String args[]) {
      int number = 4;
     String type = (number%2
     ==0 ? "even" : "odd";
     System.out.println (type);
    }
}
```

Switch Statement :-

```
import java.util.*;
public class JavaBasics {
  public static void main(String args[]){
    int number = 2;
    Switch(number){
      case 1: System.out.println("Samosa");
      Case 2: System.out.println("Burger");
      default: System.out.println("Dreaming");
    }
  }
}
```

Calculator :-

```
import java.util.*;
public class JavaBasics {
  public static void main (String args[]){
    Scanner sc = new Scanner (System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    char operator = sc.next().charAt(0);
    Switch(operator){
      Case '+': System.out.println (a+b);
          break;
      Case '-': System.out.println(a-b);
          break;
      Case '*': System.out.println(a*b);
          break;
      Case '/': System.out.println(a/b);
          break;
```

```
case '%': System.out.println(a % b);
          break;
default: System.out.println("Wrong operator");
}
}
}
```

loops:-

Types of loops — • While • For • do While

While loop:-

```
while (condition){
    //do something
}
```

```
import java.util.*;
public class JavaBasics{
    public static void main(String args[]){
        int counter = 0;
        while (counter < 100){
            System.out.println("Hello World");
            counter++;
        }
    }
}
```

Print number from 1 to n:-

```
import java.util.*;
public class JavaBasics{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int range = sc.nextInt();
        int counter = 1;
        while (counter <= range){
            System.out.println(counter+" ");
            counter++;
        }
        System.out.println();
    }
}
```


For loop:-

```
for (initialisation; condition; updation){
    //do something
}
```

```
import java.util.*;
public class JavaBasics{
    public static void main(String args[]){
        int i;
        for(i=1; i<100; i++)
            System.out.println("Hello World");
        }
    }
}
```

Print reverse of a number :-

```
import java.util.*;
public class JavaBasics {
    public static void main(String args[]){
        int n = 10899;
        while (n > 0) {
            int lastdigit = n % 10;
            System.out.print(lastDigit + " ");
            n = n/10;
        }
        System.out.println();
    }
}
```

Do-while loop -

```
do {
  //do something
} while(condition);
```

```
import java.util.*;
public class JavaBasics {
  public static void main(String args[]){
      int counter = 1;
      do {
        System.out.println("Hello");
        counter++;
      }
      while (counter <= 10);
  }
}
```

Keep entering no. till user enter a multiple of 10 :- Break Statement -

```
import java.util.*;
public class JavaBasics {
    public static void main(String args[]
    Scanner sc = new Scanner(System.in);
    do{
      System.out.print("Enter your no.");
      int n = sc.nextInt();
      if(n%10==0) {
          break;
      }
      System.out.println(n);
    } while (true);
  }
}
```

Display all numbers entered by user except multiples of 10. (continue st.)

```
import java.util.*;
public class JavaBasics {
    public static void main(String args[]){
      Scanner sc = new Scanner(System.in);
      do {
        int n = sc.nextInt();
        if (n%10==0) {
          continue;
        }
        System.out.println("no. was :" +n);
      } while(true);
  }
```

Check if a no. is prime or not :-

```
import java.util.*;
public class JavaBasics {
    public static void main(String args[]
      Scanner sc = new Scanner(System.in)
      int n = sc.nextInt();
      if (n!=2){
          System.out.println("n is prime");
      } else {
        boolean isprime = true;
        for (int i=2; i<=n-1; i++){
          if (n%i==0){
```

```
            isprime = false;
        }
    }
    if(isprime == true){
        System.out.println("n is prime");
    }
  }
}
```

Print STAR Pattern :-

```
*
* *
* * * *
* * * *
* * * * *
```

```
import Java.util.*;
public class JavaBasics{
  public static void main(String args[]){
      for(int i=1; i<=5; i++){
          for(int j=1; j<=i; j++){
              System.out.print("* ");
          }
          System.out.println();
      }
  }
}
```

Inverted STAR -

```
* * * *
* * *
* *
*
```

```
import Java.util.*;
public class JavaBasics{
  public static void main(String args[]){
      for(int i=1; i<=4; i++){
          for(int j=1; j<=4-i+1; j++){
              System.out.print("* ");
          }
      }
  }
}
```

Print half Pyramid pattern -

```
1
1 2
1 2 3
1 2 3 4
```

```
import Java.util.*;
public class JavaBasics{
  public static void main(String args[]){
      int n=4;
      for(int i=1; i<=n; i++){
          for(int j=1; j<=i; j++){
              System.out.print("j");
          }
          System.out.println();
      }
  }
}
```

Print character Pattern -

```
A
B C
D E F
G H I J
```

```
import Java.util.*;
public class JavaBasics{
  public static void main(String args[]){
      int n=4;
      char ch='A';
      for(int line=1; line<=n; line++){
          for(int=chars-1; char<=line; cha
              System.out.print(ch);
              ch++
          }
          System.out.println();
      }
  }
}
```

Introduction to Function -

- It is a block of code that perform particular task.
- It is reusable.

Syntax -

```
returnType(){
    //body
    return statement;
}
```

```
public class JavaBasics {
    Public static void printHello(){
        System.out.print("Hello");
    }
    public static void main(String args[]){
        printHello();
    }
}
```

Syntax with parameters:-

```
returnType name(type param1, type param2){
    //body
    return statement
}
```

E.g.

```
public static void calculateSum(){
    Scanner sc = new Scanner(System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    int sum = a+b;
    System.out.println("Sum is:" +sum);
}
public static void main(String args[]){
    calculatesum();
}
}
```

```
→ import java.util.*;
public class JavaBasics{
    public static void calculatesum(int a, int b){   (parameters)
        int sum = a+b;
        System.out.println("Sum is:" + sum);
    }
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b =
        Calculatesum(a, b);"   (argument)
    }
}
```

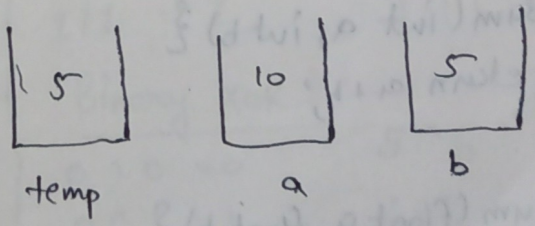
Formal parameters

- parameter
- definition

Actual parameters

- arguments
- Call

Call by value -



Java always call by value.

Find product of a & b –

```
Public static int multiply (int a, int b){
    int product = a * b;
    return product;
```

Bionomial Coefficient –

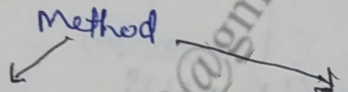
$${}^{n}C_{r} = \frac{n!}{r!\,(n-r)!}$$

```
public static int bicoeff (int n, int r){
    int fact_n = factorial(n);
    int fact_r = factorial(r);
    int fact_nmr = factorial(n-r);
    int bincoeff = fact_n / fact_r * fact_nmr);
    return bincoeff;
}
```

factorial of a number n –

```
public static int factorial (int n){
    int f = 1;
    for (int i=1; i <= n; i++){
        f = f * i;
    }
    return f;
}
```

Function (Inbuilt & User defined functions) –



User defined
{factorial, sum, product}

Inbuilt Method
{Math, pow, sqrt, max}

Function Overlooping: –

Multiple Functions with some name but different parameters.

Multiply (int a, int b)
Multiply (float a, float b)
Multiply (int a, int b, int c)

Using parameter – Did not depend on return type).

f1: Sum of 2 no.

```
int sum (int a, int b){
    return a+b;
```

f2: Sum of 3 no.

```
int sum(int a, int b, int c){
    return a+b+c;
}
```

Using Data Types: –

```
f1 = int sum(int a, int b){
        return a+b;
    }
```

```
f2 = float sum (float a, float b){
        return a+b;
    }
```

Check prime or not –

Optimised way –

```
Public Java.util.*;
import class JavaBasics{
 public static boolean isPrime(int n){
   if (n==2){
     return true;
   }
   for(int i= 2; i<Math.sqrt(n); i++){
     if(n%i==0){
       return false;
     }
   }
 return true;
}
```

Print all primes in a Range :–

```
public static void primesInRange(int n){
    for (int i =2; i<=n; i++){
       if (isPrime (i)){
          System.out.print(i + " ");
       }
    }
    System.out.println();
}
```

Convert from Binary to Decimal –

Binary ⟶ Decimal

n = 101 ⇒ Binary

n = 1 0 1

$2^2 + 2^1 + 2^0$

$2^2 \times (1) + 2^1 \times (0) + 1 \times (1) = 5$

n = 1 0 0 0

2^3 2^2 2^1 2^0

$2^3(1) + 0 \times 2^2 + 2 \times (0) + 1 \times (0)$

$= 8 + 0 + 0 + 0 = 8$

$(1000)_2 = (8)_{10}$

Decimal to Binary :–

n = 4

2	4	0
2	2	0
	1	1

$(100)_2 = (4)_{10}$

Bit-wise Operators :–

Binary AND &

Binary OR |

Binary XOR ^

Binary One's Complement ~

Binary Left Shift <<

Binary Right Shift >>

Binary AND &

0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1

5 & 6

101

& 110

100 $= (4)_{10}$

Binary OR |

0 | 0 = 0

0 | 1 = 1

1 | 0 = 1

1 | 1 = 1

5 | 6

101

110

111 $= (7)_{10}$

Binary XOR ^

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

5 ^ 6

101

110

011 $= (3)_{10}$

Binary One's Complement ~

~0 = 1 ~5 → 101

~1 = 0 010 = $(2)_{10}$

5 → 00000101

MSB LSB

One's Compl → 11111010 = $(-6)_{10}$

~~Two's compl~~ →

-6 = 11111010

One's Compl → 00000101

+1

Two's Compl → 00000110

-6

~0 → -1

Binary Left Shift :- <<

5<<2 5 → 000101

010100 → $(10100)_2$

$a << b = a * 2^b$

Binary Right Shift >> :-

~~6~~

6>>1 000110

0 000 110 = $(3)_{10}$

$a >> b = a / 2^b$

Check if a no. is Odd or Even :-

0 = 000 odd → LSB → 1

1 = 001

2 = 010 Even → LSB → 0

3 = 011

4 = 100 LSB = n & 1

```
~~public static~~ ~~void~~ class
public class BitManipulation{
  public static void oddEven(int n){
    int bitmask = 1;
    if(n & bitmask)==0){
      Syso("even");
    }else{
      Syso("odd");
    }
```

Operations :-

1) Get ith bit

00001111 i = 2

& 00000100 → 1<<2 (1<<i)

00000100 ith bit = 1

```
public static int getIthBit(int n, int i){
  int bitMask = 1<<i;
  if(n & bitMask)==0){
    return 0;
  }else{
    return 1;
  }
}
```

2) Set ith bit

00010 1 0 10 i = 2

↓

000.0 1 1 10

```
public static int setIthBit
    (int n, int i){
  int bitMask = 1<<i;
  return n | bitMask;
}
```

3) Clear ith bit

10 = 10 1 0 i = 1

↓

1000

1010

& 1101 ~(1<<i)

1000

```
public static int clearIthBit
  (int n, int i){
    int bitMask = ~(1<<i);
    return n & bitMask;
```

Update ith bit :-

```
public static int updateIthBit(int n, int i, int newBit){
  if(newBit==0){
    return clearIthBit(n,i);
  }else{
    return setIthBit(n,i);
  }
}
```

Clear Last i bits :—

n = 1111 , i = 2
↳ 1100

```
1111
& 1100   (~0) or (-1)
1100
```

```
public static int clearIBit(int n, int i){
    int bitMask = (~0)<<i;
    return n & bitMask;
}
```

Clear Range of bits :—

n = 1001 1010 11 , i = 2 , j = 7
↳ 1001 0000 11

a = 111 1100000000 (~0) << (j+1)
b = 0000000000 11 (1<<i) - 1

01 = $(1)_{10} = 2^1 - 1$
011 = $(3)_{10} = 2^2 - 1$
0111 = $(7)_{10} = 2^3 - 1$
01111 = $(15)_{10} = 2^4 - 1$
011111 = $(31)_{10} = 2^5 - 1$

$2^r - 1$, r = i-1

2^b → 1<<b = 1 x (2^b)

```
public static int clearBitsInRange(int n, int i, int j){
{
    int a = ((~0)<<(j+1));
    int b = (1<<i) - 1;
    int bitMask = a|b;
    return n & bitMask;
}
```

Check if a number is a power of 2 or not :—

4 → 2^2
8 → 2^3
7 → 2^n X

4 → 100
3 → 011
& → 0

8 → 1000
7 → 0111
& → 0

16 → 10000
15 → 01111
& → 0

n & (n-1) = 0

```
public static boolean isPowerofTwo(int n){
    return (n&(n-1))==0;
}
```

Count Set Bits in a Number :—

10 → 1010
no. of set bits = 2

(i) n = 1010 , n>>1 , count = 0
(ii) n = 0101 , n>>1 , count = 1
(iii) 0010 , n>>1 , count = 1
(iv) 0000 , count = 2

```
public static int countSetBits(int n){
    int count = 0;
    while (n>0){
        if(n&1)!=0){ //check the LSB
            count++;
        }
        n = n>>1;
    }
    return count;
}
```

Fast Exponentiation :-

a^N → 5 → 101

3^{101} → $3^4 \cdot 3^2 \cdot 3$ → 101

$a^{101} = (1 \times a^4) * (1) * (1 \times a) = a^4 \times a^1 = a^5$

① a = 3; ans = 1

ans = 3

$a = a^2 = 9$

② a = 9; ans = 3

ans = 3 × 1

$a = a^2 = 81$

③ a = 81; ans = 3

ans = 3 × 81 = 243

$a = a^2 = (81)^2$

$5^3 = 5^{011}$ → 5^4 5^2 5^1

ans = 1, a = 5, n = 011

① ans = 1 × 5 = 5

a = 25

n = 001

② ans = 5 × 25 = 125

$a = (25)^2 = 625$

n = 000

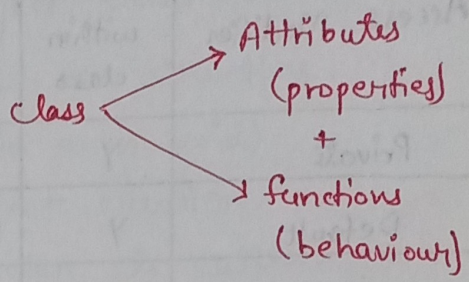
```
public static int fastExpo(int a, int n){
    int ans = 1;
    while(n>0){
        if((n&1)!=0){
            ans = ans * a; //check LSB
        }
        a = a*a;
        n = n>>1;
    }
    return ans;
}
```

Object Oriented Programming (OOPs)

Classes & objects :-

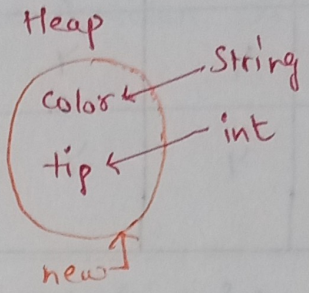
Objects – entities in the real world.

class – group of these entities. (Blueprint)



Pen –

color: String
tip: 5 (int)
setColor():
setTip():




```
public class OOPS{
    public static void main(String args[]){
        Pen p1 = new Pen(); //created a pen object called P1.
        p1.setColor("Blue");
        System.out.println(p1.color);
        p1.setTip(5);
        System.out.println(p1.tip);
    }
}
class Pen{
    String color;
    int tip;
    void setColor(String newColor){
        color = newColor;
    }
    void setTip(int newTip){
        tip = newTip;
    }
}
class Student{
    String name;
    int age;
    float percentage;
    void calcPercentage(int phy, int che, int math){
        percentage = (phy + che + math)/3;
    }
}
```

Access Modifiers :-

Access Modifier	within class	within package	outside package by Subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

```
class BankAccount{
    public String username; //can be change
    private String password; //cannot be change on main.
    public void setPassword(String pwd){
        password = pwd;
    }
}
```

Getters & Setters :-

Get : to return the value (Getters)
Set : to modify the value (Setters)

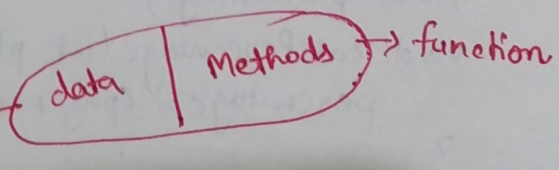
this : this keyword is used to refer the current object.

```
class Pen{
    private String color;
    private int tip;
    String getColor(){ //Getter
        return this.color;
    }
    int getTip(){ // Getter
        return this.tip;
    }
```

```
    void setColor(String newColor){ //Sette
        this.color = newColor;
    }
    void setTip(int tip){ //Setter
        this.tip = tip;
    }
}
```

- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

★ **Encapsulation** :- Encapsulation is defined as the wrapping up of data & methods under a single unit. It also implements data hiding. (private / protected)



Constructors: - Constructor is a special method which is invoked automatically at the time of object creation.

→ Constructors have the same name as class or structure.
→ Constructors don't have a return type. (Not even void)
→ Constructors are only called once, at object creation.
→ Memory allocation happens when constructor is called.

```
public class OOPS{
    public static void main(String args[]){
        Student s1 = new Student();
    }
}
class Student{
    String name;
    int roll;
    Student(){
      System.out.println("constructor is called...");
    }
}
```

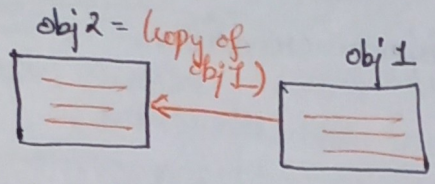
Output: -
constructor is called...

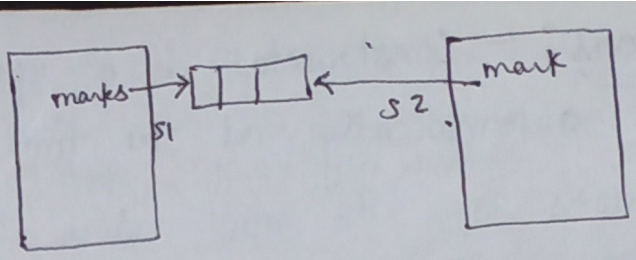
Types of Constructors: -

- Non-parameterized
- Parameterized
- Copy Constructor

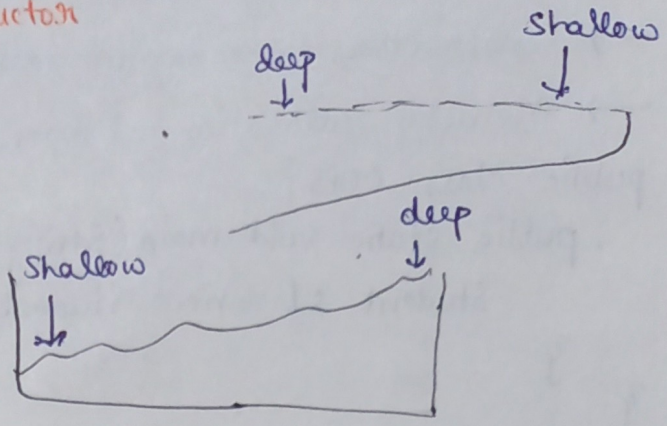
```
public class OOPS{
   public static void main(String args[]){
       Student s1 = new Student(); // Constructor Overloading (Polymorphism)
       Student s2 = new Student("Tushar");
       Student s3 = new Student(123);
   }
}
class Student{
    String name;
    int roll;
    //Student(){
    //    System.out.println("constructor is called...");
    //}
    Student(String name){
        this.name = name;
    }
    Student(int roll){
        this.roll = roll;
    }
}
```

Copy Constructors :-






```
Student(Student s1){ //shallow copy constructor
    marks = new int[3];
    this.name = s1.name;
    this.roll = s1.roll;
    this.marks = s1.marks;
}
```

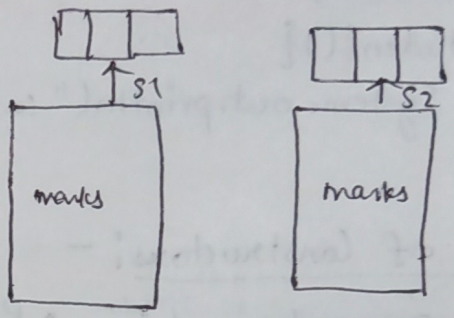


Shallow & Deep Copy:-

Shallow - only reference copy
changes reflect

Deep - new marks array,
changes dont reflect

```
Student(Student s1){ //deep copy constructor
    marks= new int[3];
    this.name = s1.name;
    this.roll = s1.roll;
    for(int i=0; i<marks.length; i++){
        this.marks[i] = s1.marks[i];
    }
}
```

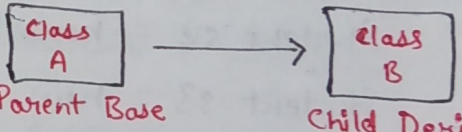


Destructors :- Java automatically delete unused object (Garbage Collector)

Balance → Constructor
→ Destructor

Inheritence :-

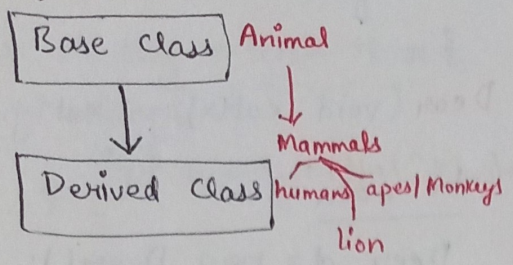
Inheritence is when properties & methods of base class are passed on to a derived class.



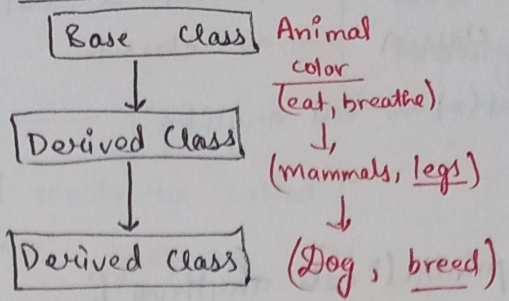

```
public class Inheritence{
  Public static void main(String args[]){
      Fish shark = new Fish;
      shark.eat();
  }
}
//Base Class
class Animal{
    String color;
    void eat(){
        System.out.println("eats");
    }
    void breathe(){
        System.out.println("breathes");
    }
}
//Derived Class/ Subclass
class Fish extends Animal{
    int fins;
    void swim(){
        System.out.println("swims in
                water");
    }
}
```

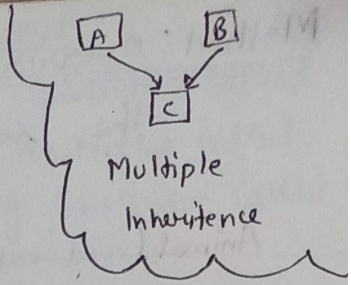
Types of Inheritence :-

Single Level Inheritence :-

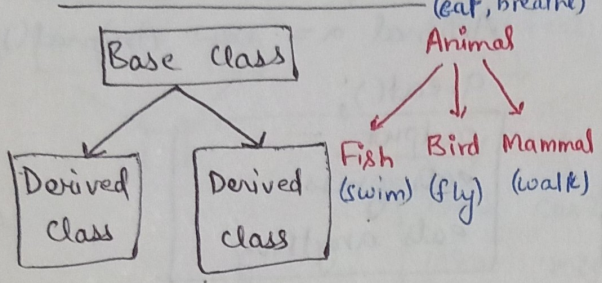


Multi Level Inheritence :-

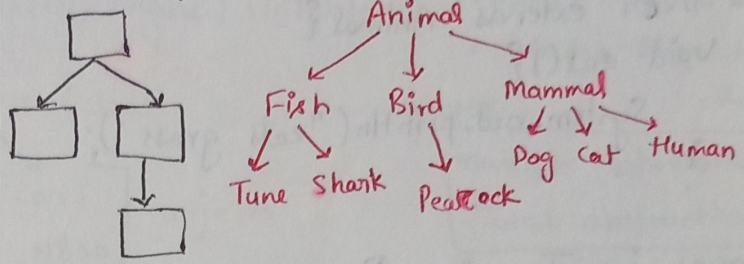




Hierarchial Inheritence :-



Hybrid Inheritence :-



★ **Polymorphism :-** Poly + morphism = Many + forms

→ Compile Time Polymorphism
- Method Overloading

→ Run Time Polymorphism
- Method Overriding

Method Overloading :- Multiple functions with the same name but different parameters.

```
~~Calculator~~{
public ~~Sum~~ class OOPS{
    public static void main(String args[]){
        Calculator calc= new Calculator();
        System.out.println(calc.sum(1,2));
        System.out.println(calc.sum((float)1.5,(float)2.5));
        System.out.println(calc.sum(1,2,3));
    }
}
class Calculator{
    int sum(int a, int b){
        return a+b;
    }
    float sum(float a, float b){
        return a+b;
    }
    int sum(int a, int b, int c){
        return a+b+c;
    }
}
```

Output -
3
4.0
6

Method Overriding :- Parent and ~~class~~ child classes both contain the same function with a different defination.

class A ⟶ class B

Animal (void eat(x) → "eat anything") ⟶ Deer (void eat(x) → "eat grass")

```
class Animal{
    void eat(){
        System.out.println("eats anything");
    }
}
class Deer extends Animal{
    void eat(){
        System.out.println("eats grass");
    }
}
```

In main →

```
Deer d = new Deer();
d.eat();
Animal a = new Animal();
a.eat();
```

output –
eats grass
eats anything

Packages in Java :- Package is a group of similar types of classes, interfaces and sub-packages.

- In-built
- User defined

Eg:- `import java.util.*;` (utility package)

`Scanner sc = new Scanner();`
Scanner → class, sc → object, new → keyword, Scanner() → constructor

★ **Abstraction :-** Hiding all the unnecessary details and showing only the important parts to the user. (दरयाल)

(idea ✓ implementation ✗)

- Abstract Classes
- Interfaces

Abstract Class :-

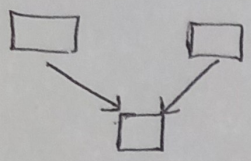
- Cannot create an instance (object) of abstract class.
- Can have abstraction - abstract methods.
- Can have constructors.

```
public class OOPS{
    public static void main(String args[]){
        Mustang myHorse = new Mustang();
        // Animal → Horse → Mustang
    }
}
abstract class Animal{
    String color;
    Animal(){
        System.out.println("animal constructor called");
    }
    void eat(){
        System.out.println("animal eats");
    }
    abstract void walk(); // compulsory
}
class Horse extends Animal{
    Horse(){
        System.out.println("Horse constructor called");
    }
    void changeColor(){
        color = "dark brown";
    }
```

```
    void walk(){
        System.out.println("walks on 4 legs");
    }
}
class Mustang extends Horse{
    Mustang{
        System.out.println("Mustang constructor called");
    }
}
```

Output –
Animal Constructor Called
Horse Constructor Called
Mustang Constructor Called

Interfaces: – Blueprint of a class. • Multiple Inheritance • Total Abstraction



car [wheel, speed, engine] (Interface)

↓

Maruti 800 (class)

car1	car2	car3
M800	M800	M800

objects

- class – blueprint of object
- Interface – blueprint of class

uses →

multiple Inheritance
Total abstraction (100%)

- All methods are public, abstract & without implementation.
- Used to achieve total abstraction.
- Variables in the interface are final, public and static.

```
→public class OOPs{
    public static void main(String args[]){
        Queen q = new Queen();
        q.moves();
    }
}
interface ChessPlayer{
    void moves();
class Queen implements ChessPlayer{
    public void moves(){
        System.out.println("up, down, left,
            right, diagonal (in all 4 dirs)");
    }
}
class Rook implements ChessPlayer{
    public void moves(){
        System.out.println("up, down, left,
                            right");
    }
}
class King implements ChessPlayer{
    public void moves(){
        System.out.println("up, down, left, right
                diagonal (by 1 step)");
    }
}
```

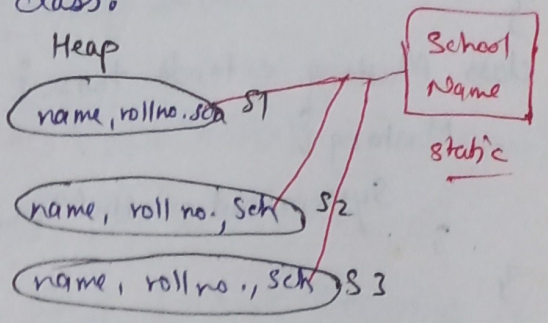
Multiple Inheritance

```
~~implements~~
interface Herbivore{


}
interface Carnivore{


}
class Bear implements Herbivore,
        Carnivore{


}
```

Static Keyword:- Static keyword in Java is used to share the same variable or method of a given class.

- Properties
- Functions
- Blocks
- Nested Classes

- Static ek baar bnta hai saare object ke liye.
- Agar koi bhi object change karta hai to sab me change ho jaega.



Super Keyword:- Super keyword is used to refer immediate parent class object.

→ to access parent's properties
→ to access parent's ~~parent~~ functions
→ to access parent's constructor.

```
Public class OOPS {
    public static void main (String args[]){
        Horse h = new Horse();
        System.out.println(h.color);
    }
}
class Animal{
    String color;
    Animal(){
        System.out.println("Animal Constructor called");
    }
}
class Horse extends Animals{
    Horse(){
        super.color = "brown"; // called animal constructor and change color
        System.out.println("horse constructor called");
    }
}
```


Output:-

Animal constructor ~~is~~ called
horse constructor called
brown

Recursion:- Recursion is a method of solving a computational problem where the solution depends on solution to smaller instances of the same problems.

① Base Case
② Kaam
③ Inner Calls

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

$(n-1) \times f(n-2)$

$(n-2) \times f(n-3)$

$1 \times f(0)$

• Directions:- Top to Down (towards base case)

$f(5) = 5 \times f(4) \rightarrow 24 \leftarrow = 120$

$4 \times f(3) \rightarrow 6$

$3 \times f(2) \rightarrow 2$

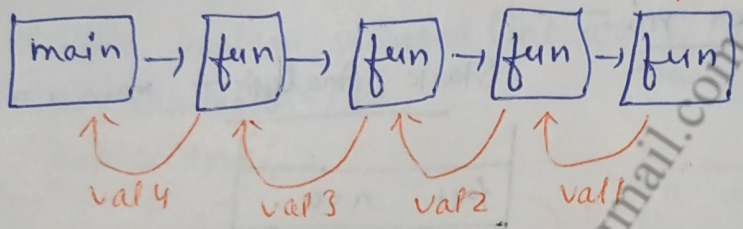
$2 \times f(1) \rightarrow 1$

$1 \times f(0) \leftarrow$ Base Case

1

Combine Soln

↓ B.C.



1. Print numbers from n to 1 (Decreasing Order):-

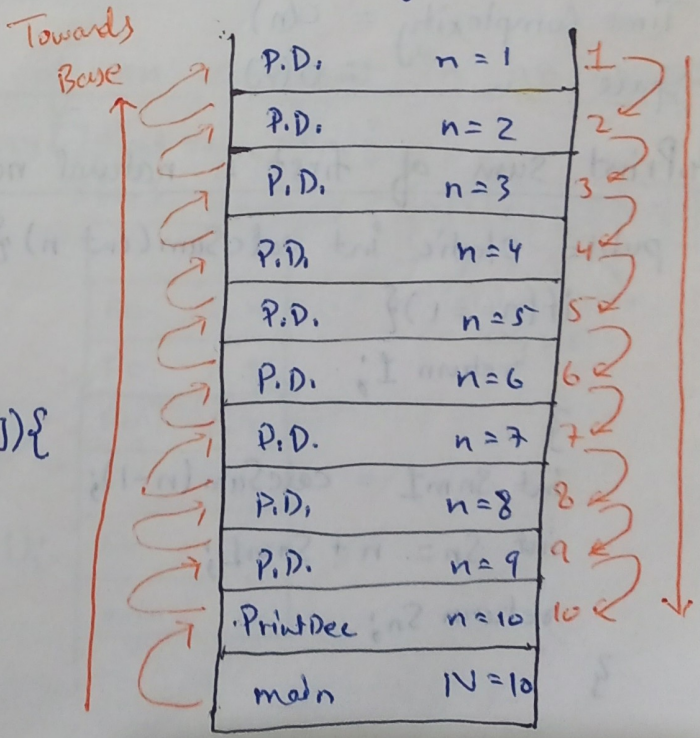
```
public class RecursionBasics{
    public static void printDec(int n){
        if (n == 1){
            System.out.println(n);
            return;
        }
        System.out.print(n + " ");
        printDec(n-1);
    }
Run | Debug
    public static void main(String args[]){
        int n = 10;
        printDec(n);
    }
}
```

Output →

10 9 8 7 6 5 4 3 2 1

Call Stack:-

Decreasing Order:-

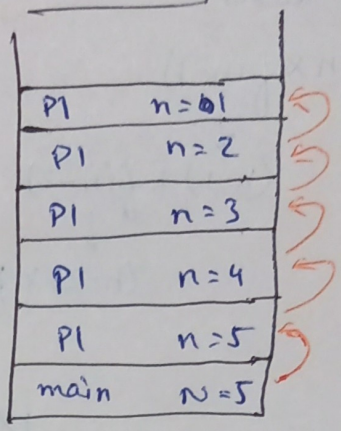


Stack Overflow:- Agar base case nahi denge tab wo infinite loop me chla jaega or memory full hojaega use stack overflow kehte hai.

2. Print numbers from n to 1 (Increasing order):-

```
public static void printInc(int n){
    if(n==1){
        System.out.println(1);
        return;
    }
    printInc(n-1);
    System.out.println(n);
}
```

Call Stack — main → n=5

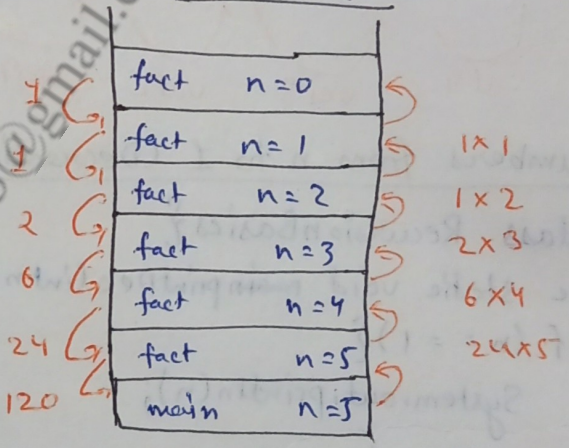


3. Print factorial of a number n:-

```
public static int fact(int n){
    if(n==0){
        return 1;
    }
    int fnm1 = fact(n-1);
    return fn;
}
```

Time Complexity = O(n)
Space " = O(n)

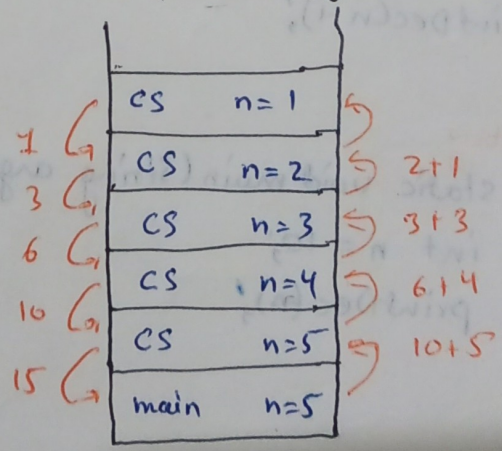
Stack Analysis — main → n=5



4. Print sum of first n natural numbers:-

```
public static int calcSum(int n){
    if(n==1){
        return 1;
    }
    int Snm1 = calcSum(n-1);
    int Sn = n + Snm1;
    return Sn;
}
```

Stack Analysis →

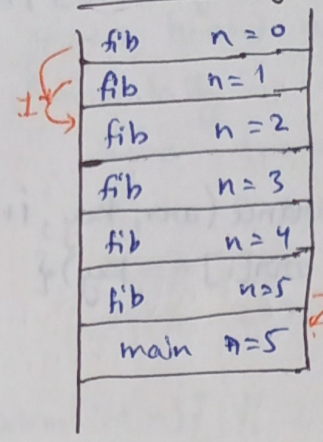


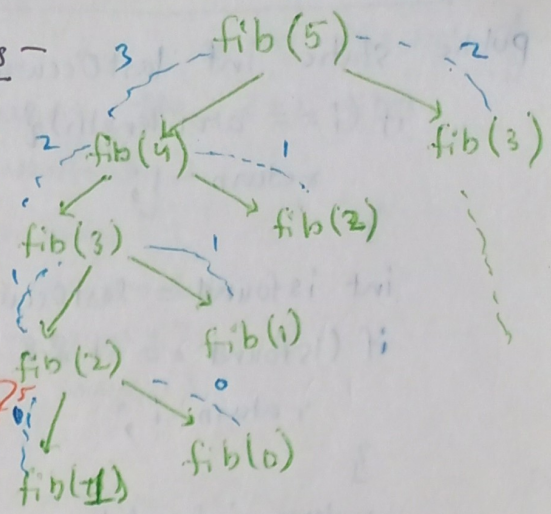
5. Print Nth fibonacci number :-

$fib_n = fib_{n-1} + fib_{n-2}$

```
public static int fib(int n){
    int fnm1 = fib(n-1);
    int fnm2 = fib(n-2);
    int fn = fnm1 + fnm2
    return fn;
    Base Case -
    if(n==0 || n==1){
        return n;
    }
```

Stack Analysis -





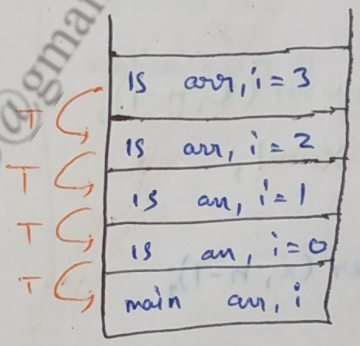
Time Complexity = $O(2^n)$

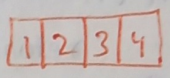
Space " = $O(n)$

6. Check if a given array is sorted or not :-

```
public static boolean isSorted(int arr[], int i){
    if (i == arr.length-1){
        return true;
    }
    if (arr[i] > arr[i+1]){
        return false;
    }
    return isSorted(arr, i+1);
```

Stack Analysis -

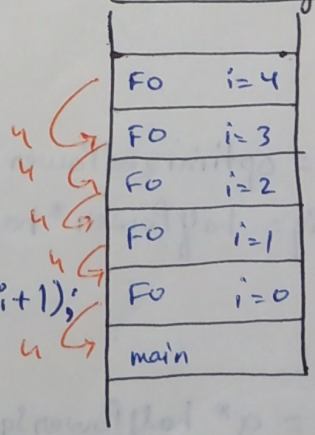


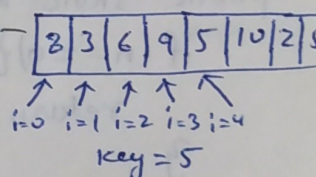


7. WAF to find the first occurance of an element in an array.

```
public static int firstOccurance(int arr[], int key, int i){
    Base Case { if (i == arr.length){
            return -1;
        }
    if (arr[i] == key){
        return i;
    }
    return firstOccurance(arr, key, i+1);
}
```

Stack Analysis -





Time Complexity = $O(n)$

Space " = $O(n)$

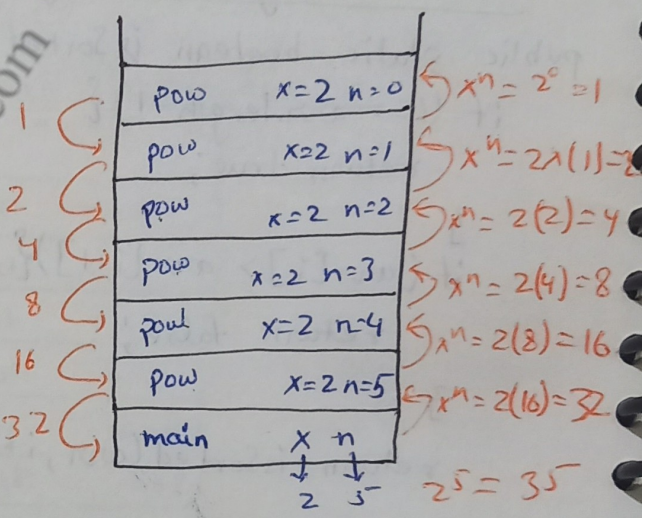
8. WAF to find the last occurance of an element in an array?

```
public static int lastOccurance(int arr[], int key, int i){
    if(i == arr.length){
        return -1;
    }
    int isfound = lastOccurance(arr, key, i+1);
    if(isfound == -1 && arr[i] == key){
        return i;
    }
    return isfound;
}
```

9. Print x to the power n :—

```
public static int power(int x, int n){
    if(n==0){
        return 1;
    }
    // int xnm1 = power(x, n-1);
    // int xn = x * xnm1;
    // return xn;
    return x * power(x, n-1);
}
```

Stack Analysis —

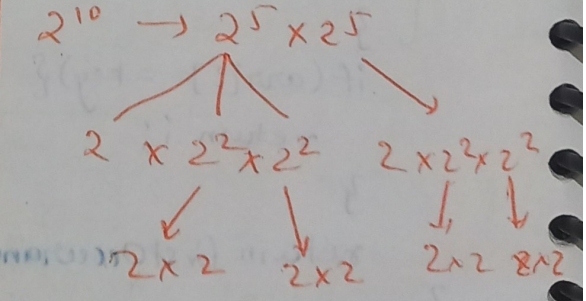


10. Print x^n in O(logn) :—

```
public static int optimizedPower(int a, int n){
    if(n==0){
        return 1;
    }
    int halfPower = optimizedPower(a, n/2);
    int halfPowerSq = halfPower * halfPower;
    // n is odd
    if(n%2!=0){
        halfPowerSq = a * halfPowerSq;
    }
    return halfPowerSq;
}
```

$x^n = x^{n/2} \times x^{n/2}$ (n = even)

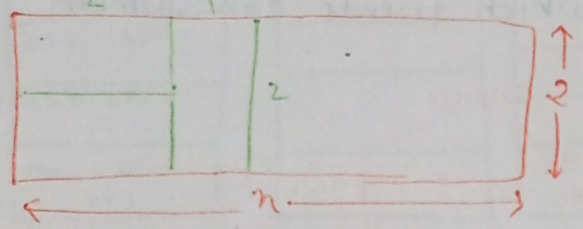
$x^n = x \times x^{n/2} \times x^{n/2}$ (n = odd)



TC = $O(\log_2 n)$

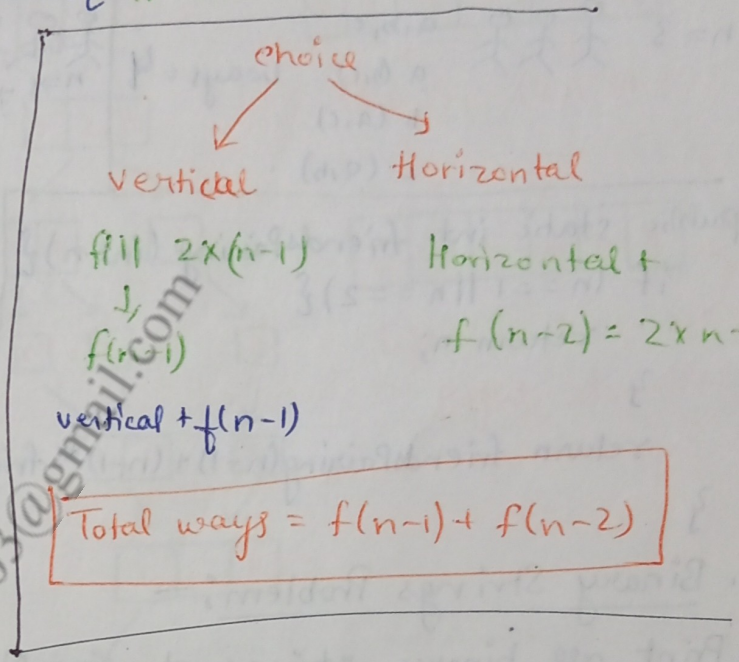
11. Tiling Problem :-

Given a "2xn" floor and tiles of size "2x1". count the no. of ways to ~~the~~ tile the given board using the 2x1 tiles.
(A tile can be placed horizontally or vertically.).



n=0 2x0 → ways → 1
n=1 2x1 → ways → 1
n=2 2x2 → ways → 2
n=3 2x3 → ways → 3

```
public static int tilingProblem(int n){ // 2 x n (floor size)
    // kaam
    // vertical choice
    int fnm1 = tilingProblem(n-1);
    //horizontal choice
    int fnm2 = tilingProblem(n-2);
    int totways = fnm1 + fnm2;
    return totways;
}
// base case
if(n==0 || n==1){
    return 1;
}
```

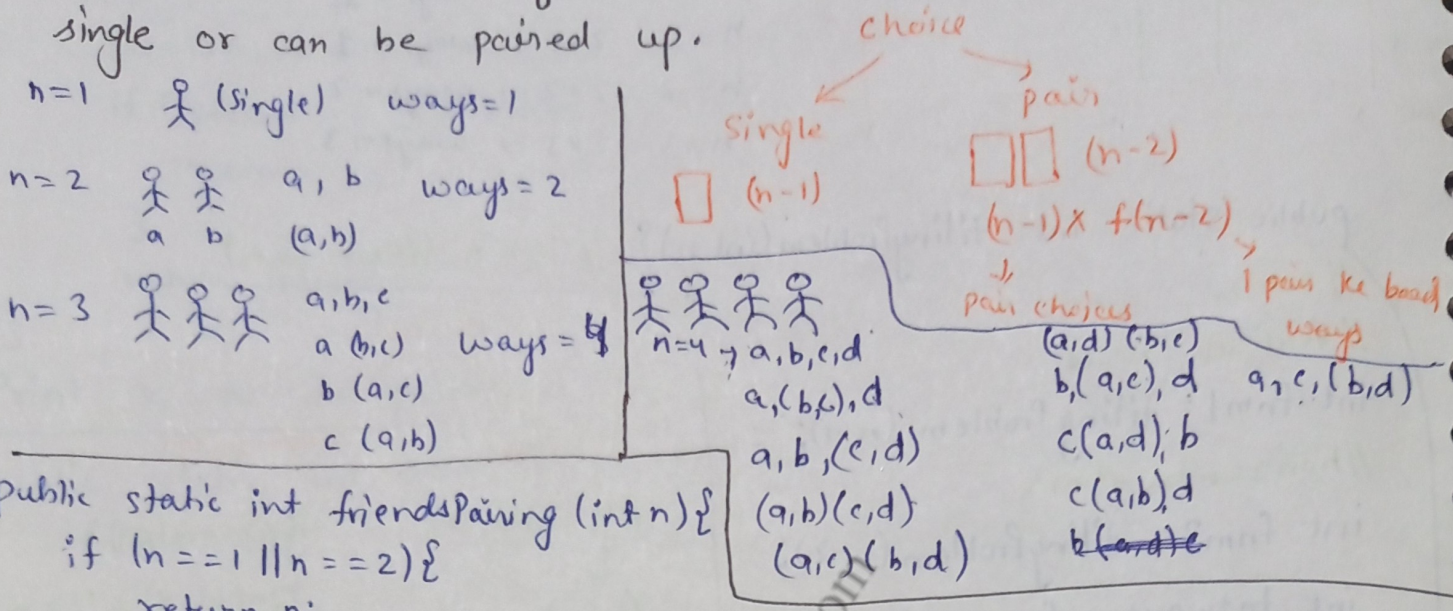


12. Remove Duplicates in a String :-

```
public static void removeDuplicates(String str, int idx, StringBuilder newStr, boolean map[]){
    if(idx == str.length()){
        System.out.println(newStr);
        return;
    }
    // kaam
    char currChar = str.charAt(idx);
    if(map[currChar-'a'] == true){
        // duplicate
        removeDuplicates(str, idx+1, newStr, map);
    }else{
        map[currChar-'a'] = true;
        removeDuplicates(str, idx+1, newStr.append(currChar), map);
    }
}
```

13. Friends Pairing Problem:—

Given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out the total no. of ways in which friends can remain single or can be paired up.

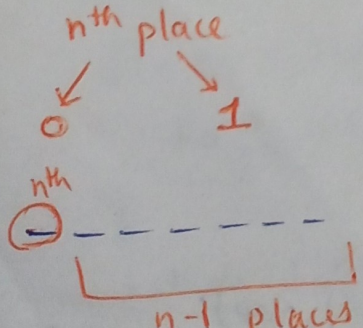



```
public static int friendsPairing (int n){
    if (n==1 || n==2){
        return n;
    }
    return friendsPairing(n-1)+(n-1)* friendspairing(n-2);
}
```

14. Binary Strings Problem:—

Print all binary strings of size N without consecutives ones.

n=0 " "
n=1 "1" "0"
n=2 "00" "01" "10"
n=3 000 001 010 100 101




```
public static void printBinaryStrings(int n, int lastPlace,
                                      String str) {
    // base case
    if (n==0){
        System.out.println(str);
        return;
    }
    // kaam
    printBinaryStrings(n-1,0, str+"0");
    if (lastPlace==0){
        printBinaryStrings(n-1,1, str+"1");
    }
}
```

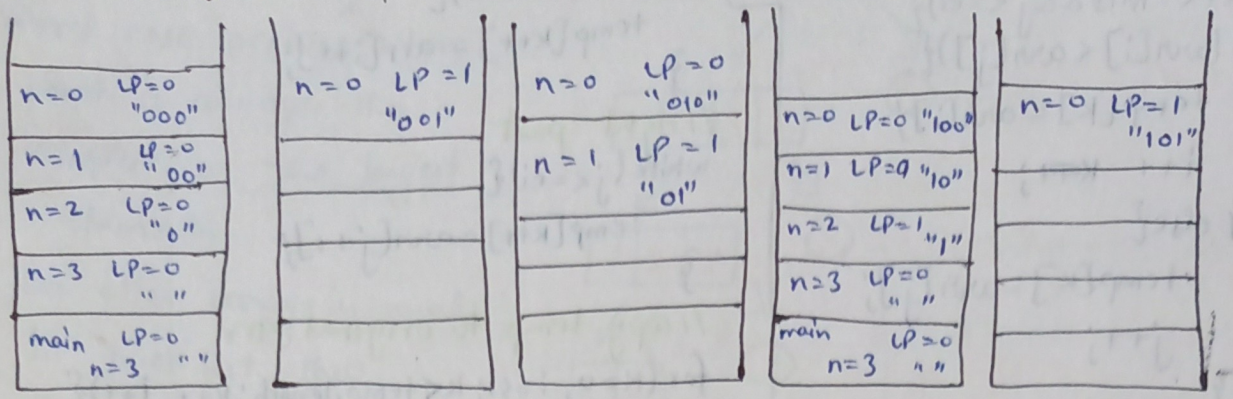
for printBinaryStrings(3,0,""); →

000
001
010
100
101
output

Stack Analysis :-

for n=3

"000"
"001"
"010"
"100"
"101"

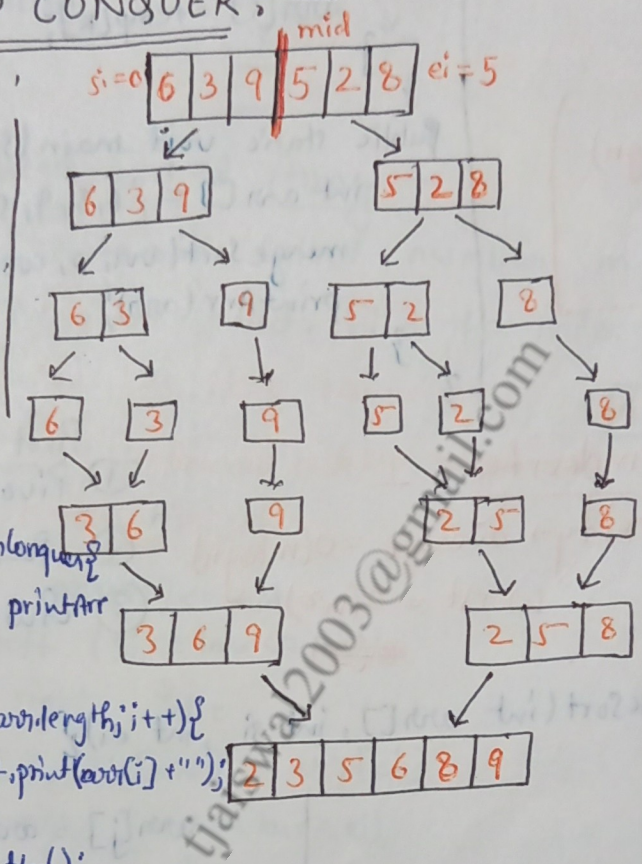


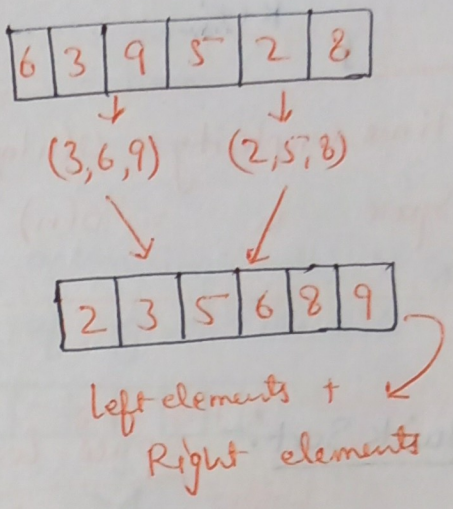
20. DIVIDE AND CONQUER :-

Merge Sort :-

Approach -

① Divide - mid
② mergeSort(left)
mergeSort(right)
③ merge






```
public class DivideConquer{
  public static void printArr
  (int arr[]){
    for(int i=0;i<arr.length;i++){
        System.out.print(arr[i]+" ");
    }
    System.out.println();
  }
  public static void mergeSort(int arr[], int si, int ei){
    if (si >= ei){
       return;
    }
    //kaam
    int mid = si + (ei-si)/2; //(si+ei)/2
    mergeSort(arr, si, mid); //left part
    mergeSort(arr, mid+1, ei); // right part
    merge (arr, si, mid, ei);
  }
  public static void merge (int arr[], int si, int mid, int ei){
       int temp[] = new int[ei-si+1];
       int i = si; //iterator for left part
       int j = mid+1; //iterator for right part
```

```
int k=0; // iterator for temp arr
while(i<=mid && j<=ei){
    if(arr[i] < arr[j]){
        temp[k] = arr[i];
        i++ k++;
    } else{
        temp[k] = arr[j];
        j++;
    }
    k++;
}
```

Time Complexity = O(n log n)
Space " = O(n)

```
// left part
while(i<=mid){
    temp[k++] = arr[i++];
}
// right part
while(j<=ei){
    temp[k++] = arr[j++];
}
// copy temp to original arr
for(k=0, i=si; k<temp.length; k++, i++){
    arr[i] = temp[k];
}
}
Public static void main(String args[]){
    int arr[] = {6,3,9,5,2,8,-2};
    mergeSort(arr, 0, arr.length-1);
    printArr(arr);
}
}
```

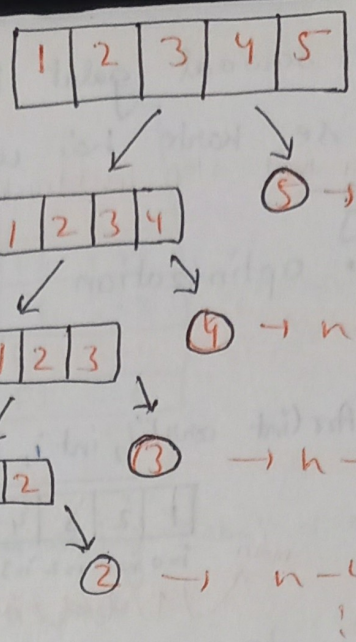
Quick Sort :- Space Complexity = 1

Time Complexity → average = O(n log n)
worst = O(n²)
~~O(...)~~

Pivot & Partition :
① Pivot (last element)
② Partition (parts)
③ QuickSort (left)
" (right)

```
public static void quickSort(int arr[], int si, int ei){
    if(si >= ei){
        return;
    }
    // last element
    int pIdx = partition(arr, si, ei);
    quickSort(arr, si, pIdx-1); // left part
    quickSort(arr, pIdx+1, ei); // right part
}
public static int partition(int arr[],
                            int si, int ei){
    int pivot = arr[ei];
    int i = si-1; // to make place for element smaller than pivot
    for(int j=si; j<ei; j++){
        if(arr[j] <= pivot){
            i++;
            // swap
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
        }
    }
    i++
    int temp = pivot;
    arr[ei] = arr[i];
    arr[i] = temp;
    return i;
}
```

Worst Case Important — Sorted Array



$n + (n-1) + (n-2) + (n-3) \cdots$

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

$= \frac{n^2+n}{2}$

$= O(n^2)$

Worst case occurs when pivot is always the smallest or the largest element.

- No other array is used in quick sort thus their space complexity is constant.

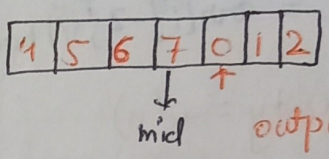
Search in Rotated Sorted Array —

Input: Sorted, rotated array with distinct number (in ascending order) it is rotated at a pivot point. Find the index of given element.

Linear Search se bhi kar skte hai → O(n)

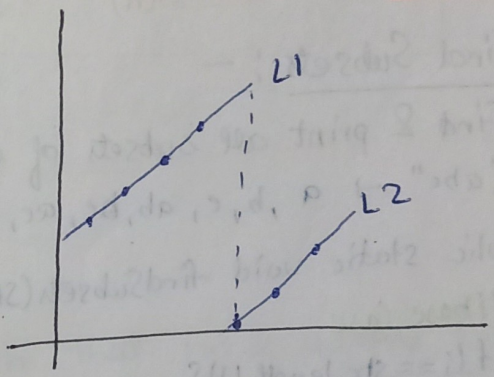
Rotated Sorted Array (optimized code) → nlogn

↳ Modified Binary Search



Case 1: mid on L1. arr[si] <= mid
- Case a: L1 left (si <= tar <= mid)
- Case b: mid right (else)

Case 2: mid on L2. arr[mid] <= arr[ei]
- Case c: L1 right (mid <= tar <= ei)
- mid left (else)




```
public static int search(int arr[], int tar, int si, int ei){
    if(si > ei){
        return -1;
    }
    //kaam
    int mid = si + (ei-si)/2; // (si+ei)/2
    // case found
    if(arr[mid] == tar){
        return mid;
    }
    //mid on L1
    if(arr[si] <= arr[mid]){
        //case a: left
        if(arr[si] <= tar && tar <= arr[mid]){
            return search(arr, tar, si, mid-1);
        }
        else{
            //case b: right
            return search(arr, tar, mid+1, ei);
        }
    }
    //mid on L2
    else{
        // case c: right
        if(arr[mid] <= tar && tar <= arr[ei]){
            return search(arr, tar, mid+1, ei);
        }else{
            // case d: left
            return search(arr, tar, si, mid-1);
        }
    }
}
```

Backtracking:- Jab koi sawaal galat ho jata hai to wapas aakar dusre trh se karte hai use backtracking kehte hai.

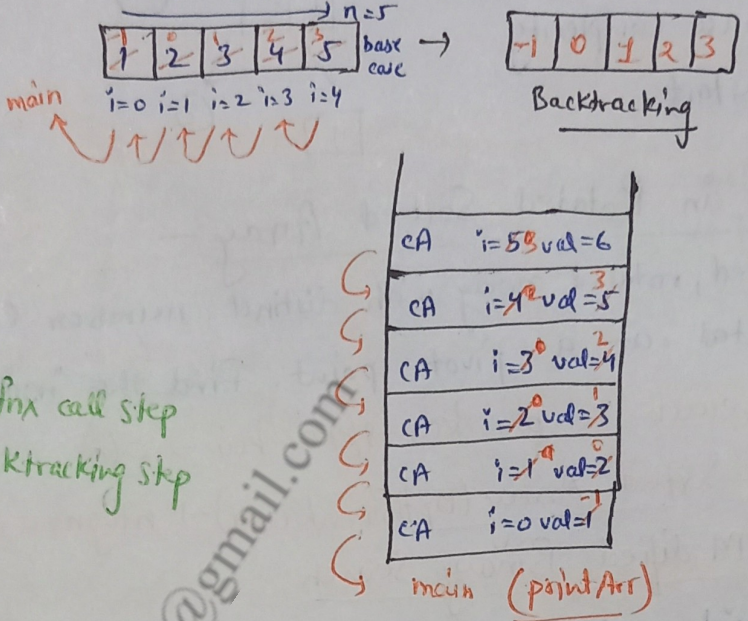
Recursion ka next step backtracking hota hai.

Types of Backtracking -

- Decision
- optimization
- Enumeration.

Backtracking - Arrays -

```
public static void changeArr(int arr[], int i, int val){
    //base case
    if(i==arr.length){
        printArr(arr);
        return;
    }
    //recursion
    arr[i] = val;
    changeArr(arr, i+1, v+1); //fnx call step
    arr[i] = arr[i]-2; //backtracking step
}
```



Time Complexity = O(n)

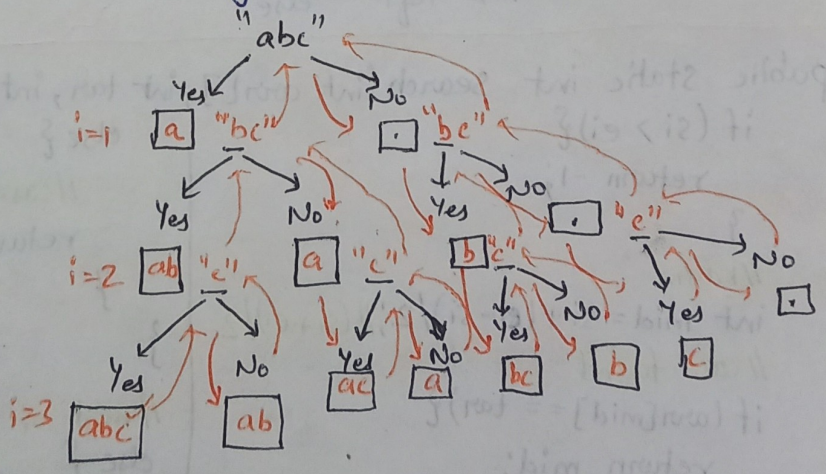
Space " = O(n)

Find Subsets ; -

Find & print all subsets of a given string.

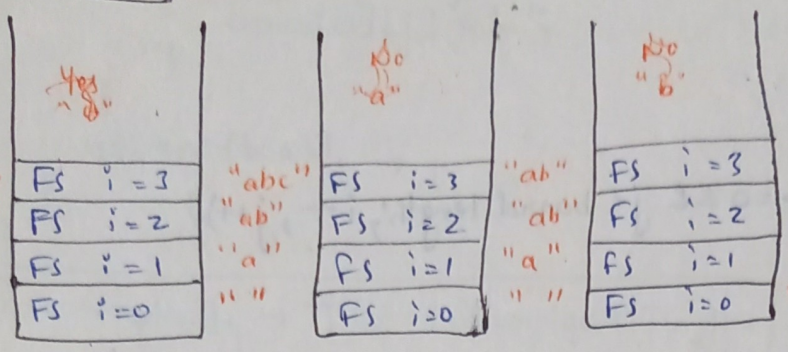
"abc" → a, b, c, ab, bc, ac, abc, "" → empty set = 2^n subsets.

```
public static void findSubsets(String str, String ans, int i){
    //base case
    if(i==str.length()){
        if(ans.length()==0){
            System.out.println("null");
        }else{
            System.out.println(ans);
        }
        return;
    }
    //Yes choice
    findSubsets(str, ans+str.charAt(i), i+1);
    //No choice
    findSubsets(str, ans, i+1);
}
```



abc
ab
ac
a
bc
b
c
"" (null)

Call Stack –



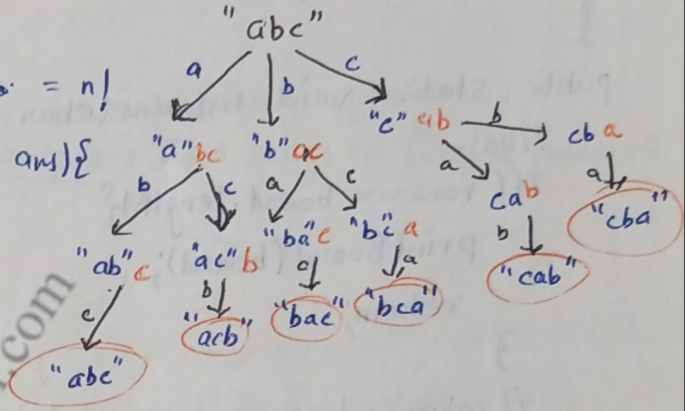
Time complexity = $O(n*2^n)$
Space " = $O(n)$

Find Permutations –

Find & print all permutations of a string. "abc" → abc, acb, bac, bca, cab, cba.

n element → n! ____ ____ ____ ____ n spaces

n (n-1) (n-2) (n-3) ... = n!



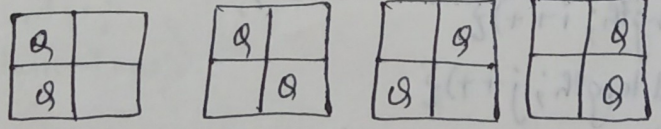

```
public static void findPermutation(String str, String ans){
    //base case
    if(str.length()==0){
        System.out.println(ans);
        return;
    }
    //recursion
    for(int i=0; i<str.length(); i++){
    char curr = str.charAt(i);
    //"abcde" => "ab"+"de" = "abde"
    String NewStr= str.substring(0,i)+str.substring(i+1);
    findPermutation(NewStr, ans+curr);
    }
}
```

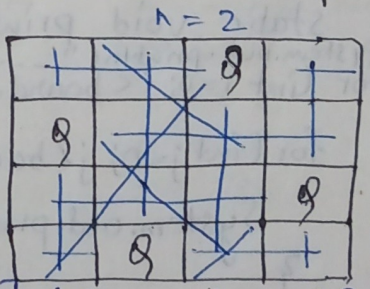
Time - factorial = $n*n!$

N-Queens –

Place N queens on an NXN chessboard such that no 2 queens can attack each other.

for n = 2.






```
public class Classroom{
    public static boolean IsSafe(char board[][], int row, int col){
        // vertical up
        for(int i=row-1; i>=0; i--){
            if(board[i][col]=='Q'){
                return false;
            }
        }
        //diagonally left up
        for(int i=row-1, j=col-1; i>=0 && j>=0; i--, j--){
```

```
        if (board[i][j] == 'Q'){
            return false;
        }
    }
    //diagonally right up
    for(int i=row-1, j=col+1; i>=0 && j<board.length; i--, j++){
        if (board[i][j] == 'Q'){
            return false;
        }
    }
    return true;
}
public static void nQueens(char boards[][], int row){
    //base
    if(row == board.length){
        printBoard(board);
        return;
    }
    // column loop
    for(int j=0; j<board.length; j++){
        if(isSafe(board, row, j)){
            board[row][j] = 'Q';
            nQueens(board, row+1); //function call
            board[row][j] = 'x'; //backtracking step
        }
    }
}
public static void printBoard(char board[][]){
    System.out.println("------chess board------");
    for(int i=0; i<board.length; i++){
        for(int j=0; j<board.length; j++){
            System.out.print(board[i][j] + " ");
        }
        System.out.println();
    }
}
public static void main(String args[]){
    int n=2;
    char board[][] = new char[n][n];
    // initialize
    for(int i=0; i<n; i++){
```

```
        for(int j=0; j<n; j++){
            board[i][j] = '.';
        }
    }
    nQueens(board, 0);
  }
}
```

Output →

------ chess board ------
x Q x x
x x x Q
Q x x x
x x Q x
------ chess board ------
x x Q x
Q x x x
x x x Q
x Q x x

Time Complexity → $T(n) = 1 \text{ Queenplace} * T(n-1) + \text{isSafe}()$

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

$T(n) = O(n!)$

N-Queens - count ways

Count total no. of ways in which we can solve N Queens problem.

```
// base
if(row == board.length){
    // printBoard(board);
    count++;
    return;
}
static int count = 0;
```

In main → System.out.println("Total ways to solve n queens = " + count);

Output → Total ways to solve n queens = 2 for n = 4
n = 5
" " " " " = 10

N Queens - print 1 Solution

Check if problem can be solved & print only 1 solution to N Queens problem.

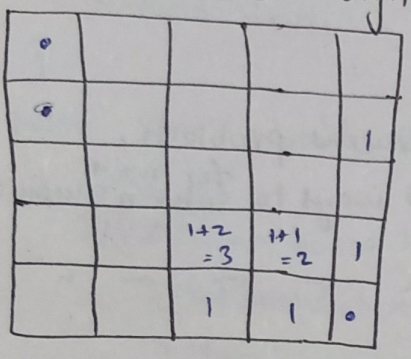
```
public static boolean nQueens(char board[][], int row){
    //base
    if(row == board.length){
        //printBoard(board);
        count++;
        return true;
    }
    //column loop
    for(int j=0; j<board.length; j++){
        if(isSafe(board, row, j)){
            board[row][j] = 'Q';
            if(nQueens(board, row+1)){
                return true;
            }
            board[row][j] = 'x'; //backtracking step
        }
    }
    return false;
```

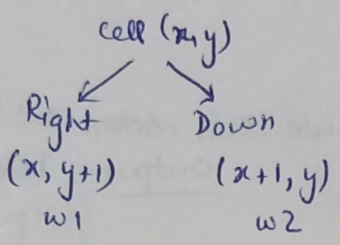
Output → Solution is possible.
------ chess board ------
x Q x x
x x x Q
Q x x x
x x Q x

In main :
```
if(nQueens(board, 0)){
    System.out.println("Solution is possible");
    printBoard(board);
}else
    System.out.println("Solution is not possible");
}
```

Grid ways :-

Find number of ways to reach from (0,0) to (N-1, M-1) in a NXM Grid.
Allowed moves – right or down.





w1 + w2 = Total ways

f(x,y) = f(x+1, y) + f(x, y+1)
Down Right

```
public class Classroom{
    public static int gridways(int i, int j, int n, int m){
        //base case
        if(i==n-1 && j==m-1){
            return 1;
        }else if(i==n || j==n){
            return 0;
        }
        int w1 = gridways(i+1, j, n, m);
        int w2 = gridways(i, j+1, n, m);
        return w1+w2;
    }
    public static void main(String args[]){
        int n=3, m=3;
        System.out.println(gridways(0,0,n,m));
    }
}
```

Output = 6

6	3	1
3	2	1
1	1	1

Time complexity →
Right turns = no. of cols = m
Left turns = no. of rows = n
Total = n+m
= $O(2^{n+m})$

Sudoku Code :—

```
public class Classroom{
    public static boolean isSafe(int sodoko[][], int row, int col, int digit){
    //column
    for(int i=0; i<=8; i++){
        if(sudoko[i][col]==digit){
```

```
        return false;
    }
}
//row
for(int j=0; j<=8; i++){
    if(sudoko[row][j]==digit){
        return false;
    }
}
//grid
int sr = (row/3)*3;
int sc = (col/3)*3;
//3x3 grid
for(int i=sr; i<sr+3; i++){
    for(int j=sc; j<sc+3; j++){
        if (sudoko[i][j]==digit){
            return false;
        }
    }
}
return true;
}
public static boolean sudokoSolver
(int sudoko[][], int row, int col){
    //base case
    if(row==9){
        return true;
    }
    //recursion
    int nextRow= row, nextCol= col+1;
    if(col+1 ==9){
        nextRow =row+1;
        nextCol = 0;
    }
    if(sudoko[row][col] != 0){
        return sudokoSolver(sudoko,
                nextRow, nextCol);
    }

    for(int digit =1;digit<=9;digit++){
        if(isSafe(sudoko, row, col, digit)){
            sudoko[row][col] = digit;
            if(sudokoSolver(sudoko,
                nextRow, nextCol){
                return true;
            }
            sudoko[row][col] = 0;
        }
    return false
}
public static void printSudoko(int sudoko[][]){
    for(int i=0; i<9; i++){
        for(int j=0; j<9; j++){
            System.out.print(sudoko[i][j]+" ");
        }
    }
}
public static void main(String args[][]){
    int sudoko[][] ={{8,0,6,0,0,0,0,0,0},
    {4,9,0,1,5,7,0,0,2},{0,0,3,0,0,4,1,9,0},
    {1,8,5,0,6,0,0,2,0},{0,0,0,0,2,0,0,6,0},
    {9,6,0,4,0,5,3,0,0},{0,3,0,0,7,2,0,0,4},
    {0,4,9,0,3,0,0,5,7},{8,2,7,0,0,9,0,1,3}};
    if(sudokoSolver(sudoko,0,0)){
        System.out.println("solution exists");
        printSudoko(sudoko);
    }else{
        System.out.println("solution does not exist");
    }
}
}
```

Output:-

Solution exists

2	1	8	3	9	6	7	4	5
4	9	6	1	5	7	8	3	2
7	5	3	2	8	4	1	9	6
1	8	5	7	6	3	4	2	9
3	7	4	9	2	8	5	6	1
9	6	2	4	1	5	3	7	8
5	3	1	6	7	2	9	8	4
6	4	9	8	3	1	2	5	7
8	2	7	5	4	9	6	1	3