Eco Friendly

N. Nature™

YOUR CHILD IS OUR PRIORITY

BIG BEN
LONDON

#NOTEBOOKS

A4

student exercise

notebook

Recycle Recyclable Reuse

easy to write

INDEX

Name: Rohit Kumar Yadav

Class: B.Tech CSE Subject: Java Roll No.: 2023101011400009

Serial No.	Date	Title	Page No.	Teacher's Sign/Remarks

"Consistency is the key to success"

Serial No.	Date	Title	Page No.	Teacher's Sign/Remarks
✗		Complete Java		
✗		Complete DSA		
✗		Live Resume Preparation		
✗		Live Interview Preparation		
✗		Coding Question on all important topic		

Topic

Catagory — Basic

- Basic of programing
- Loop function & Shorting
- Array ; 2D Array
- 2D Array & String

16 lecture → 35 days → 5

DSA

- Problem solving technique
- Object oriented programing
- Linear Data Structure
- Tree
- Advanced Data structure
- Dynamic Programing

33 lecture → 66 day → 5 days

Path of Placement

Step1 — Learn a language

Step2: — DSA

Step3. — Project, theory sub, Resume, Refaal, Apply & Interview preparation

DSA

Problem Solving technique * Recursion * Bit Manipulation
* Time space Complexity

(1) First Month...
* Basic → Java
* Basic D.S - Array, String → LeetCode / GFG
* Time & space Analysis, Searching, Sorting

(2) Second Month.
* Recursion & Advance sorting
* Prefix sum, sliding window, Math (Sieve)
* OOPS, linked list.

(3) Third Methods =
* Stack & Queue
* Tree
* Hashmap & Heap (Dusta ther implementation)

(4) Forth Month.
* Dynamic Programming
* Greedy & Backtracking
* Bit Manipulation
* graph

Java

First Program:-

```
public class Javaclass {
    public static void main(String arg[]){

    }
}  // boilerplate Code
```

output in Java

```
System.out.print("Hello World");
```
→ statement terminator

print → function, "Hello World" → output

Example:

```
public class Javaclass{
    public static void main(String arg[]){
    System.out.print("Hello world");
    }
}
```

output: Hello World

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

or

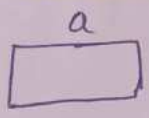
```
System.out.print("Hello World"\n"my name is
    mohit kumar yadav"\n"Rohit");
```

Point Pattern:-

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

```
public class JavaClass {
public static void main(String Arg[]){
System.out.print("*****\n****\n***\n**\n*");
```

Variable in Java

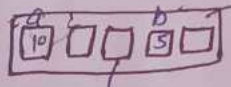


Perimeter = 2*(a+b)

Literal (Constant) ✓ Variable

a=10
b=20

int a = 10; (a,b,c Variable) Identifier
int b = 5;

Memory:- 

(Blocksize) String name = "Tony Stark";

Tony stark

* public class JavaBasic {
public static void main(String Arg[]){
int a=5;
system.out.print(a);

output: 5 } }

a=5;
system.out.println(a);
}

} output=a

Data type in Java

Primitive (already exist) — Non primitive (create)

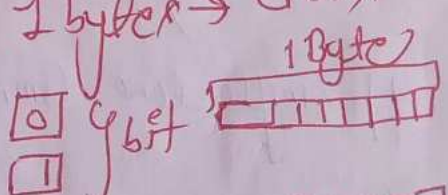
byte, short, int, long, float, double, char, boolean — {String, Array, class, objects, interface}

Java → Typed language { 3.14 (float), 10.5 decimal

Example:

```
public class forabasic {
  public static void main(String Arg[]){
    byte b = 8;
    System.out.println(b);
    char ch = 'a';
    System.out.println(ch);
    boolean var = true;
    System.out.print(var);
    float price = 10.5;
    int number = 25;
    // long
    // double
    short n = 240;
  }
}
```

1 byte → 8 bits



1 Byte → [-128 to +127]

short → 2 byte

char → 2

boolean → 1

int → 4

long → 8

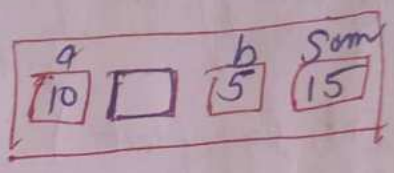
float → 4

double → 8

[Sum of a & b]

System.out.print( );

```
int a = 10;
int b = 5;
int sum = a+b;
System.out.print("Sum");
```



Comment in Java

Eg

```
public class JavaBall{
    public static void (String args[]){
        // Code to Calculate Sum or
    int a = 10;
    int b = 5;
    int sum = a+b;
System.out.println(sum);
    }
}
```

// single line Comment

/* ↔ */ multiline Comment.

Input in Java

```
import java.util.*;
public class Javabasic{
    public static void main(String Arg[]){
        Scanner sc= new Scanner(System.in);
        String input = sc.next();
        System.out.println(input);
    }}
```

or

```
|| String input =sc.next();
||    System.out.println(name);
int number=sc.nextLine();
System.out.println(name);
```

or

```
int number= sc.nextInt();
System.out.println(number);
```

output: Rohit
Rohit

& next
next line
nextInt
next byte
next float
next double
next short
next long

sc.nextDouble()

Sum of a & b

```
int a = sc.nextInt();
int b = sc.nextInt();
int sum= a+b;
```

System.out.println(sum);

Code:

```
import java.util.*;
Public class JavaBasic{
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

```
Scanner
  input a.(sc.nextInt)
  input b  (sc.nextInt)
  int product = a*b;
  syso(product)
```

```
import java.util.*;
Public class JavaBasic{
  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

(1) rad =

$$area = \pi r^2$$

```
float area = 3.14f * rad * rad *
sys( Area);
```

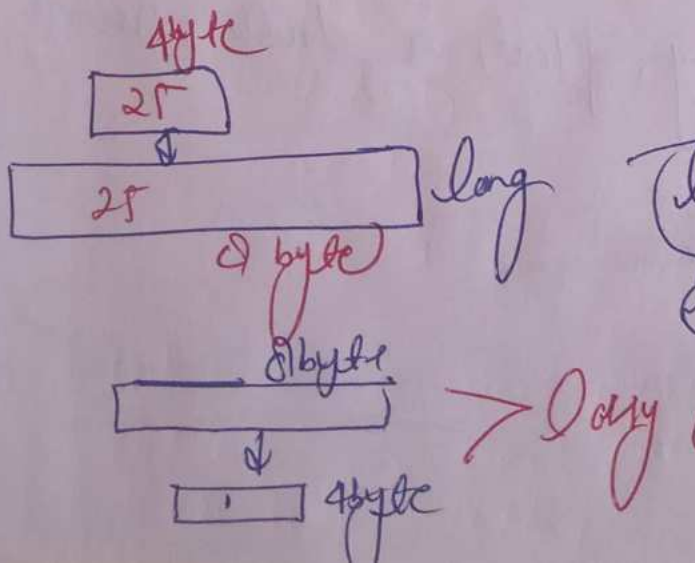
Type Conversion

Conversion happen when :-

(a) type Compatible int ↔ float int → boolean

(b) distination type > source type

byte > short > int > float > log > double



Type Casting

Java not allow but using

```
float a = 25.0;
int b = a;
```

```
float a = 25.12f;
int b = (int) a;
System.out.println(b);
```

```
int b = (int) a;
```

Type Promotion

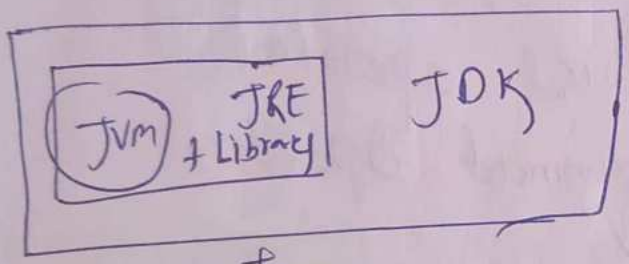
① Java automatically promotes each ~~type~~ byte, short or Char operand to int when evaluating an expression.

② If one operand is long, float, or double, the whole expression is promoted to long, float, or double respectively.

Eg

```
Char a = 'a';
Char b = 'b';
System.out.println(b-a)
```

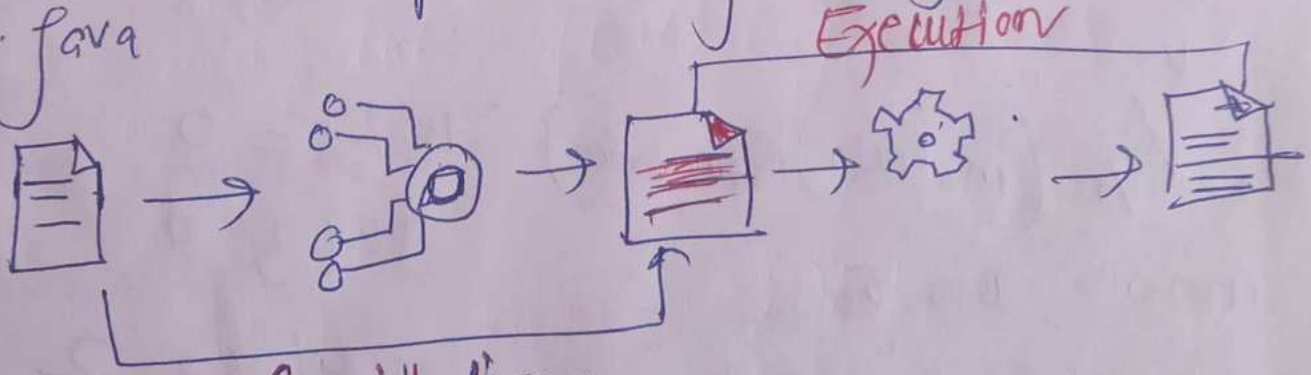
How Code is Running



Java Runtime Environment

SourceCode → Compiler → ByteCode → JVM → NativeCode

.java



G Practice :-

Day 2 Operator in Java

- Symbol that tells compiler to perform some operation.

Sum = a + b

operand, operand, operator = (+, -, *, /)

Type of operators:-
1. Arithmatic operator (Binary / Unary)
2. Relational operator
3. Logical operator
4. Bitwise operature
5. Assignment Operator.

Arithematic Operator

Binary
+
-
*
/
% (modulo) Remainder)

Unary
++
--

10 % 5 = 0
12 % 3 = 0

A = 10 B = 5

A + B = 10 + 5 = 15
A - B = 10 - 5 = 5
A * B = 10 * 5 = 50
A / B = 10 / 5 = 2

14 % 3 = 2

Program:

```
int A =10;
int B=5;
System.out.println("add = "+(A+B));
System.out.println("Subtract="+(A-B));
System.out.println("multiplication="+(A*B));
System.out.println("Divide="+(A/B));
System.out.println("module(Remainder)="+(A%B));
}
}
```

Unary operator:-

a= a+1 ⇒ a++ or ++a increcment
a= a-1 a-- , --a → Quecrement

Preincrement पहले

++a

① Value change करेगा
② Value use.

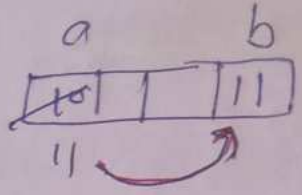
post increcment बाद में

a++

```
int a = 1;
int b = ++a;
System.out.println(a);
System.out.println(b);
```

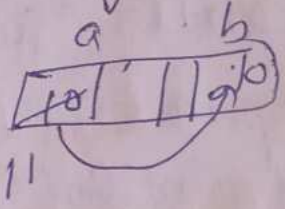
output:- 1
1

int a = 10;
int b = ++a;
a+1



Post increment:
(1) Value use
(2) Value change.

int a = 10;
int b = a++;



output: 11
10

Post Decrement
(1) Value use
(2) Value change.

int a = 10;
int b = a--;

output 9
10

Pre Decrement
(1) Value change
(2) Value use

int a = 10;
int b = --a;

S.O.pl 9
9

② Relational operators:

== → equal
!= → Not equal
>
<
>=
<=

A == B → false
(10) (5)

```
int A=5;
int B=6;
S.O.println((A==B));
```

output → false.

```
int A=10;
int B=10;
S.O.println(A!=B));
```

`!=`

`>`

```
int a=5;
int b=6;
S.O.println((a>b));
```

output → false

`>=`

```
int a=10;
int b=10;
S.O.println(A<=B));
```

→ True.

Logical operator:

&& (Logical AND)
|| (Logical OR)
! (Logical Not)

3>2 true
1!=1 false
5>10 false

Ans	Statement 1	Stm 2
T	T	T
T	T	F
T	F	T
F	F	F

|| (Logical OR)

```
S.O.println((3>2) && (5>0));
```

output → True

(3>2) && (5>0);

False

And	St1	St2
T	T	T
T	T	F
T	F	T
F	F	F

(0 → Logical Not)

Not	St1
F	T
T	F

System.out.println((3>2) || (5>1));

output - True

}}

true → false

false → true

System.out.println(!(0>5));

}}

Assignment Operators

=	A = B	A = 5 (or) 5
+=	A = A + 10 / A += 10	B = B - 5 / B -= 5
-=		
*=		
/=		

```
int A = 20;
A = A + 10;
S.o.println(A);
```

output → 20

or

```
A += 10;
B = B - 5;
B -= 5;
print(B);
```

```
int B=5;
B*=5;
print(B)
```

B=B or B= B*5;

output → 25

Day 3 (Conditional Statement)

Condition Statement:-

If, else

else if

Ternary operater

Switch

If-else statement

```
if (condition){
    ≡
}
else{
    ≡
}
```

condition → a>b, a<b, a>=b, a==b → true

Eg

```
int age = 22;
if (age >=18)
Sys.O( "Date, drive");
}
else {
    syso( "not Adult");
}
```

```
int age = 22;
if ( age >= 18){
    System.out.println("Adult : drive, Dat");
}
else {
    System.out.println("not Adult");
}
```

② `if (age > 13 && ag < 19){`

```
    System.out.println("teenaged");
}
```

Print the largest of 2:-

A=1, B=3

A > B ? if true → print "A"

else false → print "B"

```
if (A >= B){
    Syso("A");
}
else {
    Syso("B");
}
```

Q. int A = 1;
int B = 5;
if (A >= B) {
System.out.println("A is largest of 2");
} else {
System.out.println("B is largest of 2");
}

#3 Print if a number is Even or Odd:—

4 (even) → divisible of 2 or multiple of 2.

a % b = 0

1 (odd) even % 2 = 0

n % 2 == 0 → True (Even) else print("odd")

Code:

```
import java.util.*;
public class Javabasics1 {
    public static void main(String args[]);
    Scanner sc = new Scanner(System.in);
    int number = sc.nextInt();
if (number % 2 == 0) { System.out.println("Even");
else { S.out("odd");
```

Else if Statement:-

```
if (Condition 1) {
}
else if ( Condition 2) {
}
else {
}
```

→ if (cond1) { } True
→ if (Condition2) { } ✗

Code:

```
int age = 13;
if (age >= 18) {
    System.out.println("adult");
}
else if ( age >= 13 && age < 18);
{ System.out.println("teenager");
}
else {
    System.out.println("child");
}
}
```

Income Tax Calculator:-

① Income < 5 l
0% Tax

```
int income
int tax (calculate)
```

② Income between 5-10L

20% tax

income > 10L

30% tax

```
if (income < 5L){
    tax = 0;
}
else if (income >= 5L && income <= 10L)
    tax = income * (0.2)
}
else{
    tax = income * (0.3)
```

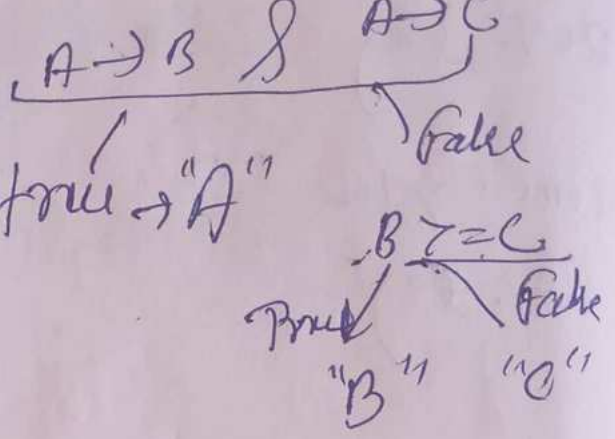
Code:-

```
Scanner sc = new Scanner(System.in);
int income = sc.nextInt();
if (income < 500000){
    tax = 0;
else if (income >= 500000 && incon. < 1000000){
    tax = (int) income * 0.2);
else {
    tax = (int)(income * 0.3);
}
System.out.println("your tax is: " + tax);
```

output:-

Print the largest of 3 Numbers

A=1 B=3 C=6




```
if (A>=B) && (A>=C){
    print("A")
}
else if (B>=C){
    print("B")
}
else print("C")
```

Code:

```
int A=1, B=3, C=6;
if (A>=B) && (A>=C){
    System.out.print("largest of A");
}
else if (B>=C){
    System.out.print("largest of B");
}
else {
    System.out.println("largest of C");
}
}
```

Ternary operator

Variable = Condition ? statement1 : statement2;

Check → true / false

Ex boolean larger = (5>3) ? 5 : 3;

true → statement1 print होगा

String type = (5%2==0) ? "even" : "odd"

Code:-

```
int number = 4;
// ternary operator
String type = ((number%2)==0) ? "even" : "odd";
System.out.println(type);
```

Check Student Pass or fail:-

marks >= 33 : Pass

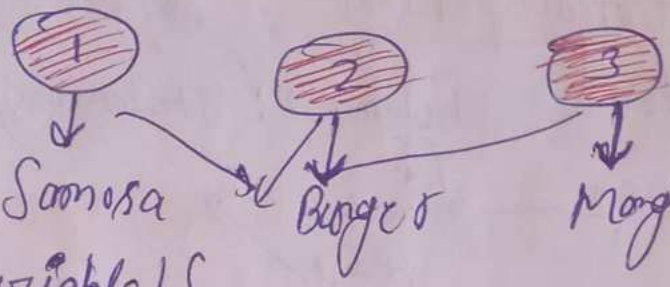
marks < 33 : fail

```
int marks
String status = marks>=33 ? "Pass" : "fail";
print(status)
```

Eg

```
int marks = 67;
String reportCard = marks>=33 ? "Pass" : "fail";
print(reportCard);
```

Switch:



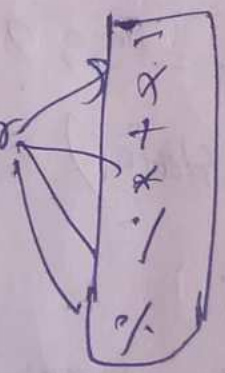

```
Switch (variable) {
  Case1:
  Case2
  Case3
  default:
}
```

```
int number=2;
Switch (number) {
  Case1: System.out.print("Samosa");
         Break;
  Case2: // ("Burger");
         Break;
  Case 3: // ("mango shake");
         Break;
  Case default:- ("We were out");
}
```

Calculator

+, -, *, /, %

int a
int b
Char → operater



Code:-

```
Scanner Sc = new Scanner (System.in);
System.out.println("Enter a:");
int a = Sc.nextInt();
System.out.println("Enter b:")
int b = Sc.nextint();
System.out.println("Enter operator:");
Char(operator){
Case '+': System.out.println(a+b);
          break;
Case '-': system.(a-b);
Case'x': break; (axb);
Case '/': ——— (a/b);
Case '%': ——— (a%b);
default: ——— ("Wrong Calculator");
}
```

Day 4 "Loop"

↳ For repeating → Loop

Ex
```
System.out.println("Hello World");
system.out.println("Hello World");
```

↓

1. while loop
2. for loop
3. Do while loop

while loop

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

```
while (condition){
    Syso("Hello World");
}
```

```
int counter=0;          ← Initialization
while (counter<100){
    System.out.println("Hello world");
    counter++;
}
```

Counter → 0 1 2 3

Hello World
Hello World
Hello World

> Print Number 1 to 10

1 2 3 4 5 6 7 8 9 10 → while

```
int counter=1;
while (counter<=10){
    System.out.println(counter);
    counter++;
}
```

output :-

1
2
3
4
5
6
7
8
9
10

Code

```
int a=1;
while(a<=10){
System.out.print(a+" ");
a++;
}
```

1000, etc.

Output:- 1 2 3 4 5 6 7 8 9 10 ———

Print Number from 1 to n

```
n=7
int a=1;
while(a<=n){
```

user input

```
System.out.print(a);
```

or

```
int n= range
Scanner sc= new Scanner(
int a=sc.nextInt();
int a=1;
while(~~counter~~ a<=a){
System.out.print(a+" ");
}
```

Code:

```
import java.util.*;
public class Java{
public static void(String[]arg){
Scanner sc= new Scanner(System.in);
int a= sc.nextInt();
int a=1;
while(a<=n){
System.out.print(a+" ");
a++;}}
```

Sum of first n Natural Number

$n = 5$

$1 + 2 + 3 + 4 + 5 = 15$

```
int n = sc.nextInt();
int a = 0;
int a = 1;
while (a <= n){
a = a
```

or

```
int n = 5
int sum = 0;
int i = 1;
while(i <= n){
sum = sum + i;
}
```

for loop

```
for (initialization; Condition; updation){
    //do something
}
```

(counter / iteration; i<5; i++; (i = i+1), i = i+2, (i = i*2))

```
for (int i = 1; i <= 10; i++){
    sout("Hello World");
}
}
```

Print Square pattern:-

```
for (int i=1; i<=4; i++){
    sout("****");
}
```

* Print Reverse of a Number:-

n = 10899 → 99801

(%) — for lastdigit

```
Lastdigit = n%10;
n = n/10;
```

10899

numbr = 132 / 10

remainder → lastdigit

Remove lastdigit = num/10

Ex

```
int n = 10899;
while (n>0){
    int lastdigit = n%10;
System.out.println(lastDigit + "");
    n = n/10;
}
System.out.println();
```

Do while Loop

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

```
int a = 1;
do {
    say("Hello World");
    a++;
} while (a <= 10);
```

- **Break Statement:** to exist the loop.

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        break;
    }
    say(i)
}
System.out.println("I am out of the loop");
```

Q/ ~~do~~ for entering number till users enter a multiple of 10.

do{ int n = sc.nextInt(); if (n%10==0){
    say(n);
}
while(True);

Continue Statement To skip an Iteration.

1 to 5
i==3 skip

```
for (int i=1; i<=5; i++){
    if (i== 3){
        continue;
    say(i);
}}
```

Check Number prime or Not

prime No.. 2, 3, 5, 7, 11

10. Pattern Part 1

Nested loop :

```
if (a>b){
    if (a>c){

    }
}
```

Ex

```
for (int i=0 ; i<n ; i++){
    for (int j=0 ; j<n ; j++){

    }
}
```

```
*          → 1
* *        → 2
* * *      → 3
* * * *    → 4
* * * * *  5
```

① line 4 outer loop → 4 times

② Number of times inner loop 5 times

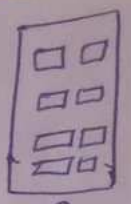
③ what print "*"

```
for (int a=1 ; a<=4 ; a++){        // Now line
    for (int j=1 ; j<=a ; j++){
        sysout("*");
    }
    sysout();                      // next line
}
```

a = 1 ... 4 , j = 1 ... a

Dry Run :

Function / Method



Remote

Blocks of Code:

{

}

Syntax:-

```
return Type name(){
    // body
    return statement;
}
```

function Name

input → output

input	output
a, b	sum
n	1 to n
n	n!

```
public class JavaBasic {
    public static void main(String Arg[]){
        // body
    }
}
```

class

Accessmodifier — Return type — Array

```
Soy("Hello World");
}
```

Making a function:-

```
public static void printHelloWorld(){
    System.out.println("Hello World");
// return;
}
public static void main(String Arg[]){
    printHelloWorld();
}
}
```

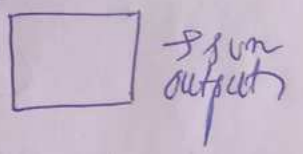
Calling of function

Method function को class के अन्दर लिखते है तो Method कहते है।

Syntax with Parameter

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

input



Code

```
import java.util.Scanner;
public class JavaBasic{
public static void CalculateSum(){
    Scanner sc = new Scanner(System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    int sum = a+b;
System.out.("sum is: " + sum);
}
public static main (String Arg[]){
    CalculateSum();
}
}
```

① ② ③

function.

```
public static void CalculateSum(int a, int b){
    int sum = a+b;
    return sum;
}
```

Parameter

Calling → ④ - main calculateSum(a,b) ← Arguments

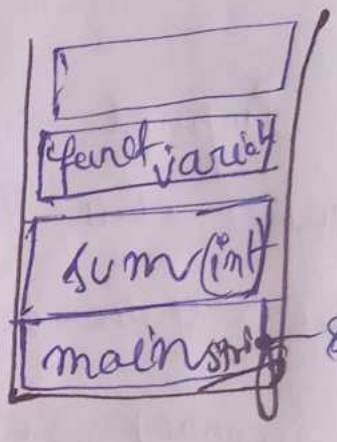
④ Parameter Vs Argument

Formal param (definition)

Actual parameter (call)

⑤ What happen in Memory

```
ps. Void Calculate (int a, int b)
    {
        int sum = a+b;
    }
p. s. Void main (String arg[])
    {
        sum();
    }
```



Call stack

⑥ Call by Value

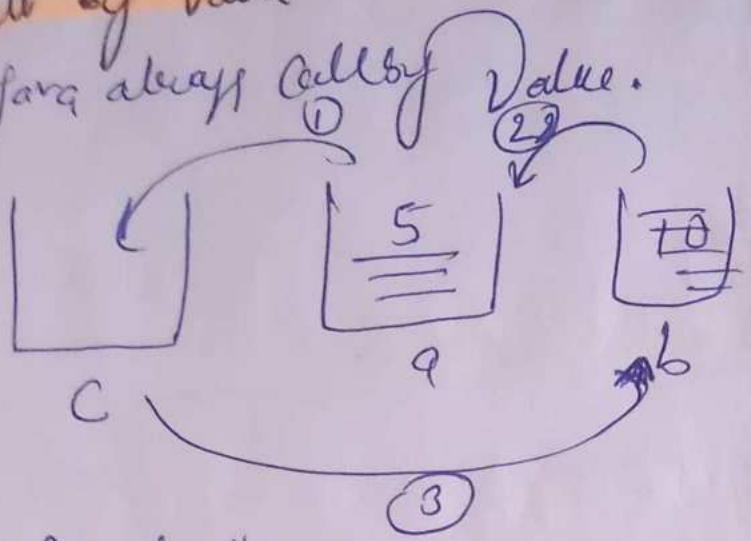
// swap → value exchange.

```
int a = 5;
int b = 10;
int c = a;
a = b;
b = c;
Say ("a = " + a);
Say ("b = " + b);
```

Call by Value

Call by Reference

→ Java always call by Value.



- By function

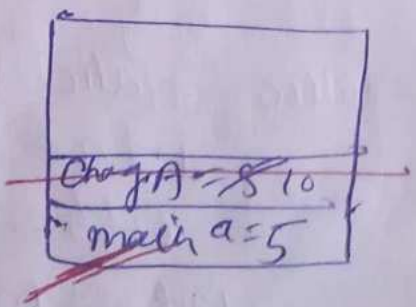
```
public static void swap (int a, int b){
  //swap -
  int c = a;
    a = b
    b = c;
  Say ("a =" +a);
  Say ("b =" +b);
}
public static void main (String arg[]){
  int a = 5;
  int b = 10;
  swap (a, b);
}
```

Calling by function

call by Value

Ex

```
ChangeA (int a){
  a = 10;
}
psvm ( ){
  int a = 5;
  ChangeA (a); Say (a); }
```



- Java always call by value.

```
Change A (int a){
    a=10;
}
psvm(){
int a = 5;
Change A(a);
Try(6);
}
}
```

Change value ~~error~~

Find the product of a & b

- a = 3 b = 5

```
int multiply (int a, int b){
  int product = axb;
  return product;
}
int p = multiply (3, 5);
print(p);
}
```

Code:

```
public static int multiply (int a, int b){
    int product = axb;
    return product;
}
public static void main( String args[]){
    int a = 3;
    int b = 5;
    int p = multiply (a,b);
Say ("a*b =" + p);
```

p = multiply(10, 25);
Soy("a*b=" + p);
}}

Factorial of a Number

n = 4 n! ⇒ n x (n-1) x (n-2) (n-3) ---- x 1

4! = 4x3x2x1 ⇒ 24 3! = 3x2x1 ⇒ 6

n! = (n-1)! x n = (n-2)! x (n-1) x n

n! = 1x2x3x4x5x ---- n

Code:

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

Explanation: n = 4

```
int f = 1
for(i = 1 ton){
    f = f * i;
}return f; ← factorial
}
```

f = 1
i = 1 f = 1*1 = 1
i = 2 f = 1*2 = 2
i = 3 f = 2*3 = 6
i = 4 f = 6*4 = 24

24 = n! = 4!

Binomial Coefficient

$n=10$ $r=2$

$nCr = \frac{n!}{r!(n-r)!}$ factorial(n).

factorial (n) (Reusable)

```
int binCoeff (int n, int r){
    n_fact = factorial (n); ——①
    r_fact = factorial (r); ——②
    nmr_fact = factorial(n-r); ——③

    return BC = a/(b*c);
```

Code:

```
public static int binCoeff(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;
}
public static void main( String Args[]){
    sop( binCoeff( 5, 2));
}
```

Method

- user defined
  - factorial
  - sum
  - product
- Inbuilt method
  - Math
    - pow
    - sqrt
    - max

SC . nextInt

Function Overloading

* Multiple function with the same name but different parameters.

Ex Calculator (Parameter)

multiply (int a, int b)
multiply2 (float a, float b)
multiply 3 (double a, double b)
multiply 3 (int a, int b, int c);

Function overloading using Parameter

→ only depend on parameter

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;
}

Coding

```
// funct to Calc sum of 2 numbs:
public static int sum(int a, int b){
    return a+b;
}
// funct to Cal sum of 3 numbs:
public static int sum(int a, int b, int c){
    return a+b+c;
}
public static void main(String args[]){
    Soy(sum(3,5));
    Soy(sum(5,2,1));
}
```

Function Overloading using Data type

→ f1: add 2 int Value.

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

f2: add 2 float

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

Code:
```
public static int sum(int a, int b){
    return a+b;
}
// function Call float:
public static float sum(float a, float b){
    return a+b;
}}
public static void main(String arg[]){
    Sy(sum(3,5));
    Sy(sum(3.2f, 4.0f));
}}
```

Check if a number is prime or not

Approach (1)

$n \rightarrow 1, n$

```
for (int i=2 to i=n-1)
    (n%i==0) Not prime
             True
False → Prime
```

Code:
```
public static boolean isPrime(int n){
    int isPrime = true;
for (int i=2; i<=n-1; i++){
    if (n%i==0)  // Completing device
```

```
        isPrime = false;
        break;
    }
}
return isPrime;
}
public static main(String arg[]){
    Soy(isPrime(12));
}}
```

Optimised Method: 6

1×6

2×3

3×2

6×1

8 → 1×8, 2×4, 4×2, 8×1

```
for (int i=2 to i<=√n)
public static boolean isPrime(int n){
for(int i=2; i<=Math.sqrt(n); i++){
    if(n%i==0){
        return false;
    }
}
return True.
public static void main(String arg[]){
    Soy(isPrime(6));
}}
```

Print all Primes in a Range

n = 10 → 2 → 10

(2, 3, 5, 7)

P.S Void PrimeInRange (int n){

for (int i = 2 to n){

if prime(i) == True / false.

}

Code.

}

public static Void primeInRange (int n){

for (int i = 2 ; i <= n ; i++){

if (isPrime(i)){

Say (i + " ");

}}

Say ( );

}

public static Void main (String Args[]){

primesInRange (20); // 2 to 20

Convert from Binary to Decimal n=101

Binary Number System:-
↳ Bit manipulation
Bit operator

Math → 0 to 9
[0,1,2,3,4,5,6,7,8,9] → Decimal Number System (10)

Binary → (0,1)

15 → bin
'a' → bin

$(0)_{10} = (0)_2$
$(1)_{10} = (01)_2$
$(2)_{10} = (10)_2$
$(3)_{10} = (11)_2$
$(4)_{10} = (100)_2$
$(5)_{10} = (101)_2$
$(6)_{10} = (110)_2$
$(7)_{10} = (111)_2$
$(8)_{10} = (1000)_2$

$n = 101$
$2^2\ 2^1\ 2^0$
$\Rightarrow 4+0+1 \Rightarrow (5)_{10}$

$(1000)_2 \rightarrow (\ )_{10}$
3 2 1 0

1×8 + 0×4 + 0×2 + 0×1

Scope Method

① Method Scope

function → Var

```
int sum (int a, int b){
    int s = a+b;
}
main (){
    →
}
```

← (यहां declare होगा / sirf यहीं होगा)

✓

✗ यह नहीं कर सकते यहां पर

Block Scope

```
{
    → Block of Code.
}
```

← Block ①

```
{
}
```

Block ②

Ex

```
for (int i=1; i<=5; i++){
}
```

Class Scope

String 1

String are type of object which can store character and element.

Char [ ] = { 'a' 'b' 'c' 'd' };

String str = "abcd"

Code:

```
public class Strings {
    public static Void main( String args[ ]){
Char Arr[ ] = { 'a' , 'b', 'c', 'd' };
String str = "abcd";
String str2 = new String("xyz");
// String are (Immutable) Not Change.
```

2. Input & output.

```
Scanner sc = new Scanner (System.In);
String name;
name = sc.next();   (next line)
System.out.println (name);
}}
```

3. String lenght

```
String full name = "Tonny Starks";
System.out.println(full name.lenght());
}}
```

output : 10

4. Concatenate

// Concatenation - Add first + last

```
String firstname = "Rohit";
String lastname = "yadav";
String name = firstname + " " + lastname;
Sop(fullName);
```

output Rohit yadav

5. String CharAt

```
String firstname = "Rohit";
String lastname = "yadav";
String fullname = firstname + " " + lastName;
Sop(fullName.charAt(0));
}
```

output: R

```
public class string{
public static void printletters(String str){
for(int i=0; i<str.length(); i++){
Sop(str.charAt(i) + " ");
}
Sop();
}
```

public static void main(String[] args){
String fullname = firstname + " " + lastname;
printletters(fullName);

6. Check the palindrome

Question:- "racecar" "noon" "madam"

(दोनों तरफ से same आना चाहिए) → Palindrome

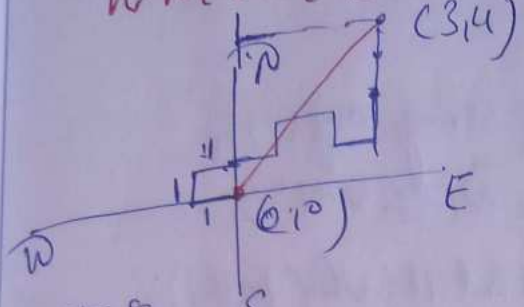
```
for (int i=0; i to length/2)
    str(i) == str.(n-i-1)
```

racecar
0 1 2 3 4 5 6
(n-1-1)

```
public static boolean isPalindrome (String str){
for (int i=0; i<str.length()/2; i++){
    int n = str.length;
if (str.charAt(i) != str.charAt(n-1-i)){
    return false;
}}
return true;
}
public static void main( String args[]){
    String str = "noon";
System.out.println( isPalindrome(str));
}}
```

Question: given a route containing 4 direction (E, W, N, S) find the shortest path to reach destination.

"WNEENESENNW"



displacement = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$= \sqrt{9+16}$

d = 5

x=0
y=0
N = y+1
S = y-1
W = x-1
E = x+1

```
public class Solution {
    public static int getShortestPath(String path) {
        int x = 0, y = 0;
        for (int i = 0; i < path.length(); i++) {
            char dir = path.charAt(i); // South.
            if (dir == 'S') {
                y--;
            } else if (dir == 'N') {
                y++;
            } // West
            else if (dir == 'W') {
                x--;
            } else {
                x++;
            }
        }
```

```
int x2 = x*x;
int y2 = y*y;
return Math.sqrt(x2+y2);
(float) // type casting
public static void main(String arg[]){
    String path = "WEENDESEAWW";
System.out.println(get shortestpath(path));
}
}
```

output 5.0

8: String function

```
String s1 = "Tommy";
String s2 = "Tony";
String s3 = new String("Tony");
if (s1 == s2){
system.out.println("String are equal");
} else { say("string are not equal");
}if (s1.equals(s3){
say("string are equal");
} else { say("string are not Equal");
}
```

9 Substring

"Hello World"

Substring

Substring

si ei

excluded

[3, 5] → "o w"

Index → Substring

```
public class substring {
    public static substring (String str, int si, int ei){
        for (int i=si; i<ei; i++){
            subStr += str.charAt(i);
        }
        return subStr;
    }
    public static void main (String arg[]){
        //sub string
        String str = "Hello World";
        Say (str.substring(str, 0, 5));
    }
}
```

output

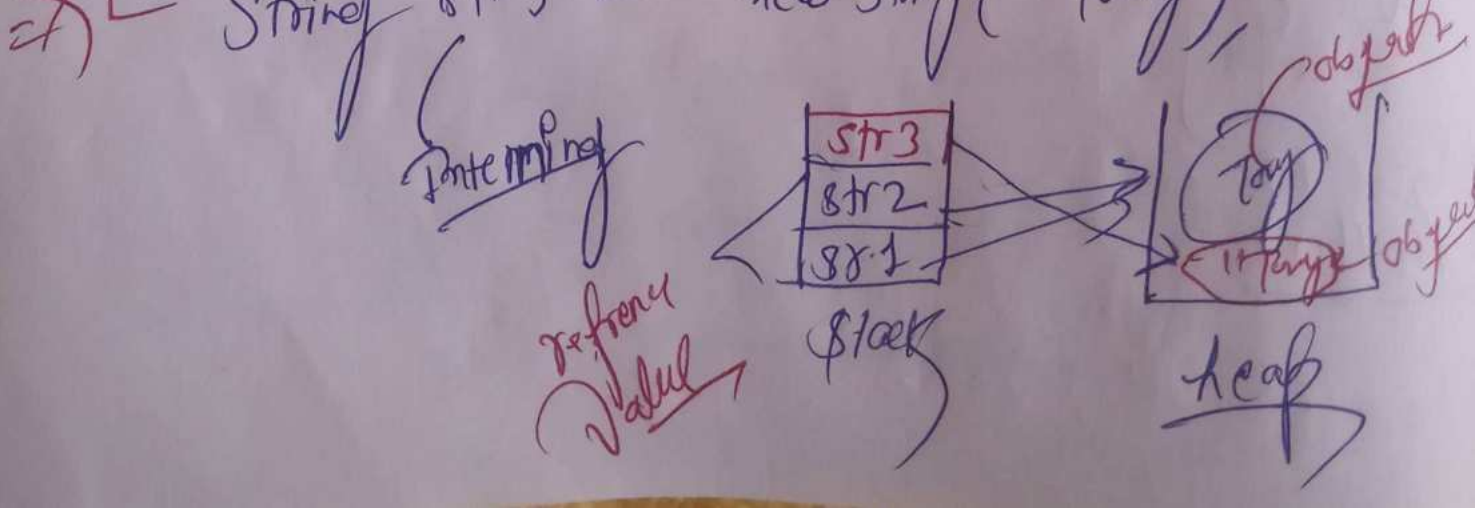
Hello

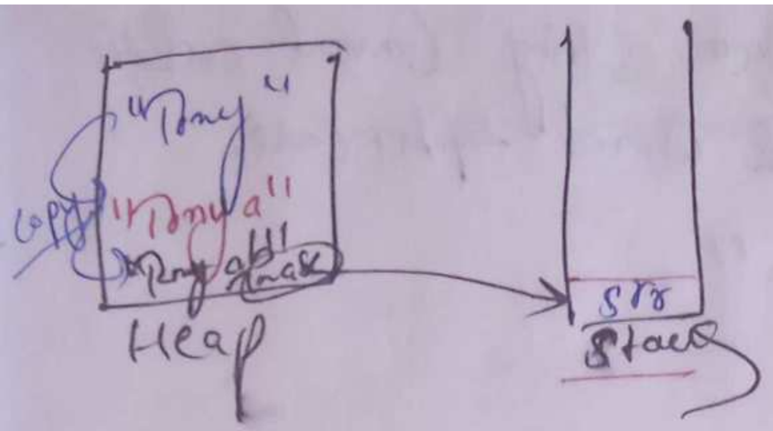
Q. For a given set of strings, print the longest string.

"Apple", "mango" "banana".

11. Why string are Immutable

```
String str1 = "Pony";
String str2 = "Pony";
String str3 = new String("Pony");
```





(Time Complexity)

T ∝ string lenght

(Copying)

```
String str = "Tony"
for (Char = 'a' to 'z'){
    str += ch
}
```

26 x n   Tony a

n x n   Tony ab

n^2 Tony abc

(time Complexity)

StringBuilder

```
String Builder sb = new StringBuilder("H.W");
Sb. to String()
```

Char ch = 'a'
Ch.to String() ✗

int a = 10;
a to String() ✗
only object change

Code:

```
public static void main(String arg[]){
    String builder sb = new StringBuilder("");
    for (Char ch = 'a'; ch <= 'z'; ch++){
        sb.append(ch);
    }
    Sop(sb.lenght());
}}
```

Question 4 For a given String Convert each first letter of each word upper Case.

"hi, I am shraddha"

Imp
① Say(Str. length());
② Say(Str. charAt(0));
③ Say(Str. toUpperCase());
④ Say(Str. toLowerCase());
⑤ Say(Str. substring(0,0));

Q5 String Compareesion:..

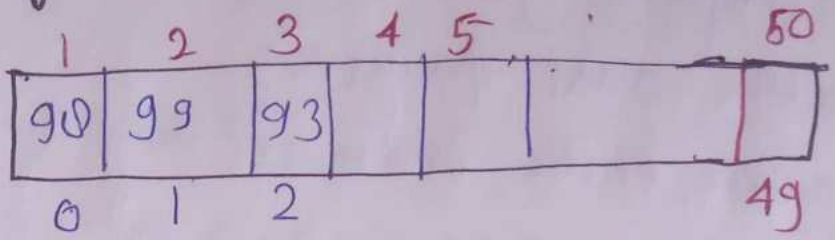
"aaabbCCcdd" ⇒ "a3b2C2d2"

"aaabbdd" → "a3b2d2"

"abC" → "abc"

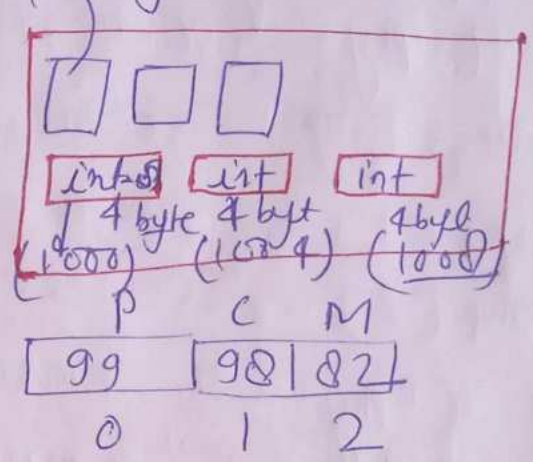
Arrays (DATA Structure)

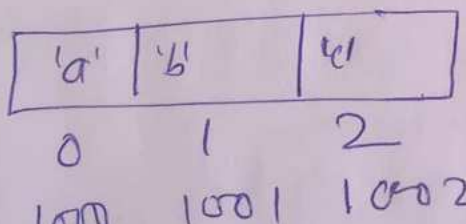
int Phy = 90
int Chem = 99
int math = 93



* "List of elements of the same type placed in a Contiguous memory location"

1, 2, 3, 4 → Int
a, b, c, d → Char
1.2, 3.4, 4.8 → float
"Apple" "Banana" → String





(+1)

Operation On Arrays

- Create
- Input
- Output
- Updation

(1) Creating An Array:-

dataType arrayName[] = new dataType[size];

Ex int mark[] = new int[50];

int number[] = {1, 2, 3}; (3)

int moreNumber[] = {4, 5, 6}; (3)

String fruits[] = {"Apple", "Mango", "Orange"};

size (3)

3. Input, output - Op

```
import java.util.*;
public class Arrays cc {
public static void main(String Arg[]){
    int marks[] = new int[100];
Scanner sc = new Scanner(System.in);
    // int Phy;
    // Phy = sc.nextInt();
marks[0] = sc.nextInt(); //phy
marks[1] = sc.nextInt(); // Chem
marks[2] = sc.nextInt(); // math
Soy ("Phy : " +marks[0]);
Soy ("Chem : " +marks[2]);
Soy ("math : " + marks[1]);
    // Updation
    marks[2] = 100;
Soy ("math : " + marks[2]);
}
}
```

* Accessing Arr

Ex

```
import java.util.Arrays;
    int[] arr = {2,3,4,5};
Soy (Arrays.toString(arr));
}
```

int [] arr = new int[4]; array lenght = 4

arr[0] = '1';
arr[1] = '2';
arr[2] = '3';
arr[3] = '4';

Soy (Arrays.toString(arr));

}}

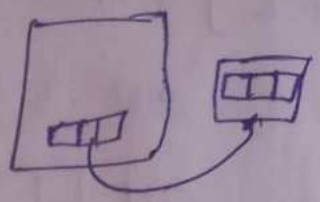
Q String str = "Hello word";
for (int i=0; i < str.lenght(); i++){
if (str.charAt(i) == 'o'){
Sout(i).

output: (5, 7)

import.util.Array;

Q int []arr = new int[10];
for (int i=0; i < arr.lenght; i++){
arr[i] = obj.nextInt();
}
S.ub (Arrays.toString(arr));

4 Passing arrays as Arguments



Pass by value

by refference

main()

```
import java.util.*;
public class Arrays {
    public static void update(int marks[]){
        for(int i=0; i<marks.length; i++){
            marks[i] = marks[i] + 1;
        }
    }
    public static void main(String args[]){
        int marks[] = {97, 98, 99};
        update(marks);
        // print our marks:
        for(int i=0; i<marks.length; i++){
            System.out.print(marks[i] + " ");
        }
        System.out.println();
    }
}
```

5. Space And Time Complexity

Scenario:- Imagine you give 100 Rupees to one of your friend. Having them all together, you would like your pen back.

Space Complexity: Space Complexities is represented as a function portrays the amount of space necessary for an algorithm to run until complete.

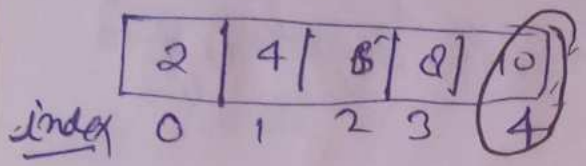
⇒ In our scenario we are looking at you can think of space complexities as the number of room you need to fig out who has pen. In Computer means how much memory does the process and data structure in our code base / function take up to achieve their goal.

Time Complexity

Time Complexity is represented as a function that portrays the Amount of time necessary for an algorithm to run until Complete.

Linear Search

- Find the index of element in a given array:-



Key=10 return → 4

Code:-

```
int number[];
import java.util.*;
public class Arrayfee{
public static int linearSearch(int number[], int key){
    for(int i=0; i<numbers.length; i++){
        return i;
    }
    return -1;
}
public static void main(String args[]){
    int numbers[] = {2,4,6,8,10,12,14,16};
    int key = 10;
    int index = linearSearch(number, key);
    if(index == -1){
        Say("Not found");
    }else{
    Say("Key is at index: "+ index);
    }
```

Time Complexity → loop पर depend करता है, time -

for (i = 0 to n). ← length of Array

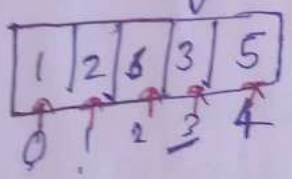
O(n) n operation.

(big O)

TC → loop (n)

Largest Number

Find the largest number in given Array:-



largest = ~~-∞, 1, 2, 6~~

-∞ = integer. MIN_VALUE

+∞ = integer. MAX_VALUE

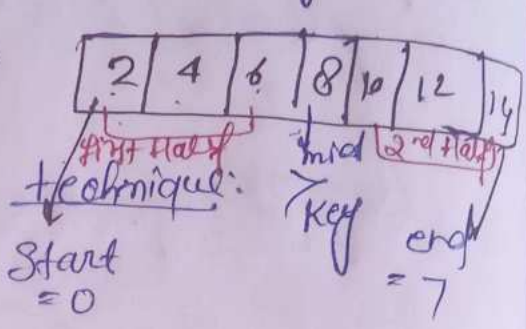
Code:

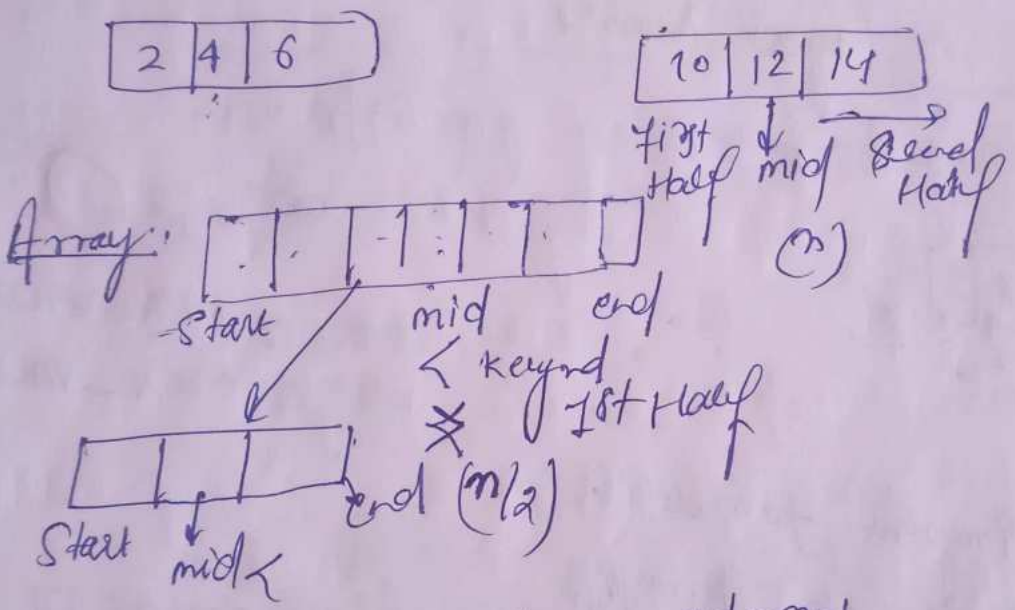
```
import java.util.*;
public class Arrays{
public static int getLargest(int numbers[]){
int largest = Integer.MIN_VALUE; // -infinity
for(int i=0; i<numbers.length; i++){
if(largest<numbers[i]){
largest = numbers[i];
}}
return largest;
}
public static void main(String args[]){
int number[] = {1,2,6,3,5};
System.out.println("Largest value is :" + getLargest(numbers));
}}
```

Binary Search

Prerequisitic - Sorted arrays

Dictonary → Shortcol





Psedo Code:-

start=0, end=n-1
while (start <= end)
find mid
Compare mid & key
mid == key Found
mid > key Left
mid < key Right

Binary Search Code

Code:

```
import java.util.*;
public class Array{
public static int binarySearch (int number[], int key){
  int start = 0, end = number.length-1;

while (start <= end){
int mid = (start + end)/2;
  // Comparision;
if (number[mid] == key){
  return mid;
} if (number[mid] < key){
    start = mid+1;
} else {
    end = mid-1;
}}
  return -1;
}
public static void main (String Args[]){
  int numbers[] = {2, 4, 6, 8, 10, 12, 14};
    int key = 10;
  Soy("index for key is: " binarySearch (number));
}}
```

Time Complexity:

Iteration 1 → n ($n/2^0$)
Iteration 2 → $n/2$ ($n/2^1$) formula:
Iteration 3 → $n/4$ ($n/2^2$)
Iteration 4 → $n/8$
⋮
$n/2^k$

$(k+1)^{th}$ iteration

$n = 2^k$

$k = \log_2 n$

$TC \propto \log_2 n$

$O(\log n)$

$n > \log_2 n$

Linear Search — Binary Search

Linear Search:
Size = $n = 8$
$O(8)$

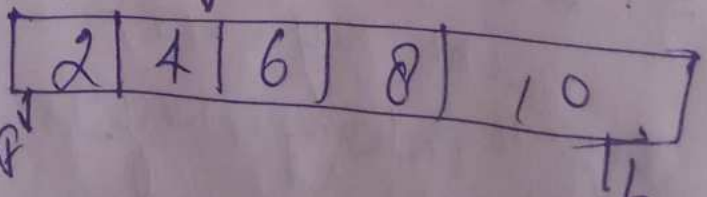
Binary Search:
8 → 4 → 2 → 1
$O(4)$

Reverse an Array

Space

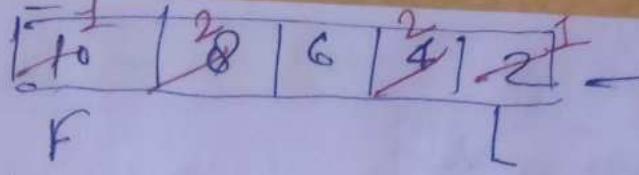
Time Complexity = $O(n)$

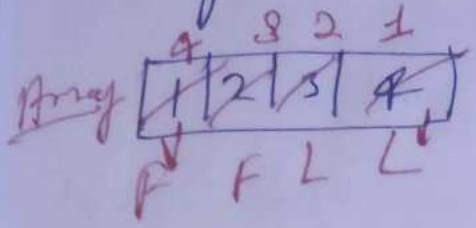
Space Complexity = $O(3)$



Swap

reversed Array





4 3 2 1 ← Reversed Array

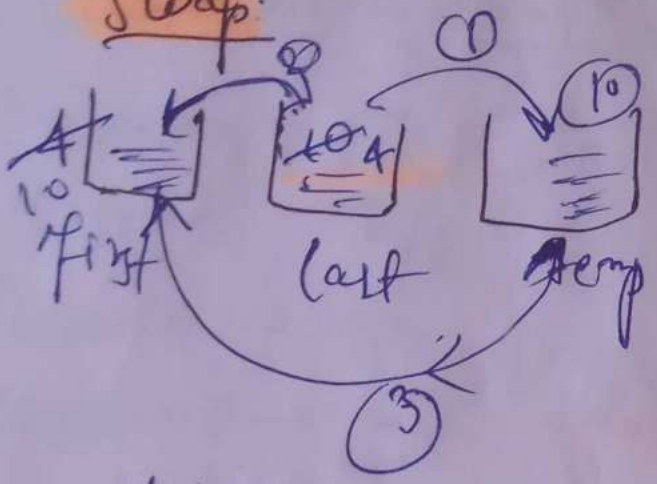
$O(2) = O(1)$

SC → $O(1)$
$O(n/2)$
$O(n)$

Code:

```
import java.util.*;
public class Arrays {
public static void reverse(int number[]){
    int first=0; last= numbers.length-1;
while (first < last){
 // swap
    int temp= number[last];
    number[last]= numbers[first];
    number[first]= temp;
    first++;
    last--;
}
```

Swap:



4 10

10 4

```
public static void main (String Arg[])
    int number[] = {2, 4, 6, 0, 10};
    reverse (number);
for( int i=0; i < number.length; i++){
    say( number[i] + " ");
}
    say( );
}
```

output: 10 0 6 2

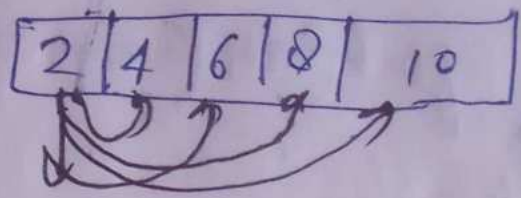
Pairs in An Array

(2,4) (2,6) (2,8) (2,10) → 4

(4,6) (4,8) (4,10) → 3

(6,8) (6,10) → 2

(8,10) → 1



Nested loop

"2" → (2,4) (2,6) (2,8) ; (2,10)

"4" → (4,6) , (4,10) (4,8)

for (int i = 0 to n) 2, 4, 6, 10

i = n-1

current

for (int j = i+1 to n) [2, 6, 8, 10]

[4, 8, 10]

[8, 10]

∃ pairs

import. java.util.*;

```
public class Arrayff {
public static void printpairs (int numbers[]){
    for (int i=0; i<numbers.length; i++){
    int curr = number[i];
    for (int j=i+1; j<numbers.length; j++){
        soy ("(" +(curr+", " +numbers[j]+")");
    }
    soy(i);
}
    public static void main(String Arg[]){
```

```
int numbers[] = {2, 4, 6, 8, 10};
printPairs(numbers);
}
}
```

n elements

$$Tp = \frac{n(n-1)}{2}$$ → formula

Total pair

Time Complexity = $O(n^2)$

nested loop

$n + (n-1) + (n-2) + (n-3) \propto O(n^2)$

12. Print Sub Array

A Continous part of Array :- [2 | 4 | 6 | 8 | 10]

[2 | 4 | 6]

Sub Array

[2]	[2 | 4]	[2 | 4 | 6]	2, 4, 6, 8	2, 4, 6, 8, 10
4	[4 | 6]	[4 | 6 | 8]	4, 6, 8, 10 — (4)	
6	6, 8	[6 | 8 | 10] — (3)		
8	[8, 10] — (2)			
10	→ 1 (1)			

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

Logic

start

for (int i=0 to n) → L1

for (int j=i+1 to n) → L2

for (start to end) → L3

≡ Sub Array

2	4	6	8	10

start = 2.

end = 4, 6, 8, 10

2 4	2 4 6	2 4 6 8	2 4 6 8 10

Code:

```
import java.util.*;
public class Arrays{
    public static void printSubarray(int numbers[]){
        int ti = 0;
        for(int i=0; i<numbers.length; i++){
            int start = i;
            for(int j=i; j<numbers.length; j++){
                int end = j;
                for(int k=start; k<=end; k++){
                    System.out.print(numbers[k]+" ");
                }
                System.out.println();
            }
        }
    }
}
```

```
}   ~~t++~~;
    soy(" ");
}}  soy("total subarray =" + tp);
public static void main (String args[]){
    int numbers [] = {2, 4, 6, 8, 10}
    print subarray (numbers);
}}
```

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

Q. Max SubArray Sum

1	-2	6	-1	3

(1) (1, -2) (1, -2, 6) (1, -2, 6, -1) (1, -2, 6, -1, 3)

1 -1 5 4 7

-2

6

-1

3

1	-3	2

Maximumsum = ~~-∞~~ ~~1~~

Currentsum

CS = (-3

⇒ -1

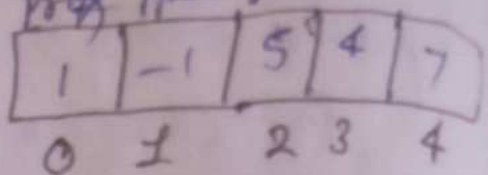
CS = 2

```
import java.util.*;
public static void printSumSubArray(int numbers){
    int currSum=0;
int maxSum= Integer.MIN_VALUE;

for(int i=0; i< numbers.length; i++){
    int start=i;
for(int j=0; j< numbers.length; j++){
    int end=j;
    currSum= 0;
for(int k=start; k<=end; k++){
    currSum+= numbers[k];
}
if(maxSum< currSum){
    maxSum= currSum;
}
}
}
Sop("maxSum="+maxSum);
}
public static void main(String[]){
    int numbers[]={2,4,6,8,10};
    maxSubarraySum(numbers);
}
}
```

Output:- maxSum=30

II. Max SubArray Sum

Array:



III. Max SubArray Sum

Trapping Rainwater

Given a non-negative integers representing an elevation map where the width of each bar is 1. Compute how much water it can trap after rain

height = [4, 2, 0, 6, 3, 2

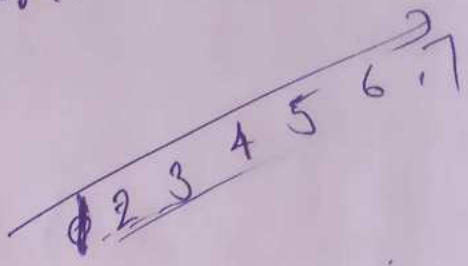
13.1Q Best time to buy stock

Q You are given an Array prices where prices[i] is the price of a given stock on the i^{th} day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0;

Basic of Sorting

Arrange in an order:

- Buble Sort
- Selection Sort
- Inyection Sort
- Counting Sort (Advance)



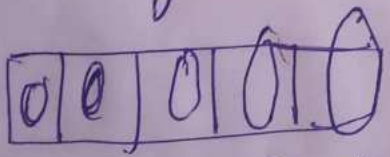
Dubble Short

{5, 4, 1, 3, 2} unsorted Array

{1, 2, 3, 4, 5} increasing order

{5, 4, 3, 2, 1} decreasing order.

Idea: large element come to the end of array by swapping with adjacent element.



→ increasing order (0 to n-2)

Bubble sort n = 5

5, 4, 1, 3, 2

4 5, 1 3 2

4, 1 5 3 2

4 1 3 5 2

4, 1, 3 2 5

~~4 1 3 5 2~~

1st 4 1 3 2 5

1 4 3 2 5

1 3 4 2 5

1 3 2 4 5

2nd 1 2 3 4 5

3rd 1 2 3 4 5

1 2 3 4 5

turn = n-1

⇒ 5-1 ⇒ 4th

for (turn = 0 to n-2)

for (int j = 0 to n-2-turns)

turn 0

j = 0 to (5-2-0) ⇒ 3

Code:

```
import java.util.*;
public class BasicSorting {
    public static void bubble(int arr[]){
        for (int turn=0; turn<arr.length-1; turn++){
            for (int j=0; j<=arr.length-1-turn; j++){
                if (arr[j] > arr[j+1]){
                    // swap
                    int temp = arr[j];
                    arr[j+1] = temp;
                }
            }
        }
    }
    public static void printArr(int arr[]){
```

Bubble Sort

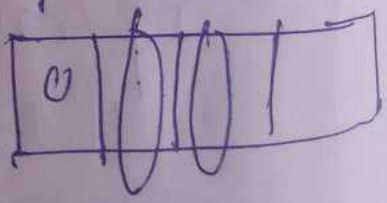
```
for (int i=0; i<arr.length; i++){
    Soy.(arr[i]+ " ");
}
}
Soy()
public static void main(String args[]){
int arr[]={5,4,1,3,2};
bubbleSort(arr);
printArr(arr);
}}
```

o/p 1,2,3,4,5

Time Complexity $O(n^2)$ Best

Selection Short

idea: picks the smallest (from unsorted) put it at the begining.



5, 4, (1), 3, 2

1 5 4 3 (2)

1 2 5 4 (3)

1 2 3 5 (4)

for(int i = 0 to n-2) → Smallest = min

for(j = i+1 to n-1)

unsorted Arr → start

Code:

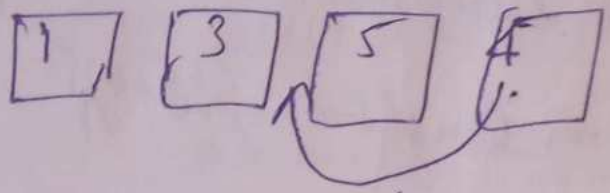
```
public static void selectionSort(int arr[]){
    for(int i=0; i<arr.length-1; i++){
        int minPos = i;
        for(int j=i+1; j<arr.length; j++){
            if(arr[minPos] > arr[j]){
                minPos = j;
            }
        }
        // swap
        int temp = arr[minPos];
        arr[minPos] = arr[i];
        arr[i] = temp;
    }
    public static void main(String args[]){
        int arr[] = {5,4,3,2,1};
        bubbleSort(arr);
        print(arr);
    }
}
```

Solution

O(p → 1,2,3,4,5

Insertion Sort

Idea: Picks an element (from unsorted part) & place in the right pos in Sorted part



insertion Sort. n=5

5 | 4, 1, 3, 2

Sorted | UnSorted

4 5 | 1 3 2

1 4 5 | 3 2

1 3 4 5 | 2

1 2 3 4 5

Code

```
import java.util.*;
public static void insertionSort(int Arr[]){
    for(int i=1; i<arr.length; i++){
        int curr = arr[i];
        int prev = i-1;
        while(prev>=0 && arr[prev]>arr[curr]){
            arr[prev+1] = arr[prev];
            prev--;
        }
```

// insertion

```
arr[prev+1] = arr[curr];
}
}
public static void main(String args[]){
    int arr[] = {5, 4, 1, 3, 2};
    insertionSort(arr);
    printArr(arr);
```

Inbuilt Sort

```
import java.util.Arrays;
Arrays.sort(arr)
O(n log n)
Arrays.sort(arr, sf, el).

import java.util.*;
public static void main(String args[]){
    int arr[] = {5, 4, 1, 3, 2};
    // insertion sort
    Arrays.sort(arr);
    printArr(arr);
}
}
```

1P

```
import java.util.*;
public static void main(String Args[]){
    int arr[] = {5, 4, 1, 3, 2};
    // insertion Sort;
    Arrays.sort(arr, 0, 3);
    printArr(arr);
}
```

O/P : [3 4, 5, 2, 1]

Counting Sort (1, 4, 1, 3, 2, 4, 3, 7)

- Marks: Min → max (+ve) no.

$n \log n$ ≈ n

min → 1
max → 7
range → 7

2	1	2	2	0	0	1
0	1	2	3	4	5	6	7

Count → frequency

Ascending order:-

1	1	2	3	3	4	4	7

① for(int i=0 to n) freq.

② for(int i=0 to max no.(range)

T ∝ O(n + range)

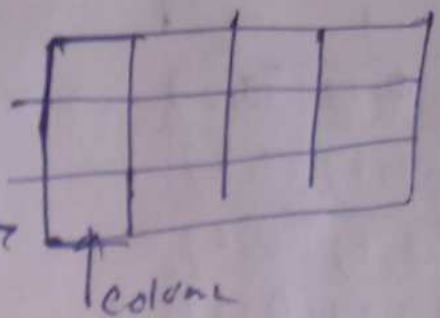
Code:

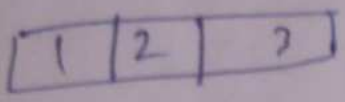
```
public static Counting Sort (int arr[]){
    int largest = Integer.MIN_VALUE;
    for (int i=0; i<arr.length; i++){
        largest = Math.max(largest+1, arr[i]);
    }
    int count[] = new int [largest+1];
    for (int i=0; i<arr.length; i++){
        count[arr[i]]++;
    }
```

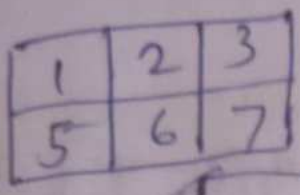
```
// sorting:
    int j = 0;
for (int i = 0; i < count.length; i++) {
    while (count[i] > 0) {
        arr[j] = i;
        j++;
    count[i]--;
}
}
public static void main(String args[]){
    Integer arr[] = {5, 4, 1, 3, 2};
    CountingSort(arr);
    printArr(arr);
}
```

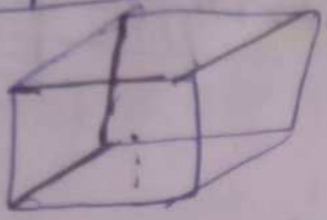
2D Array

Matrix (Rows And Columns)



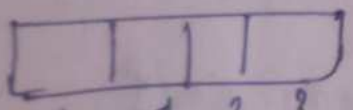
1D 

2D 

3D 

n-D Array

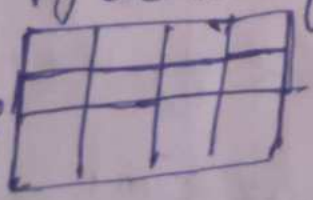
Real life Example

S1 

→ marks

S10 → 2D

Pay check mark eg → matrix



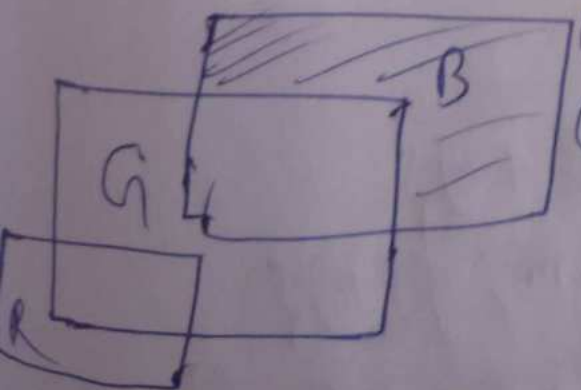
RGB matrix

Red Green Blu

(255 255 255) white

(0 0 0) Black

(0 0 225) Blue



pixel

Representation of 2D Array

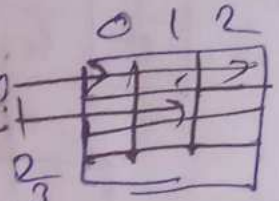
	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)
3	(3,0)	(3,1)	(3,2)

(n x m)

rows = 4

Column = 3

Create 2D Array



Code:

```
import java.util.*;
public class Matrices {
public static void main(String[] args){
    int matrix[][] = new int[3][3];
      int n=3 , m=3;
Scanner sc = new Scanner(System.in);
for (int i=0 ; i<n; i++){
    for (int j=0; j<m; j++){
        matrix[i][j]= sc.nextInt();
}}
// output
for (int i=0; i<n; i++){
    for( int j=0; j<m; j++){
        sout( matrix[i][j] + " ")
}}
```

```
Scy();
}
}
```

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

- you can also search in Matrix.

```
public class matrices{
  public static boolean search(int matrix[][], int key){
    for(int i=0; i<matrix.length; i++){
      for(int j=0; j<matrix[i].length; j++){
  if (matrix[i][j]==key){
   Scy("found at cell("+i+","+j);
      return true;
    }
  }
    Scy("key not found");
    return false;
  }
  public static void main(String args[]){
    int matrix[][]=new int[3][3];
```

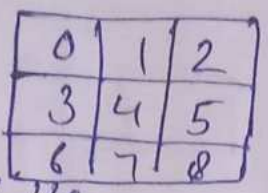
```
int n = matrix.length, m = matrix[0].length;
Scanner sc = new Scanner(System.in);
for (int i=0; i<n; i++){
  for (int j=0; j<m; j++){
    matrix[i][j] = sc.nextInt();
}}
//output
for (int i=0; i<n; i++){
  for (int j=0; j<m; j++){
  Sop(matrix[i][j] + " ");
}
  sop();
}
  search(matrix, 5);   // key
}}
```

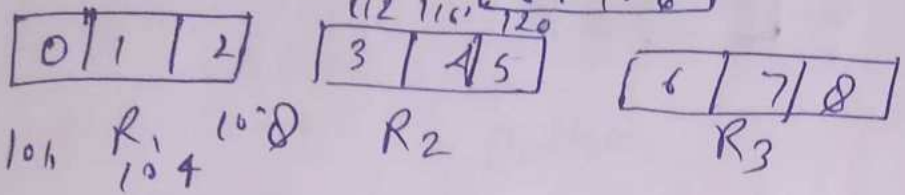
output

```
1 2 3
4 5 6
7 8 9
found at cell (1,1)
```

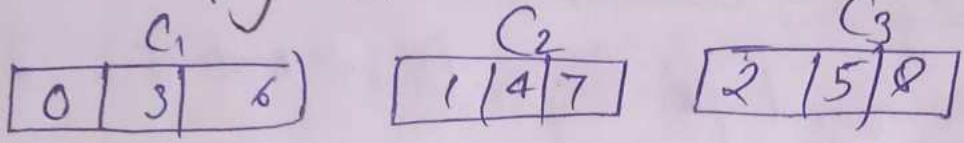
2D Array in Memory

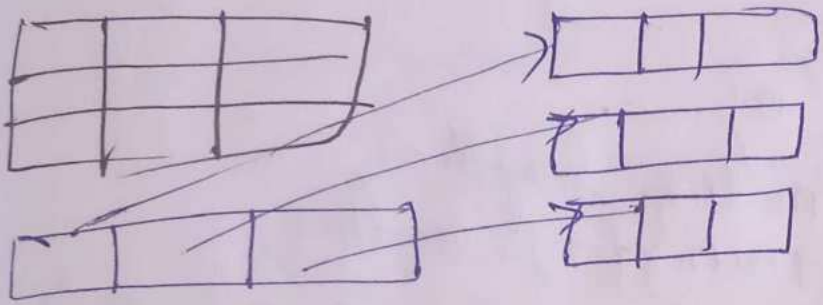
→ row major





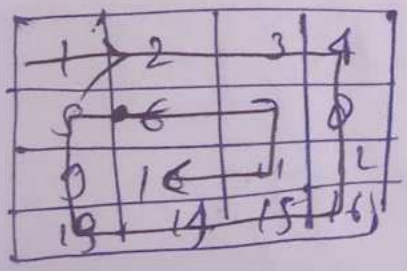
→ Column Major





Spiral Matrix

(google, oracle, Amazon)



output: 1, 2, 3, 4, 8 12 16 15 14 13 9 5 6, 7 11 10

Approach: Border → 2 cols, 2 rows (loop)

- Starting row = 0
  ending row = 3 (n-1)
- Start Col. = 0
  Ending Col. = 3 (n-1)

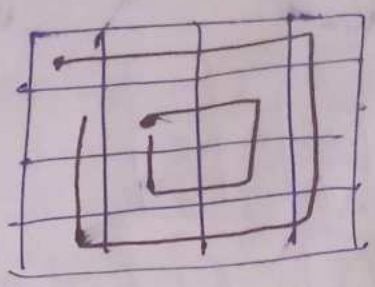
2nd

1 → ++
2 → --
1 → ++
2 → --

SR = ++
ER = --
SC = ++
EC = --

while (=)

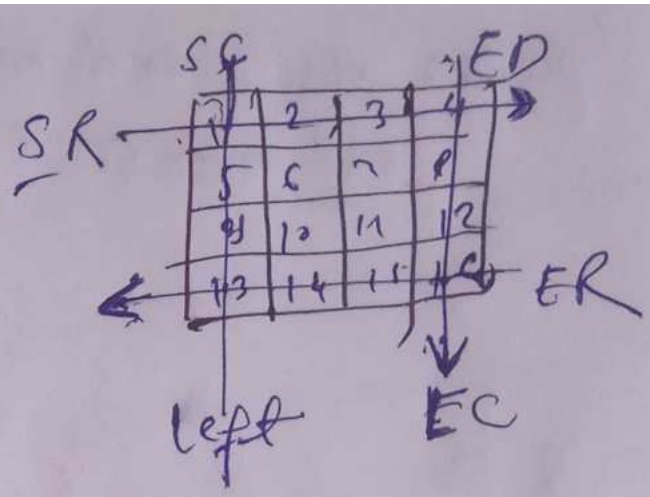
{ ① top
② Right
③ bottom
④ left.



Spiral matrix Code

```
public static void printSpiral(int matrix[][]){
    int startRow = 0;
    int startCol = 0;
    int endRow = matrix.length-1;
    int endCol = matrix[0].length-1;
    while( startRow <= endRow && startCol <= endCol){
        // top
        for(int j = startCol; j <= endCol; j++){
            System.out.print(matrix[startRow][j]+" ");
        }
        // right
        for(int i = startingRow+1; i <= endRow; i++){
            System.out.print(matrix[i][endCol]+" ");
        }
        // bottom
```

SR → Top
SC → EC
SR+1 → ER → Right
EC fixed
EC-1 → SC → Bottom
ER ~~fixed~~



Left SC (ER-1 → SR+1)

// bottom

```
for (int j = endCol-1; j >= startCol; j--){
    Sop(matrix[endRow][j]+" ");
}
```

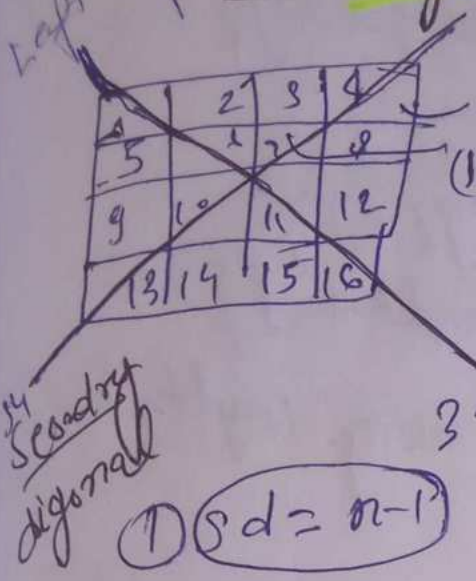
// left

```
for (int i = endRow-1; i >= startRow+1; i--){
    Sop(matrix[i][startCol]+" ");
}
startCol++;
startRow++;
endRow--;
endRow--;
}
Sop();
```

```
}
public static void main (String args[]){
    int matrix[][] = {{1,2,3,4},
                      {5,6,7,8},
                      {9,10,11,12},
                      {13,14,15,16}};
    printSpiral(matrix);
}
}
```

output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Digonal Sum (n x n)



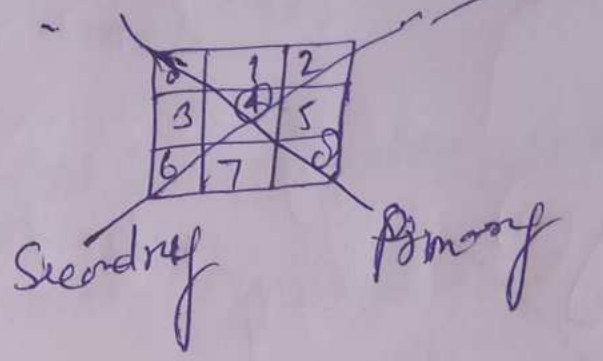
n = m

PD = 1+6+11+16 = 34

SD = 4+7+10+13 = 34

(1) Sd = n-1

P.D ℓ = j (row = col)



Code:

```
public static digonal sum( int matrix[][]){
        int sum=0;
{ for(int i=0; i<matrix[0].length; i++){
    for(int j=0; j<matrix[0].length; j++){
      if(i==j){
        sum+ = matrix[i][j];
      }else if(i+j== matrix.length-1){
          sum+= matrix[i][j];
      }
    }
  }
  return sum;
```

O(n²)

```
}
for(int i=0; i<matrix.length;i++){
//pd
sum+= matrix[i][i];
// Sd if(i!= matrix.length-1-i)
sum+= matrix[i][matrix.length-i
}
return sum;
}
```

Single loop

O(n)

Search in Sorted Matrix

Search for a key in row wise & col. wise sorted matrix.

10	20	30	40
15	25	35	45
27	29	37	48
32	33	39	50

key = 33

Approach

Key = 39

$n^2 = 16$ elements

① Brut force. (row wise, col wise)

$O(n^2)$

② Row wise

Key = 33

$O(n \log n)$

(n log n) (Binary Search)

Staircase Search

(0, m-1)

- key < cell value → LEFT
- key > cell value → Right

(4) → (0, m-1)

cell
- key < cell Value → Left
- key > cell Value → Bottom

(32) (n-1, 0)
- key < cell Value → Up
- key > cell Value → Right

(3) Staircase Search

Code:

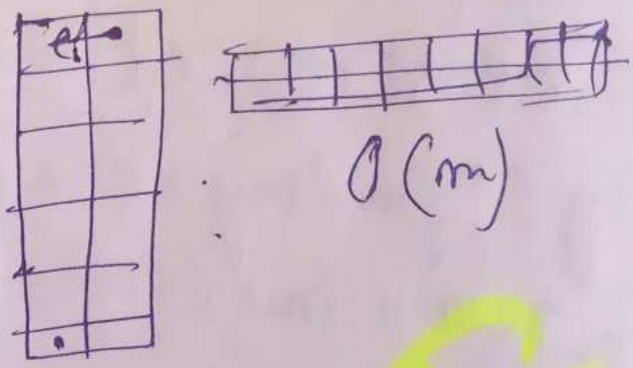
```
public static boolean staircaseSearch(int matrix[][], int key){
    int row = 0; col = matrix[0].length-1;
    while (row < matrix.length && col >= 0){
if (matrix[row][col] == key){
    Sop("found key at(" + row + ", " + col + ")");
    return True;
} else if (key < matrix[row][col]){
        col--;
    } else { row++;
    }}
    Sop ("key not found!");
    return false;
}
```

```
public static void main(String arg[]){
int matrix[][]={{10,20,30,40},
                {15,25,35,45},
                {27,29,37,48},
                {32,33,39,50}};
int key = 33;
staircaseSearch(matrix,key);
}}
```

Time Complexity :-

$O(n+m)$

$O(n)$ $n >> m$



$O(m)$

Complete Java D

Eco Friendly

N. Nature™

YOUR CHILD IS OUR PRIORITY

BIG BEN
LONDON

#NOTEBOOKS

A4

student exercise

Notebook

Recycle

easy to write

INDEX

Name: Rohit Kumar Yadav

Class: _____ Subject: D Roll No. : _____

Serial No.	Date	Title	Page No.	Teacher's Sign/Remarks
		Recursion		
		Divide and Conquer		
		Backtracking		
		Array list		
Unit 2		* Linked list		
Unit-3		* Stack		
		* Queue		
		Greedy		
Unit-4		* Binary Tree		
		Binary Search tree (BST)		
		Heap		
		Hash		
		Trie		
Unit-5		* Graphs		

Serial No.	Date	Title	Page No.	Teacher's Sign/Remarks

Dynamic Programing

* DP Part 1
* DP Part 2
* DP Part 3
* DP Part 4
* DP. Part 5
* DP. Part 6
* Segment Tree

Rule:

Total day = 100 day = 3 month 10 day.

Chapter = 50

Day 1 = Lecture

Day 2 = Practice Break.

Sep ∋ 25

oct ∋ 30

Nov. ∋ 15

70

Sep

6 → Recursion Lectur
7 → Practice
8 → Divide and Conquror
9 → Practice
10 → Back tracking
11 → Practice
12 → Arraylist
13 → Practice
14 → Linked list 1
15 → Practice
16 → Linked list 2.
17 → Practice
18 → Linked list 3
19 → Practice
20 → Stack
21 → Practice
22 → Que
23 → Practice
24 → Greedy
25 → Practice
26 → Binary tree 1
27 → Practice
28 → Binary tree 2
29 → Practice
30 → K th level

oct

1 → BST
2 → Practic
3 → BST - 2
4 → Practic
5 → Heap
6 → Practic
7 → Hashing
8 → Priority
9 → Trees
10 → Practic
11 → Graph
12 → Practic
13 → Graph - 2
14 → Practice
15 → BFS
16 → Practic
17 → Dijkstra
18 → Practic
19 → Flood fill algorithm
20 → Graph
21 → Practic
22 → DP Part 1
23 → Practice
24 → DP 2
25 → Practic
26 → DP 3
27 → Practic
28 → DP 4
29 → Practice
30 → DP 5
31 → Practic

Nov.

1. DP 6
2. Practice
3. segment tree
4. Practice

Complete
Lect Code
gfg
HackerRank

Recursion

➢ Iteration → Concept
→ function

Tree, graph, DP } Mainly used

→ What is recursion:-

using Math: $f(x) = x^2$

$f(f(x)) = (x^2)^2$

$x = 2$

$f(x) = 4 = (2)^2 = 16$

Factorial =

$n = n \times (n-1) \times (n-2) \ldots$

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

$4! = 4 \times 3 \times 2 \times 1$

$0! = 1$

$f(5) = 5 \times f(4) = 6 \rightarrow 6$

$4 \times f(3) \rightarrow 2$

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

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

$1 \times f(0) \rightarrow 1$

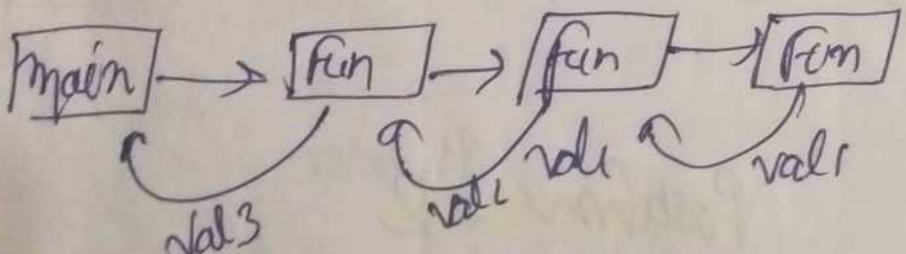
$f(n) = n \times f(n-1)$

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

Base Case. } $1 \times f(0)$

Recursion: Recursion is a method of solving a computational problem where the solution depend on solution to smaller instance of the same problem.

Base Case: where recursion stop (end).



① Base Case
② Recursion (मतलब) → $n \times f(n-1)$
③ Call inner function

PMI
Principal of mathematical induction

Problem 1

Print number from n to 1 (Decreasing order)

n = 10

10 9 8 7 6 5 4 3 2 1

```
for (int i=10; i>=1; i--)
    print (i);
```

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

$9 + f(8)$

$8 + f(7)$

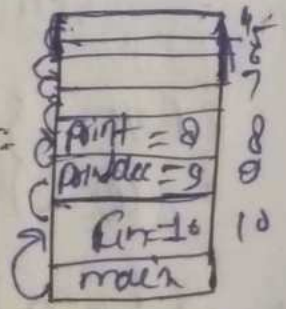
$7 + f(6)$

➤

```
funct (int n)
    print (n);
    fun (n-1);
```

Call stack:

Decreasing order:



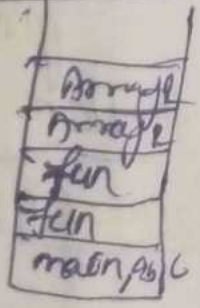
Code:

```
public static void printDec(
    if (n==1){
    Say(n)
    return;
    }
    Say (n);
    printdec(n-1);
    }
```

➤

Stack overflow

```
front = front++;
```



① Parameter

② too many call } stack overflow.

Base call ✗

Problem = 2

Print number from n to 1. (Increasing order)

n = 5

1 2 3 4 5

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

$f(n-2) + (n-1)$

```
fun(int n)
  fun(n-1)
  print(n).
```

n == 1 → print return

```
public static void printInc(int n)
if (n==1){
say(1);
return;
}
printInc(n-1);
say(n);
}
```

0! = 1

Problem 3

Print factorial of a number n:

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

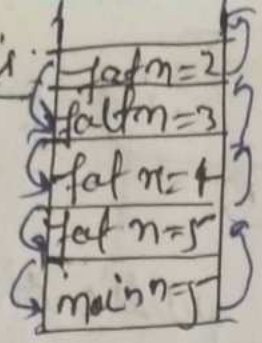
$(n-1) \times f(n-2)$

```
fn-1 = fact(n-1);
fN = n x fN-1;
if (n==0)
    return n1;
```

Code:

```
public static int fact(int n){
    if (n==0){
        return 1;
    }
    int fnm1 = fact(n-1);
    int fn = n * fact(n-1);
    return fn;
}
p.s.v.m(){
    int n=5; say(fact(0));
```

Stack Analysis:



$n == 0 \Rightarrow$ return 1

$f_{n-1} \rightarrow 1$

$f(n) = n \times f_{n-1}$

return f_n.

Problem 4

Print sum of N natural no. $n=5$

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

$f(5) = 5 + f(4)$ 15

$4 + f(3)$ 6

$3 + f(2)$ 3

$2 + f(1)$ 1

```
sum(int n){
    if(n==1)
        return 1;
    S_{N-1} = sum(n-1)
    S_N = n + S_{N-1}
    return S_N
}
```

Problem 5

Print Nth Fibonacci Series:-

0 1 1 2 3 5 8 13 21 ——

0th 1st 2nd 3rd

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

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

$fib_5 = fib_4 + fib_3$

Base case: $fib_3 + fib_2$

$fib_2 + fib_1$

$fib_5 = fib_4 + fib_3$

$fib_3 + fib_2$

① $fib_2 + fib_1$ $fib(1) + fib(0)$
(1) 0

fib_1 $fib(0)$
(1) 0

```
fib(int n){
if(n==0 → 0
   n==1 → 1 } Base case
fib_{n-1} = fib(n-1)
fib_{n-2} = fib(n-2)
fib_n = fib_{n-1} + fib_{n-2}
return fib_n;
}
```

Problem 6 Check array are sorted or not

| 1 | 2 | 3 | 4 | 5 |

i, i+1

| 1 | 2 | 3 | 4 | 5 |

True 2<3 True
1<2

Loop i=0 to n-1
a[i] <= a[i+1]
} sorted(n) = a[0] < a[1]
+ sorted(n-1)

if Sorted(int arr[], int i)
arr[i] <= arr[i+1]
if Sorted (arr, i+1);

→ Base case i = arr.length

Problem 7 → WAF to find occurence of an element in an array.

Key = 5
indx = 4 ✓
indx = 7

| 0 | 3 | 6 | 9 | 5 | 10 | 2 | 5 | 3 |

indx = 0 i=1 i=2 i=3 i=4 i=9-1

return (4). → Not found

firstoccur(arr, i, e)
a[i] == key
return i;
return fo(arr, i+1);

Code public static int firstocc(int arr[], int key)

if(arr[i] = key){
return i;
}
return firstocc(arr, key, i+1);
}

public static void main(String[] args)
int arr[] = {0, 3, 1, 2, 4, 1, 5}
Sop(firstocc(arr, 5, 0));
}

Last Occurence

WAF to find last occurance of an element in a array

key = 5

0	3	6	9	5	10	2	5	3
				i=4			e=7	

Compare with self
Looks forward
↓
Look forward
Compare with self

int isfound = lastocc(arr, key, i+1)
if(isfound != -1){
return isfound;
}
if(arr[i] = key){
return i;
}

Problem 9

Print x^n. $x=2$, $n=10$

$x^n = x \times x^{n-1}$

$x \times x^{n-2}$

$x \times x^{n-3}$

$2^{10} \rightarrow 1024$

pow(x, n) = x * pow(x, n-1)

n = 1

$x^1 = x$

pow(x, N)

return pow(x, n-1)

Code

```
public static int power(int x, int n){
    if (n == 1){
        return 1;
    }
    int xnm1 = power(x, n-1);
    int xn = x * xnm1;
    return xn;
    return x * power(x, n-1);
}
public static void main(String[] args){
    int[] arr = {5,5,5,5};
    Say(lastEven(arr, 5, 0));
    Say(power(2,10));
}
```

Problem 10

Print x^N in $O(\log n)$

$x^n (2^{10}) = x^{n/2} \times x^{n/2}$

n = even $(2^5) \times (2^5) = 2^{5+5} = 2^{10}$

$2^{10} \rightarrow 2^5 \times 2^5$

$2 \times 2^2 \times 2^2$

$2 \times 2 \quad 2 \times 2$

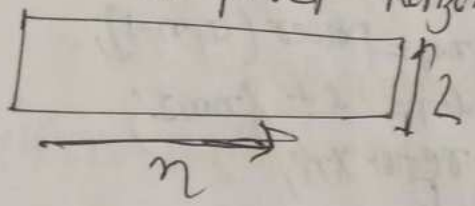
$O(\log n)$

$\frac{n}{2^e} \Rightarrow 1 \Rightarrow n = 2^l$

$\boxed{\log_2 n = l}$

Problem #1

Tiling problem: given a "2 x n" board and tiles of size (2x1) count the number of way to tile the given board using the 2 x 1 tiles. A tile can either be placed horizontally or vertically)



Code:

```
public static int tilingProblem(int n){ //2*n (floors)
    if(n==0 || n==1){
        return 1;
    }
    // vertical choice
    int verticalTile = tilingProblem(n-1);
    // horizontal choice
    int fnm2 = tilingProblem(n-2);
    int totalWays = fnm1 + fnm2;
    return totalWays;
}
```

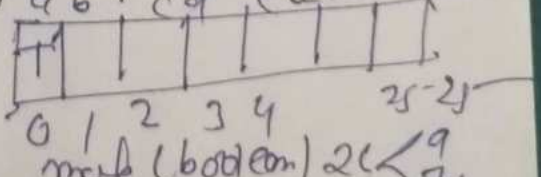
Problem No. 12

remove Duplicate in a string "appnacollege"

appnacollege

"apnacolleg"

"unique"



map (boolean) 26 < 9/2

newString (SB)

index string index

(1) base idx = a to n

(2) [illegible] Char → present in map → Yes → new str X

→ Not → new str ✓

(3) idx = idx + 1

Problem 13

Friend pairing problem

Q Given n friend, each can remain single or can paired up with some other friend. Each friend can be paired only once. Find out total way in which friend can remain single or can paired up.

$n=1$ (single) (a)

$n=2$ (a,b)

$n=3$ a, b, c — way = 4

(a b c)

(a,b), c, (b,c), a, (a,c), b,

① Base Case: $n=1$ way = 1; $n=2$ way = 2 } $n=1 \,||\, n=2$ way = n

② Recurrence.

Binary String Problem

Print all binary string of size N without consecutive one.

010101 ✓ n=6

01101001 ✗

n - size n=2

Binary string (01)

0 0

0 1

1 0

Base Case:

n=0 ""

n=1 "1", "0"

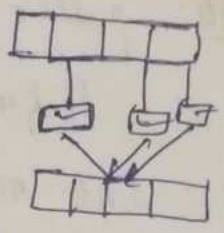
n=2 00, 01, 10, 11 ✗

① Base: n=0 empty

② Loop

Divide And Conquror

Convert into small problem



Merge Sort

Unsorted Array: | 6 | 3 | 9 | 5 | 2 | 8 |

Sorted Array: | 2 | 3 | 5 | 6 | 8 | 9 |

{ buble
selection
insertion
Counting

Approach:

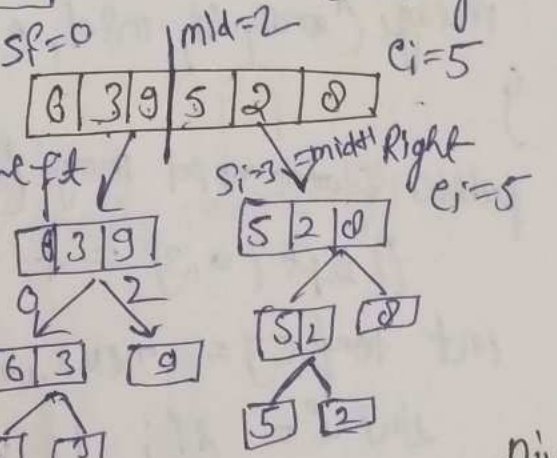
① To Divide mid

s_i = starting index

e_i = ending index.

$mid = \frac{(s_f + e_i)}{2}$

$s_i + (e_i - s_i)/2$

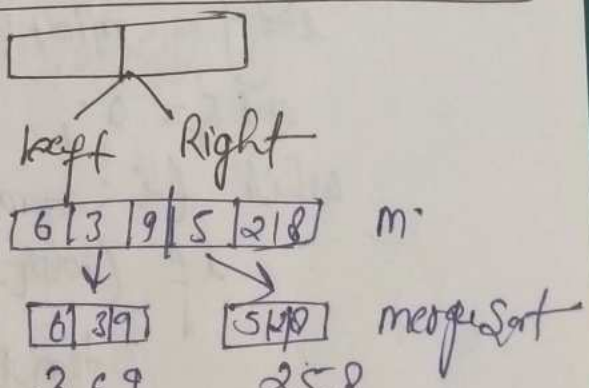


Baje

Divide

Step 2 mergeSort(left)

mergeSort(Right)

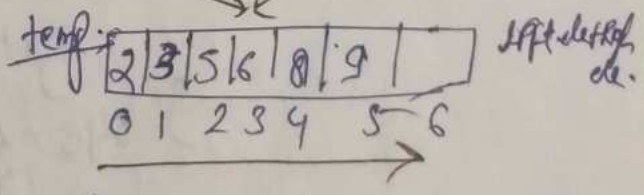


Steps merge

① Baje → $s_i > e_i$

$s_i = e_i$

② Kaam



divide

mergeSort(left, right)

merge

Code:

```
public static void mergsort(int arr[], int si, int ei){
    if(si>=ei){
        return;
    }
    int mid = si+(ei-si)/2;
    mergsort(arr, si, mid);
    mergsort(arr, mid+1, ei);
    merge(arr, si, mid, ei);
}
public static void merge(int arr[], int si, mid, int ei){
    // left (0,3) = 4  right (4,6) = 3 -> 7-0=6
    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++;
        }
    }
```

```
// left part
while (i<=mid){
 temp[k++] = arr[i++];
}
while (j<=ei){
 temp[k++] = arr[j++];
}
// copy temp to original arr.
for(k=0; i=si; k<temp.lenght; k++, i++){
      arr[i] = temp[k]
  }
}

public static void main(String args[]){
 int arr[] = {6,3,9,5,2,8};
mergeSort (arr, 0, arr.length-1);
}
```

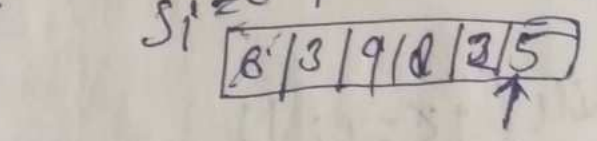
Quick Sort → average $O(n \log n)$
worst $O(n^2)$
space $O(1)$

Pivot & Partition

① Pivot :- random, median, first, last, last element

② Partition (part)

③ Quick Sort (left) (Right)

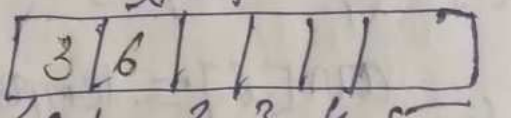


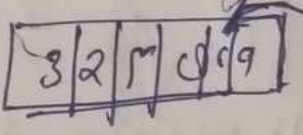
3 2 < 5 < 6 9 6

quicksort quicksort

Base

single sorted





(8) (3) (9) (6) (3) (5)

i++;

// swap

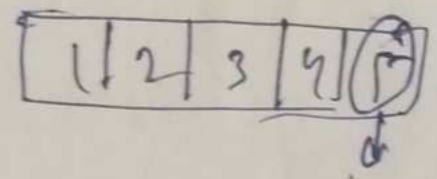
```
int temp = arr[j];
arr[j] = arr[i];
arr[i] = temp;
```

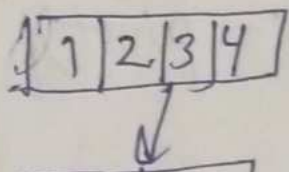
Code:

```
public static void quickSort(int arr[], int si, int ei){
    // last element
    if(si >= ei){
        return;
    }
    int pIdx = partition(arr, si, ei);
    quickSort(arr, si, pIdx-1);
    quickSort(arr, pIdx+1, ei);
}
public static int partition(int arr[], int si, int ei){
    int pivot = arr[ei];
```

```
int i = start-1;
for (int j = 0; j < end; j++){
    if (arr[j] <= pivot){
        i++;
        //swap
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
    }
}
i++;
int temp = pivot;
arr[end] = arr[i];   // pivot = arr[i]
arr[i] = temp;
return i;
}
public static void main(String args[]){
    int arr[] = {6, 3, 9, 8, 2, 5};
    quickSort(arr, 0, arr.length-1);
    printArr(arr);
}
```

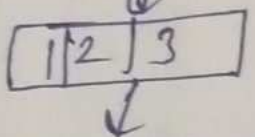
Worst Case: worst case occur when pivot is always the smallest or the largest elem





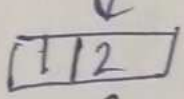
(5) n-1

last element → largest

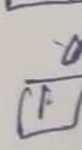


(4) n-2

Loop → n times.



(3) n-3



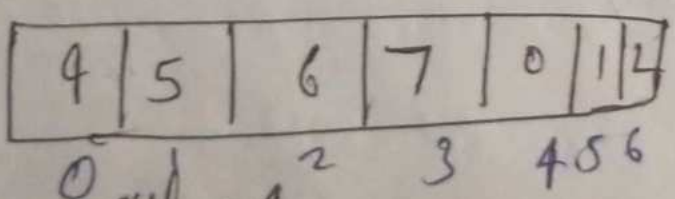
(2) n-4

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

Worst case $O(n^2)$

Search in Rotated Sorted Array

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



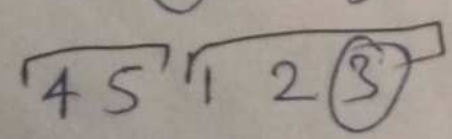
target = 0

0 1 2 3 4 5 6

out = 4

Ex rotate:- 1 2 3 4 5 (pivot)

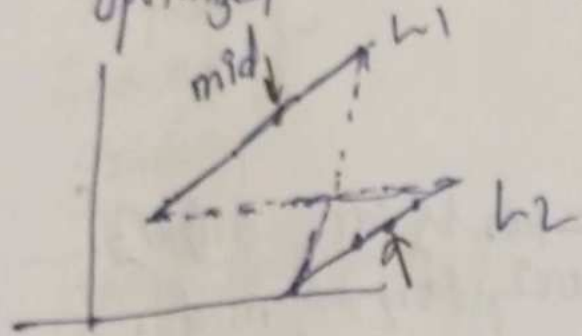
0 1 2 4 5 6 7



Linear Search: T.C → O(n)

Modified binary search

optimized: T.C → O(logn)



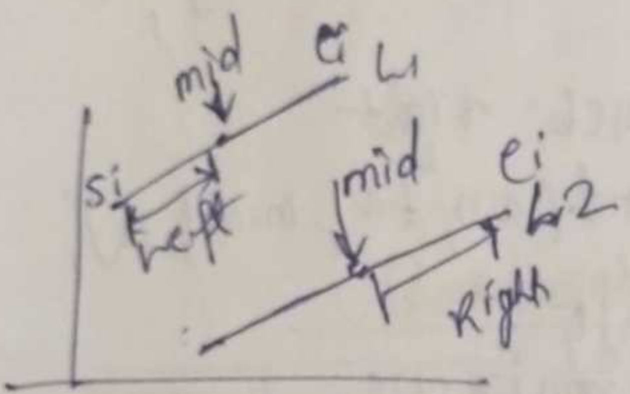
mid = (si + ei)/2

arr [mid]

L1   L2 (mid =< targ <= ei)

(si <= tar <= mid) → left, ↘ Right (false)

right   left



Case1 mid on L1   * arr[si] <= mid

Case a: L1 left (si <= tar <= mid)

↓

Case b: L1 right   else

Case2 mid on L2   arr [mid] <= arr [ei]

Case c = L1 right (mid =< tar <= ei)

Case d = mid left else.

Code:

```
public class divideConquer {
public static int search( int arr[], int tar, int si, int ei){
// kaam      if (si > ei){
                return -1; }
int mid = si+(ei-si)/2;  // (si+ei)/2
// case found
if (arr[mid] == tar ){
```

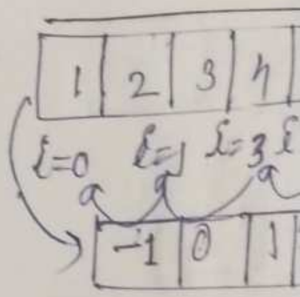
```
    return mid;
}
//mid on L1
if (arr[si] <= arr[mid]){
    // casea: left  if(arr[si] <= tar && tar <= arr[mid]){
                       return search (arr, tar, si, mid);
}   // mid on L2
                   Else:{
else {                  // caseb: right
                       return search(arr, tar, mid+1, ei);
    // casec: right     }}
if (arr[mid] <= tar && tar <= arr[ei]){
    return search (arr, tar, mid+1, ei);
}   else {
        // cased: left
        return search (arr, tar, si, mid-1);
        }
    }
}
public static void main( String args[ ]){
    int arr[]= {4,5,6,7,0,1,2};
    int target = 0; // output -> 4;
    int tarIdx = search (arr, target, 0, arr.length-1);
    Sop( tarIdx);
}}
```

(पीछे आकर देखना) Back

n-queen
Sudoku } →
gridway

Types of Backtracki

Ba



Code: public
pu

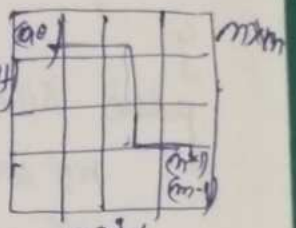
// base

// recu
arr
Change
a
}

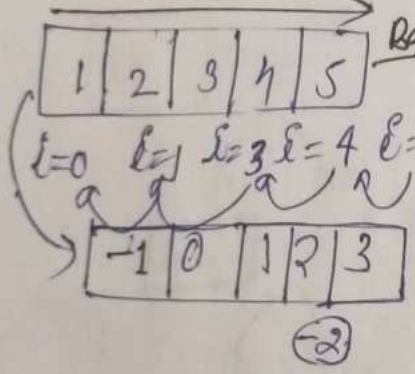
(पीछे आकर देखना) Backtracking → Back
→ Divide And Conquer.

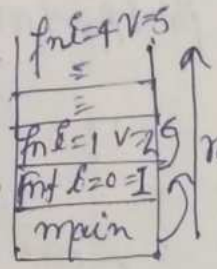
n-queen
Sudoku
grid way

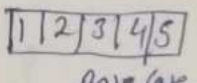
Types of Backtracking:-
① Decision
② Optimization (shortest)
③ Enumeration (region)



Backtracking on Array



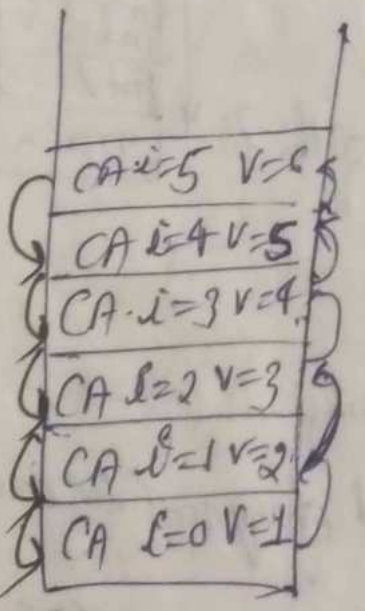


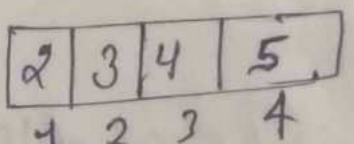


Code:
```
public class classroom{
    public static void changeArr(int arr[], int i, int val){
        // base case
        if(i == arr.length){
            printArr(arr);
            return;
        }
        // recursion
        arr[i] = val;
        changeArr(arr, i+1, val+1);
        arr[i] = arr[i] - 2;
    }
```

```
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 main(String args[]){
     int arr[]= new int[5];
      changeArr(arr,0,1);
      printArr(arr);
}
```

Call Stack





Find Subset

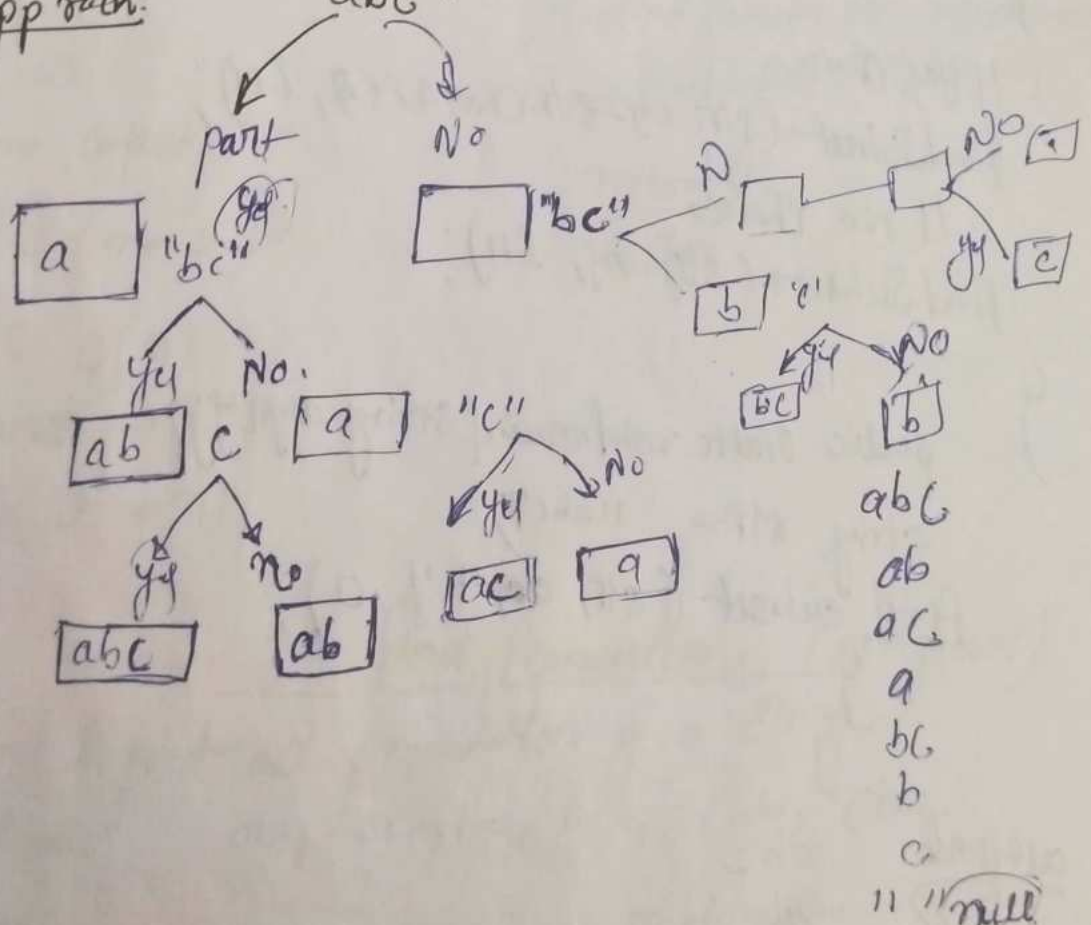
Find & print all subset of a given string

"abc" → a, b, c, ab, bc, ac, abc, "" ↳ empty (null set)

Total subset :- 2^n

String lenght $n \rightarrow 2^n$ subset

Approach:



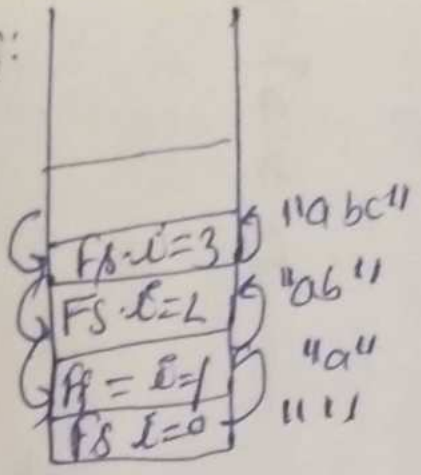
Code:-

```
public class classroom{
public static void findSubset(String str, String ans, int i){
    // base case
if(i == str.length()){
 System.out.println(ans);
    return;
}

  // yes choice
  findSubset(str, ans+str.charAt(i), i+1);
    // No choice
  findSubset(str, ans, i+1);

}  public static void main(String args[]){
    String str = "abc";
  findSubset(str, "", 0);
    }
}
```

output:

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

Call Stack:




```
public static void FindSubset(String str, int i,
                              String ans)
if(i==str.length()){
  if(ans.length==0){
    System.out.println(ans);
  }
  return;
}
yy → FindSubset(str, ans+str.charAt(i), i+1, ans);
No → FindSubset(str, i+1, ans);
```

Str = "abc";

ans = ""

i = 0

Time Complexity

→ $O(n * 2^n)$

n element

Char

S.C. $O(n)$

Subset → 2^n

$O(2^n * n)$

Find Permutation (arrangement)

- Find & print all permutation of a String

"abc" abc, acb, bac, bca, cab, cba.

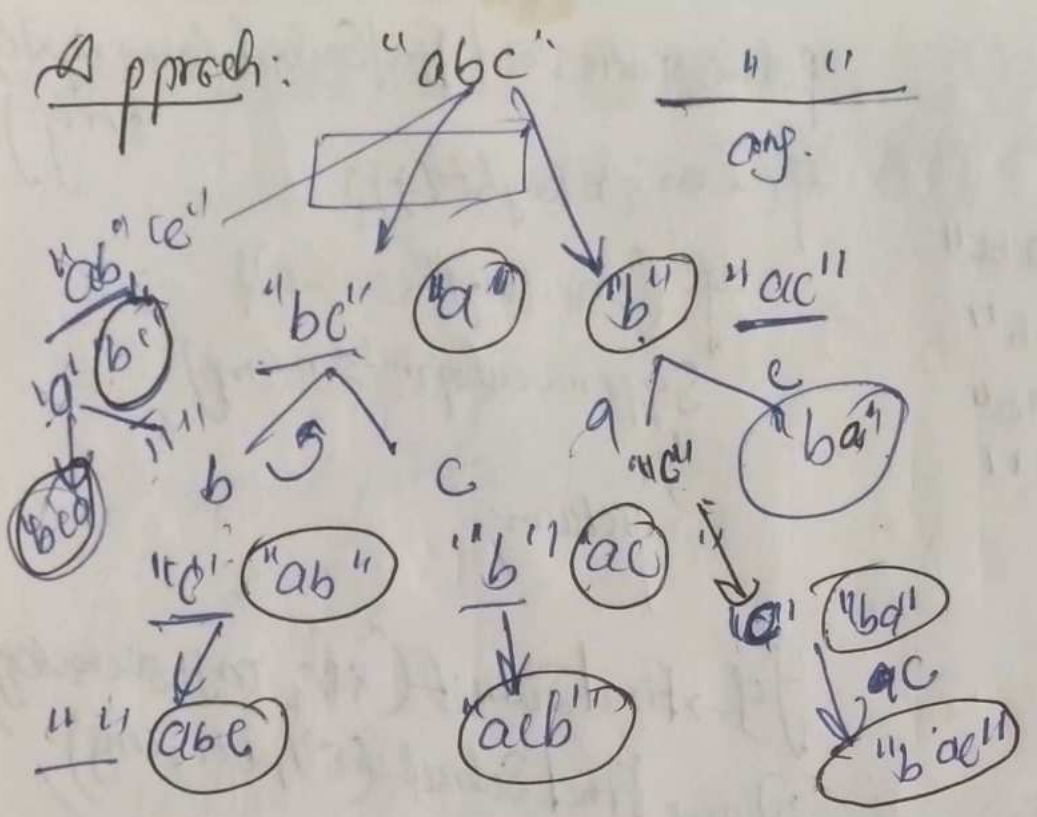
ans

length n → $\frac{n!}{0}$ permutation.

n element

$n!$

Approach: "abc"



— — — — n places: (1)

n (n-1) (n-2) (n-3)

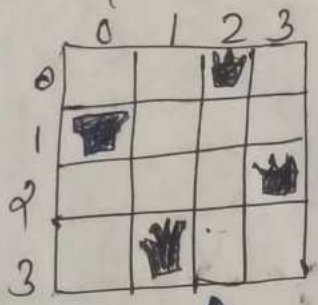
$n \times (n-1) \times (n-2) \ldots \times 1 = n!$

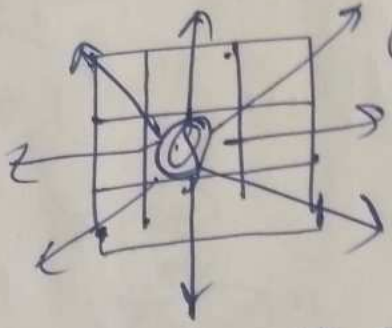
Time. $n \times n!$

```
per (str str1, String ans){
    for(i=0 to str.length()){
        ans + str.charAt(i)
        Remove
    }
```

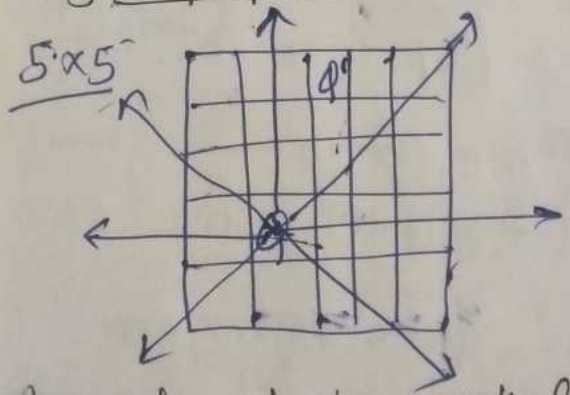
N-Queens

Place N queen on an N×N chessboard such that no 2 queen attack each other N=4.



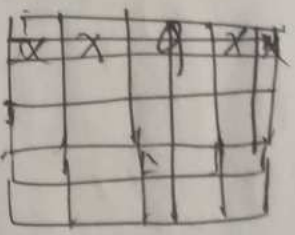


5×5

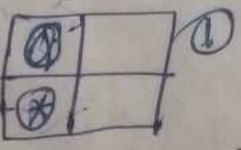


① All Solution Calculate
② Yes/No Solution
③ Count Solution

Approach:- Logic vertical:-
horizontal:-
diagonal:

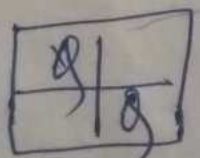


N=2

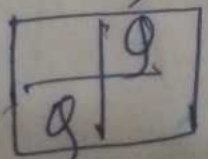


nQ
nR

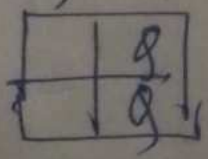
②



③



④

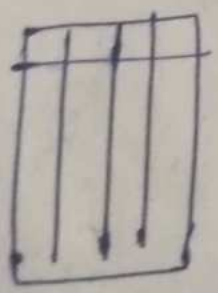


public

① publ
//ban call
t
if(now == ban
printBoard(
return;
}

```
public class classroom{
    public static void nQueen(char board[][], int row){
        ① //base case
        if(row == board.length){
            printBoard(board);
            return;
        }
        ② // Colomn loop
        for(int j=0; j<board.length; j++){
            board[row][j] = 'Q';
            nQueen(board, row+1); // Function call.
            board[row][j] = '.'; // backtracking step
        }
    }
    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] = '.';
            }
        }
        nQueen(board, 0);
    }
```

N Queen Time Complexity



n queens

1 2 3 ... n

n n n n

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

$O(n^n)$

$T(n) = 1 \text{ queen place} * T(n-1) + \text{isSafe}$

$O(n)$

$T.C. : O(n!)$

$T(n) = n * t(n-1) + \text{isSafe}()$.

N Queens - Count ways

Count total number of ways in which we can solve N queen problem.

∴ Base Case:

row == board.length

— board print —

Count++;

N Queen - print 1 solution

Yes / No.

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

isSafe (board, row, j){

place

f(n-1)

→ true

}

— place

→ (n-1)f

→ unplace

Solution not exist

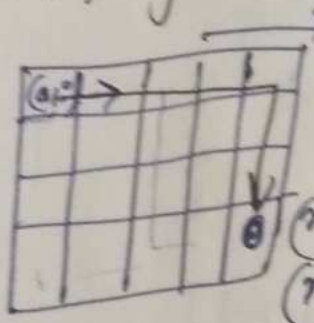
↳ return

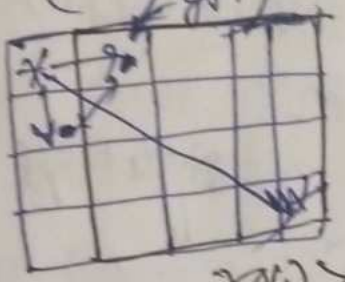
solution exist

```
public static void nQueen(char board[][], int row){
    // base
    if (row == board.length){
    // printBoard(board);
        count++;
        return;
    }
    // Column loop
    for(int j=0; j<board.length; j++){
        if(isSafe(board, row, j)){
            board[row][j] = 'Q';
            nQueen(board, row+1); // Function call
            board[row][j] = 'x'; // backtaking step
        }
    }

    public static void printBoard(char board[][]){
        System.out.println("---- chess board ----");
    }
```

Grid ways

Find number of way to reach form (0,0) to (N-1, M-1) in a NxM grid. Allowed move right → or down ↓.





w_1 w_2

$w_1 + w_2$ = total way

$f(x,y) = f(x+1,y) + f(x,y+1)$

Down Right

Approach:

(x+1, y)

row ↓ m

Target

source = (n-1, m-1)

1 way

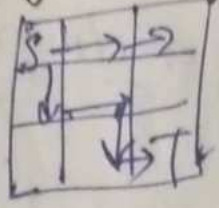
Code:

```
public class classroom{
  public static int gridways(int i, int j, int n, int m){
    // base case
    if(i==n-1 && j==m-1){ //end for last cell
      if(i==null||j==m) return 1;
      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));
```

gridway tricks

Mathtricks

Permutation-



RRDD

DRDR

way/path (n-1)D

(m-1) Rightpath

{DDDD → (n-1)
{RRRRR → (m-1)

total character = (n-1+m-1)

permutation

{DDRR ← single ways
{DRDR
{DRRD

Repeat → (n-1)D
(m-1)R

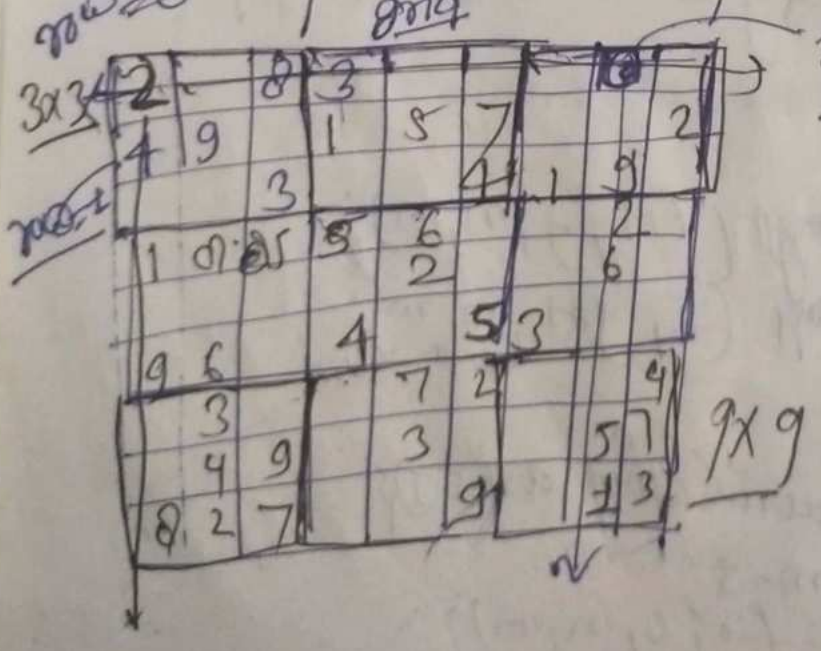
1.

$$\frac{(n-1+m-1)!}{(n-1)!\,(m-1)!}$$ ← Total Way

T.C O(n+m)

Suduko (n Queen)

Write a function to complete a Soduku.



Approach:

```
for(int e=1 .to 9)
  if safe(){
    place
      cell
  }
```

Code

```
public class class room{
  public static boolean SudukoSolver(int suduko[][], int row, int col){
    // baye Case:
    if(row == 9 && col == 9){
      return true;
    } else if (row == 9){
      return false; }
    // recursion
    int nextrow = row, nextCol = col+1;
    if (col+1 == 9){
      nextRow = row+1;
      nextCol = 0;
    }
    if (suduko[row][col] != 0){
      return SudukoSolver(suduko, nextRow, nextCol);
    }
    for (int digit = 1; digit <= 9; digit++){
      if (issafe(suduko, row, col, digit)){
        suduko[row][col] = digit;
        if (SudukoSolver(suduko, nextrow, nextCol)){
          return true;
        }
        suduko[row][col] = 0;
      }
    }
```

```
}
public static void main(String arg[]){
    int suduko[][]={{0,0,0,0,0,0,0,0,0},
                    {1,9,0,1,5,7,0,9,2},
                    {0,0,3,0,0,4,1,9,0},
                    {1,8,5,0,6,0,0,6,0},
                    {0,0,0,0,2,0,0,6,0},
                    {9,6,0,4,0,5,3,0,0},
                    {0,3,0,0,7,2,0,0,5,7},
                    {0,4,9,0,3,0,0,5,7},
                    {8,2,7,0,0,9,0,1,3}};
    if(sudukoSolver(suduko,0,0)){
     System.out.println("Solution exist");
     printSuduko(suduko);
    }else{
     System.out.println("Solution does not exist");
    }
  }
}
```

Arraylist — inbuilt

Array list → linear form

Array

- fixed size
- primitive data types can be stored

- dynamic size
- primitive data type can't stored directly
- Dynamical size.

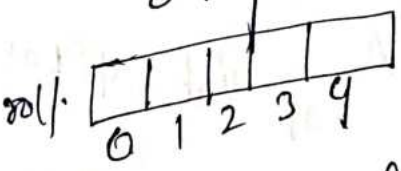
Integer → class

int a[] = new int[5]

↓

max 5 element

out of bound



Codes- import java.util.Arraylist;

// import java.util.*;

public class classroom {

public static void main(String args[]){

// String | Boolean | Float

ArrayList < Integer > list = new ArrayList<>();

ArrayList <String> list2 = new ArrayList<>();

}

operation on ArrayList

- Addelement O(1)
- get element O(1)
- Remove element O(n)
- Set Element at index O(n)
- Contain element O(n)

Code

```
import java.util.*;
public class classroom{
public static void main(String Arg[])
{
// classname objectname = new classname();
ArrayList<Integer> list = new ArrayList();
ArrayList<String> list2 = new ArrayList();
ArrayList<boolean> list3 = new ArrayList();
// Add
list.add(1);
list.add(2);
list.add(3);
list.add(4);
System.out.println(list)
```

or `list.add(1,9);` (index, 1, 4, 9)

```
// Get operation
int element = list.get(2);
System.out.println(element);
// Delete
list.remove(2);
System.out.println(list);
// Set
list.set(2,10);
System.out println(list);
// Contain elements O(n)
System.out.print(list.Contains(1));
```

list.get().

Size of Array list

Code:
```
import java.util.ArrayList;
public class classroom{
public static void main(String args[]){
ArrayList<Integer> list = new ArrayList<>();
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(4);
System.out.println(list.size());
}
// print ArrayList
for(int i=0; i<list.size(); i++){
   System.out.println(list.get(i));
}
   System.out.println()
}
```

output: 5
1 2 3 4 5

Print Reverse of Arraylist

list = 1,2,3,4,5 → 5 4 3 2 1

Code:
```
ArrayList<Integer> list = new ArrayList<>();
    list.add(1);
    list.add(2);
    list.add(3);
//Reverse
for(int i = list.size()-1; i>=0; i--){
    System.out(list.get(i) + " ");
}
System.out();
}
```

Find Maximum ele on ArrayList

list = 2, 5, 9, 3, 6 Max = -∞ (Integer.MIN_VALUE)

```
ArrayList<Integer> list = new ArrayList<>();
    list.add(2);
    list.add(5);
    list.add(9);
int max = Integer.MIN_VALUE;
for (int i=0; i<list.size(); i++){
    if (max<list.get(i)){
        max = list.get(i);
    }
}
    Sop("max element =" + max);
}
```

out ⇒ 9

Swap 2 Number

list = 2, 5, 9, 3, 6 index: idx1 = 1, idx2 = 3.

```
int temp = list.get(idx1);
list.set(idx1) = list.get(idx2);
list.set(idx2) = temp;
```

```
ArrayList<Integer> list = new ArrayList<>();
    list.add(2);
    list.add(5);
    list.add(9);
    list.add(3);
    list.add(6);
int idx1 = 1, idx2 = 3
swap(list, idx1, idx2);
}
```

→ psvm swap(ArrayList<Integer> list, int idx1, idx2)

```
int temp = list.get(idx1);
list.set(idx1, list.get(idx2));
list.set(idx2, temp);
}
```

original → [1 5 9 3 6]

swap → [1 3 9 5 6]

Sorting an ArrayList ← Scratch

{ Bubble
Selection
Insertion
Merge
Quick

optimized

Collections.sort();
Collections.sort(list);

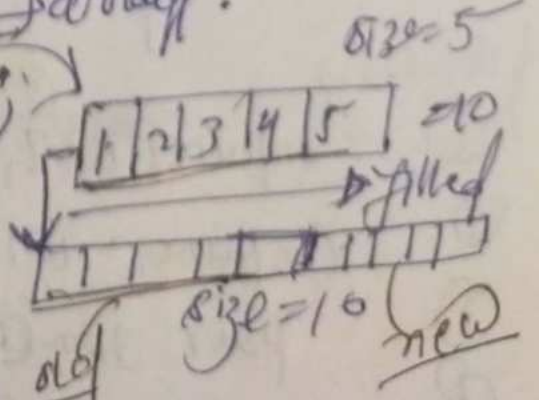
✓ Collections → class (sort) → Ascending order
Collection → interface

Code: // import java.util.ArrayList; or import java.util.*;
// import java.util.Collections;

```
public class classroom{
public static void swap(ArrayList<Integer> list, int idx1, int idx2){
    int temp = list.get(idx2);
    list.set(idx1, list.get(idx2));
    list.set(idx2, temp);
}
    public static void main(String args[]){
        ArrayList<Integer> list = new ArrayList<>();
            list.add(2);
            list.add(5);
            list.add(9);
            list.add(3);
            list.add(6);
    Say(list);
    Collections.sort(list);
    Say(list);
    };
```

Implementation → array.

ArrayList<Integer> list = new AL<>();



Multidimensional Arrays -

list1: 12345 (1 table)
list2: 246810 (2 table)
list3: 3691215 (3 table)

```
ArrayList<ArrayList<Integer>> mainList =
        new ArrayList<>();
ArrayList<int> arr = new AL<>();
mainList.add(arr);
```

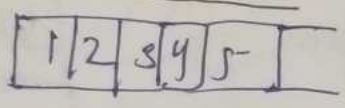
```
public static void main(String args[]){
ArrayList<ArrayList<Integer>> mainList = new ArrayList<>();
ArrayList<Integer> list = new ArrayList<>();
    list.add(1); list.add(2);
    mainList.add(list);
    ArrayList<Integer> list2 = new ArrayList<>();
    list2.add(3); list2.add(4);
    mainList.add(list2);
for(int i=0; i<mainList.size(); i++){
    ArrayList<Integer> currList = mainList.get(i);
for(int j=0; j<currList.size(); j++){
        Soy(currList.get(j)+" ");
    }
    Soy();
  }
}
```

Question

1. Container with most water
2. Container with MW & trapping water
3. pair sum-1
4. pair sum-2

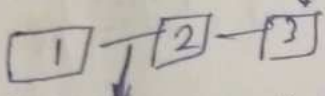
Linked list

Linear list (data structure)



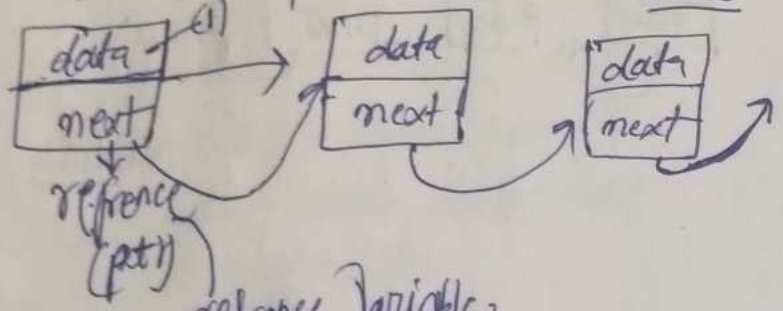
dynamic

Node



(Node1) link Node2 Node3

Class → Node.



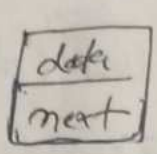
Syntax

```
Class node {
    int data;
    Node next;
public node (int data){
    this.data = data.
    this.next = null;
}
}
```

Code.

```
public class linked list {
    public static class Node {
        int data;
        Node next;
    public static void main node(int data){
        this.data = data;
        this.next = null;
    psvm() }
}
```

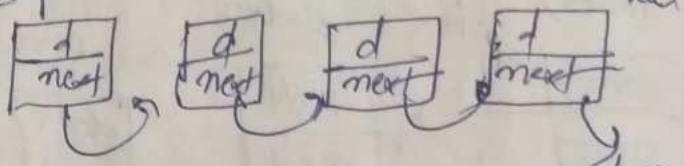
Head node:



Tail

```
Class Node{
    int data;
    Node next;
    public Node(int data){
        this.data = data;
        this.next = null;
    }
}
```

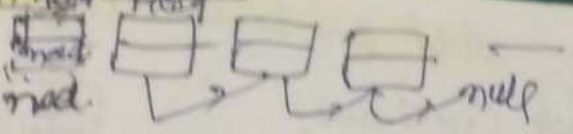
Head



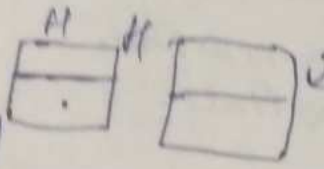
Code

```
public class LinkedList {
    public static class Node{
        int data;
        Node next;
    public Node(int data){
        this.data = data;
        this.next = null;
    }
    }
    public static Node head;
    public static Node tail;
    // Method
        add()
        remove()
        print()
        search()
public static void main(String args[]){
    LinkedList ll = new LinkedList();
}
```

Add in Linked list

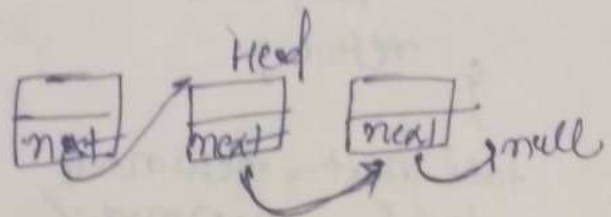


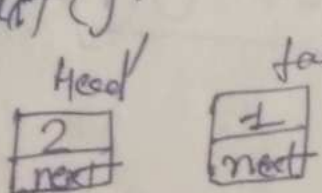
- add first
- add last



① To create new node
② new node next = head
③ head = new node

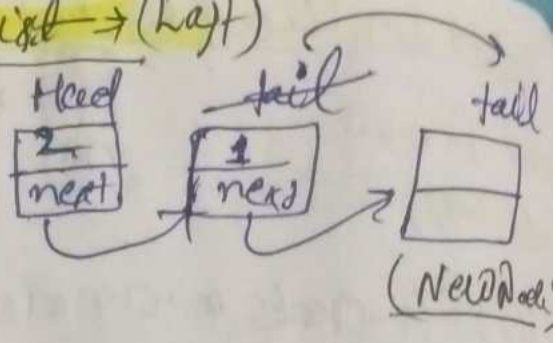
→
```
public class LinkedList {
    public static class Node {
        int data;
        Node next;
    public Node (int data) {
        this.data = data;
        this.next = null;
    }
}
public static Node head;
public static Node tail;
public void addFirst (int data) {
    // step1 = create new node
    Node newNode = new Node (data);
    if (head == null) {
        head = tail = newNode;
        return;
    }
// step2 - New Node next = head
    newNode.next = head; // link.
// step3 - head = new Node
    head = newNode;
}
public static void main (String args[]) {
    LinkedList LL = new LinkedList();
    LL.addFirst (1);
    LL.addFirst (2);
}
```





T.C = O(1).

Add in linked list → (Last)

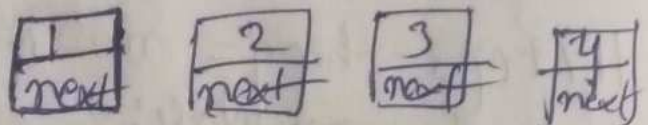


① Create a node (new node)
② tail.next = new Node
③ tail = new Node.

```
public void addLast(int data){
    Node newNode = new Node(data);
    if(head == null){
        head = tail = newNode;
        return;
    }
    tail.next = newNode;
    tail = newNode;
}
public static void main(String args[]){
LinkedList ll = new LinkedList();
    ll.addFirst(2);
    ll.addFirst(1);
    ll.addLast(3);
    ll.addLast(4);
    }
}
```

1 → 2 → 3 → 4 → null

Head tail

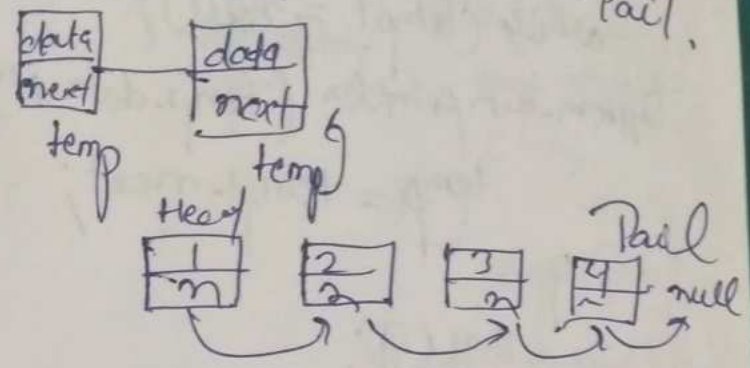


T.C = O(n)

Print a linked list

```
Node temp = head;
print(temp.data);
temp = temp.next;
while(temp != null){
}
```

1 → 2 → 3 → 4 → null



output 1 2 3 4

Code:

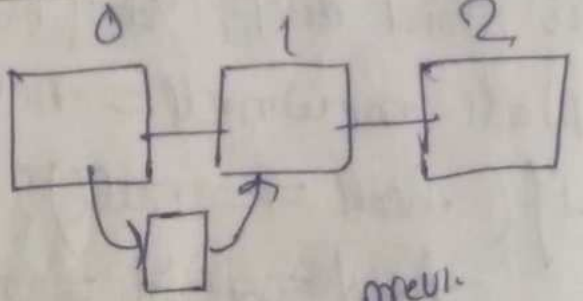
```
public class LinkedList{
  public static class node{
    int data;
    int next;
  public node(int data){
    this.data = data;
    this.data = null;
  }}
  public static Node head;
    public static Node tail;
  public void addFirst(int data){
    Node newnode = new Node(data);
    if(head == null){
      head = tail = new node;
    return;
    }
    head.next = new Node;
    } Head = new nal;
```

```
public void print(){
    Node temp = head;
    while (temp != null){
System.out.println (temp.data + " ");
        temp = temp.next;
    }
    say();
}
public static void main(String arg[]){
LinkedList ll = new LinkedList();
ll.addFirst(2);
ll.addFirst(1);
ll.print();
ll.addFirst(4);
ll.print()
}
}
```

```
if(head == null){
    say("LL is empty");
    return;
}
```

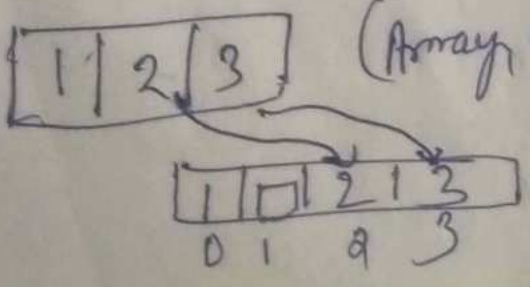
Add in the middle

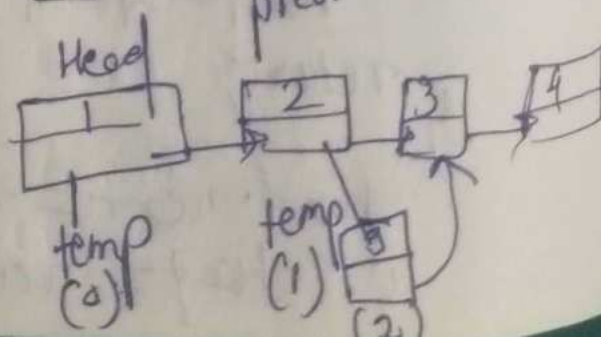
add (index, data)



index=1

(Array, ArrayList)





```
public void add(int idx, int data){
    Node newNode = New Node(data);      → if(idx==0){
    Node temp = head;                       addFirst(data)
    int i = 0;                              return;
                                          }
 While( i<idx-1){
    temp = temp.next;
    i++;
 }
    // i = idx-1; temp -> prev
    newNode.next = temp.next;
    temp.next = newNode;
 }
public Static void main(String args[]){
    LinkedList ll = new LinkedList();
        ll.addFirst(2);
        ll.add(2, 9);
        print();
}
```

```
Node temp = head
i = 0
While (i < idx-1){
    temp → next
    i++;
}
```

temp (prev)

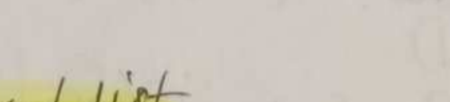
① newNode.next = temp.next

② temp.next = newNode

output:

Size of Linked list

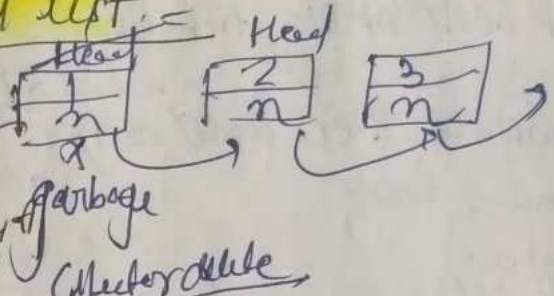
Static int size = 1



size = 4

हर new node के बाद (size++) होगा ·

Remove in a linked list :-



- Remove first
- remove last

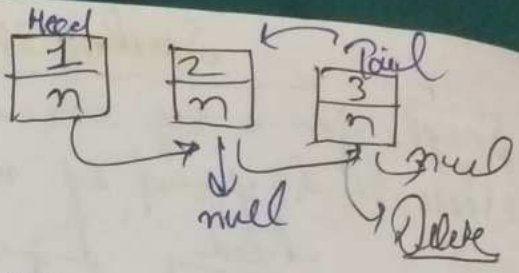
Singly list :- link होती है और एक होता है

```
//Method
public int removeFirst(){
    int val = head.data;
    head = head.next;
    return val;
}
public static void main(String args[]){
    LinkedList ll = new LinkedList();
    ll.removeFirst();
    ll.print();
}
```

output : 2 3 4 5

```
Edge Case  if(siz==0){
               System.out.println("ll is empty");
               return Integer.MIN_VALUE;
           }else if(siz==1){
               int val = head.data;
               head = tail = null;
               return val;
           }
```

Remove Last



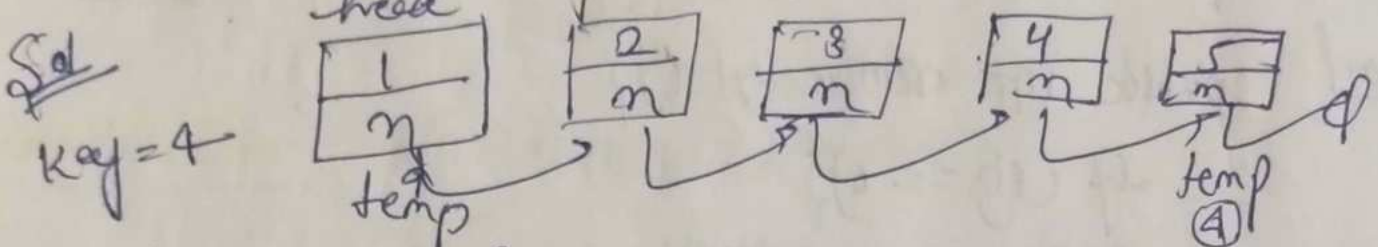
① prev.next = null
② tail = prev

```
//method  public int removeLast(){
          if (size == 0){
          Say("LL is empty");
          return Integer.MIN_value;
        } else if ( size == 1){
            int val = head.data;
             head = tail = null;
              size = 0;
               return val;
        }
          // prev = size - 2
      for( int i=0; i< size-2; i++){
          prev = pre.next;
        }     int val = prev.next.data; //tail.data.
        prev.next = null;
          tail = prev;
            size--;
        }
public static void main ( ){
dl.removeLast();
 ll.print()
}}
```

Seed (Iterative) Linear Search

Search for a key in a linked list return the position where it is found if not found return -1.

Sol




```
Node temp = head;
while (temp != null){
    Check (temp.data = key)
}

public int ~~intSearch~~(int key){
        Node temp = head;
        int i = 0;
    while(temp != null){
        if (temp.data == key){ // key found
            return i;
        }
        temp = temp.next;
        i++;
    }
    // key not found
    return -1;

    p.s.v.m( )
    LinkedList ll = new LinkedList();
    ll.addFirst(2);
                1);
```

```
ll.addLast(4);
ll.addLast(5);
ll.addFirst(2,3);
ll.print(); // 1->2->3->4->5
Say(ll.iterSearch(3));
Say(ll.rtrSearch(10));
```

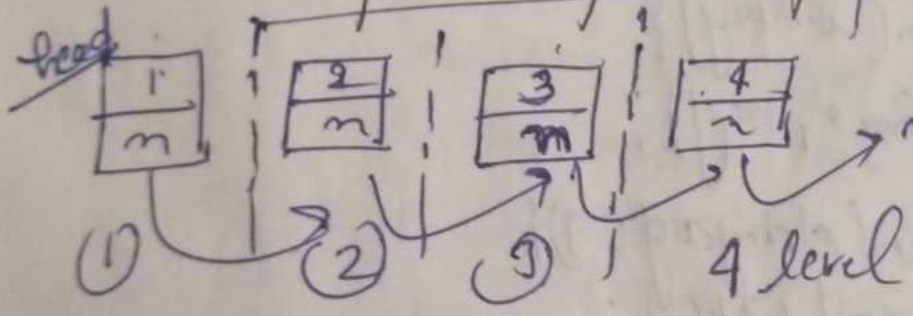
T.C = O(n)

```
}
```

out 2
-1

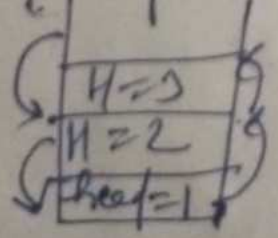
Search (Recursive)

Search for a key in a linked list. Return the position where it is found if not found return -1. We Recursion



Base Case:. head = null → if (head.data == key)
return -1; return 0
searchRec(head.next);

1 → 2 → 3 → 4 → null key=3
Head

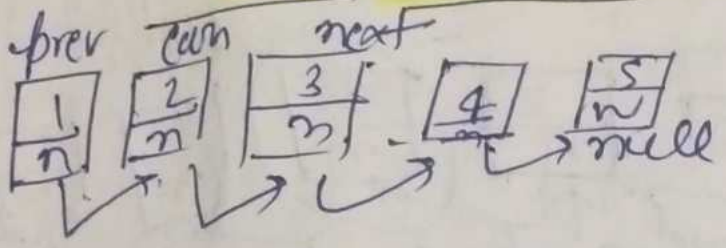


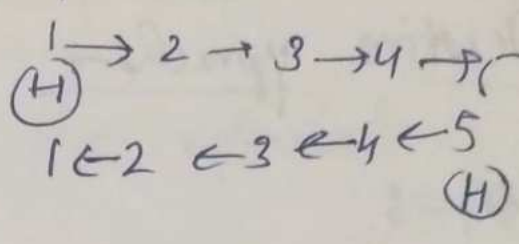
Code:

```
                    helper
public int recSearch(Node head, int key){
    if (head == null){
        return -1;
    }
    if (head.data == key){
        return 0;
    }
    int idx = helper(head.next, key);
    if (idx == -1){
        return -1;
    }
    return idx+1;
}
public int recSearch(int key){
    return helper(head, key);
}
public static void main(String args[]){
    LinkedList ll = new LinkedList();
    ll.addFirst(1);
    ll.addFirst(2);
    ll.add(3);
    ll.print();
    System.out.println(ll.recSearch(3));
    System.out.println(ll.recSearch(10));
}
```

output: 2
-1

Reverse a linked list





① next = curr.next;
② curr.next = prev
③ prev = curr
④ curr = next

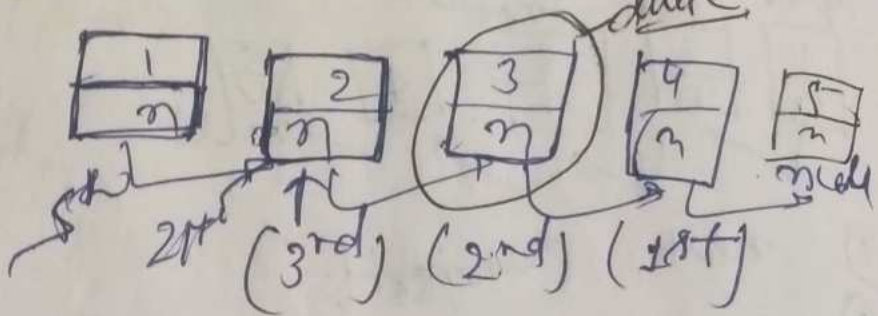
Method

```
public void reverse(){
    Node prev = null;
    Node curr = head; // fabl;
    Node next;
    while (curr != null){
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
public static void main(String args[]){
    LinkedList ll = new LinkedList();
    ll.addFirst(2);
    ll.addFirst(3);
    ll.add(2,1);
    ll.print();
    ll.reverse();
    print();
}
```

output 5->4->3->2->1

Find & remove N^{th} node from End

Iterative approach:-



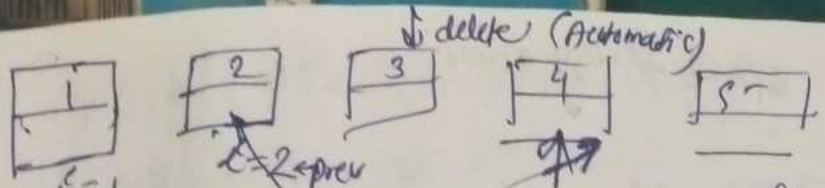
$n = 3$

n^{th} end
start

$(size - n + 1)^{th}$ start

Code:

```
public void deleteNthFromEnd(int n){
    // Calculate size
    int sz = 0;
    Node temp = head;
    while (temp != null){
        temp = temp.next;
        sz++;
    }
    if( n == sz){
        head = head.next; // remove first
        return;
    }
    // sz - 3
    int i = 1;
    int toFind = sz - n;
    Node prev = head;
    while (i < toFind ){
        prev = prev.next;
        i++;
    }
    prev.next = prev.next.next;
}
public void printList(){
    Node currNode = head;
    while (currNode != null){
        System.out.print(currNode.data + " ");
        currNode = currNode.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    LinkedList ll = new LinkedList();
    ll.addFirst(1);
    ll.addFirst(2);
    ll.addFirst(3);
    ll.addFirst(4);
    ll.addFirst(5);
    ll.printList();
    ll.deleteNthFromEnd(3);
    ll.printList();
}
```



$S3 - n = 2 = 5 - 2$
33

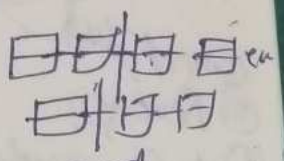
prev.next = pre.next.next.

Check if LL is a palindrome → slow fast concept → reverse half LL

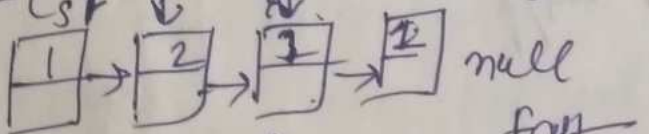
1221
1441
CaC
madam
race car

① LL → ArrayList → O(n) → TC
array O(n) → SC
String

Step ① Find mid node
② 2nd half reverse
③ Check if 1st half = 2nd half.
left = right



Even Case: S F

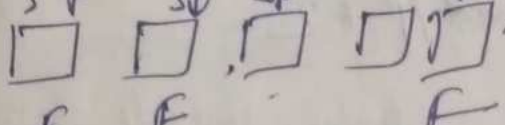


① mid
② size = size/2

① Slow Fast technique

(Start) slow = head ||+1
(Start) fast = head ||+2

Odd Case: S F SL SF F



fast.next = null.

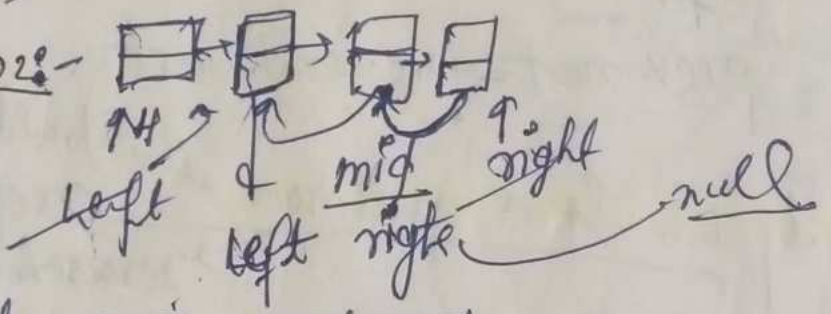
① Find mid [fast != null]
fast.next != null

Find middle:

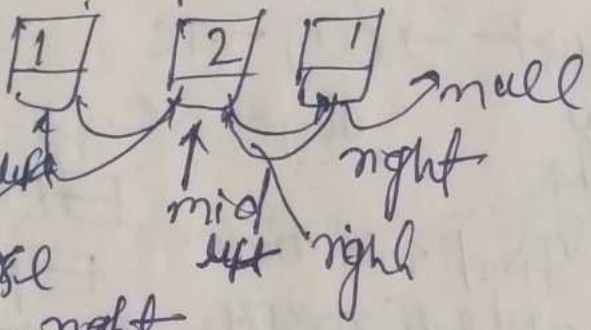
```
public Node findMid(Node head){
    Node slow = head;
    Node fast = head;
    while(fast != null && fast.next != null){
        slow = slow.next;      ||+1
        fast = fast.next.next; ||+2
    }
```

return slow; // slow is my mode
}

Step2:-



odd



① mid
② Reverse
③ Left = right

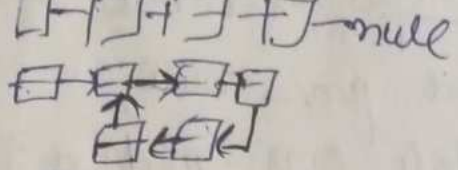
Code

```
public Node find mid (Node head){
    Node slow = head;
    Node fast = head;
while (fast != null && fast.next != null){
    slow = slow.next; // +1
    fast = fast.next.next // +2
}
    return slow;
}
    public boolean checkpalidrm (){
// step1 → find mid
// step2 → Reverse second half
// step3 → check left half & right half
if( head == null || head.next != null){
    return true;
}
```

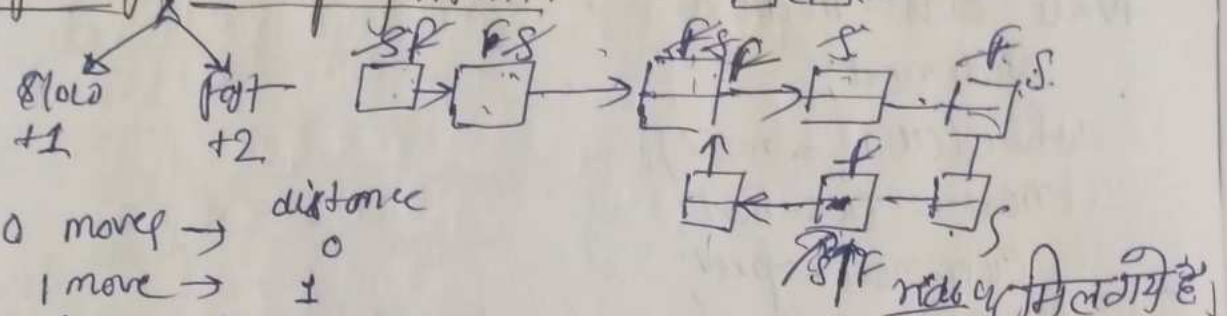
```
Node midNode = FindMid(head);
// step2 → Reverse 2nd Half.
Node prev = null;
Node curr = midNode;
Node next;
while (curr != null){
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
Node right = prev; // right half head
Node left = head;
// step3  Check left and right half
while (right != null){
  if (left.data != right.data){
    return false;
  }
  left = left.next;
  right = right.next;
}
return true;
}

ll.print();  // 1 > 2 > 2 > 1
Sy(ll.checkPalidrome());
}
```

Linked List - 2

Detect a loop/cycle in LL :-

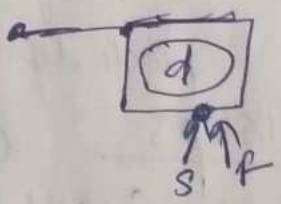


Floyd's Cycle finding Algorithm :-



0 move → distance 0
1 move → 1
2 move → 2
3 move → 3
4
5
6
7

slow = fast
Cycle ✓ true.
false



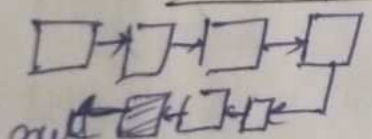
Logic

```
slow = head
fast = head
while (f != null && f.next != null) {
        slow → +1
        fast → +2
    (slow == fast) // cycle
        return true;
}
```

```
public boolean isCycle() {
    Node slow = head;
    Node fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next; // +1
        fast = fast.next.next; // +2
        if (slow == fast) {
```

```
return true; // Cycle exist
}
return false; //Cycle does not exist
}
public static void main(String args[]){
    head = new Node(1);
    head.next = new Node(2);
    head.next.next = new Node(3);
    head.next.next.next = head;
    // 1 -> 2 -> 3 -> 1
System.out.println("is cycle");
}
```

Remove a loop / cycle in a linked list



Approach: ① detect cycle

② slow = head

③ slow → +1
fast → +1

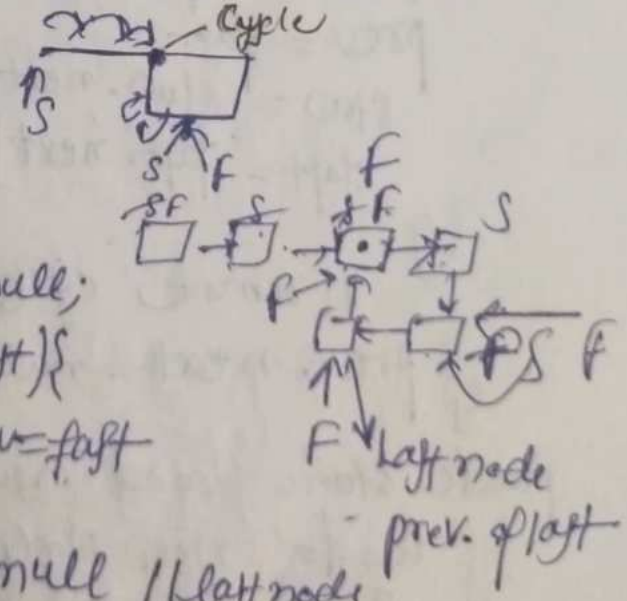
④ while (slow != fast){
    slow → +1
    fast → +1
}

prev = null;
prev = fast

prev.next = null // last node

① Find last node

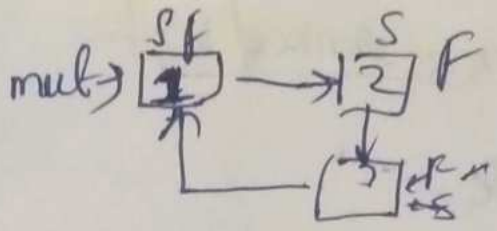
② last node.next = null.



```
public static void removeCycle(){
    Node slow = head;
    Node fast = head;
    boolean cycle = true;
while (fast != null && fast.next != null){
    slow = slow.next;
    fast = fast.next.next;
if (fast == slow){
    cycle = true;
    break;
}
if (cycle == false){
    return;
}
// find meeting point
    slow = head;
Node prev = null;
while (slow != fast){
    prev = fast;
    slow = slow.next;
    fast = fast.next;
}
// remove cycle -> last.next = null
prev.next = null;
}
public static void main(String args[]){
    head = new Node(1);
    head.next = new Node(2);
    head.next.next = new Node(3);
    head.next.next.next = head;
```

Step
① Detect cycle
② Find meeting point
③ remove cycle
last.next = null

```
// 1 -> 2 -> 3 -> -1
Soy(" is cycle ");
remove(cycle);
Soy(" is cycle ");
}
```



Java Collection Framework

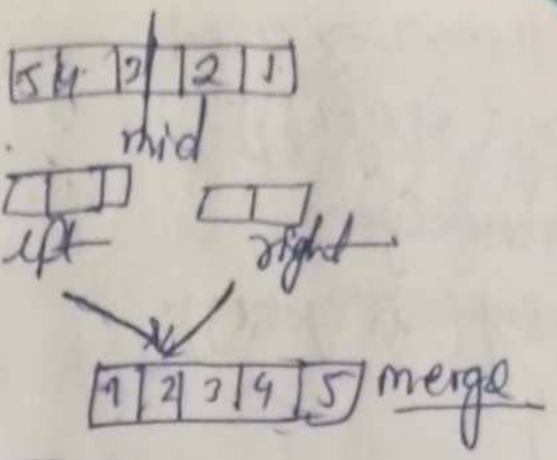
Arraylist → import java.util.Arraylist
import java.util.linklist;

J.C.F: Arraylist
linked list
stack
Queue
Hashset
Hashmap

↳ LL in java Collection framework:—

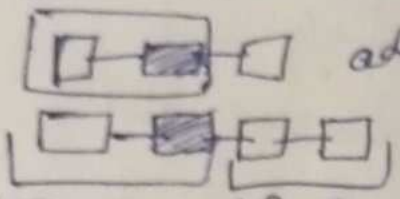
```
import java.util.Linked list;
public class classroom {
  public static void main(String args[]){
    // create
  Linked list < Integer > LL = new Linked list <>();
    // add
    LL.addLast(1);
    LL.addLast(2);
    LL.addFirst(0);
      // 0 -> 1 -> 2
    Soy(LL);
    // remove
    LL.removeLast();
      LL.removeFirst();
  }
```

Merge Sort on a linked list



Steps:-

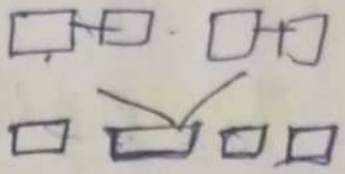
① LL → middle

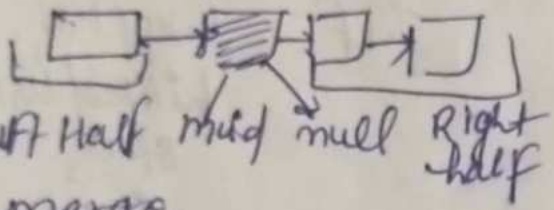


② ① Left half → MS

② Right half → MS

mid.next = null



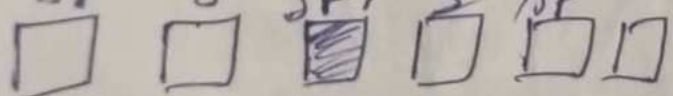


③ merge

Mid

```
slow = head
fast = head.next
while(fast != null && fast.next != null){
    slow → +1
    fast → +2
}
slow ⇒ middle node
```

Ex

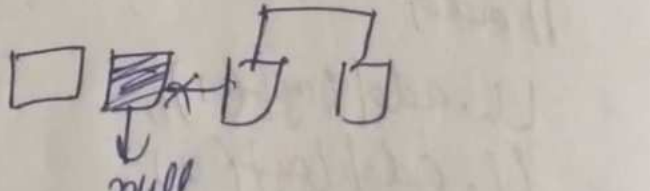


② rightHead = mid.next

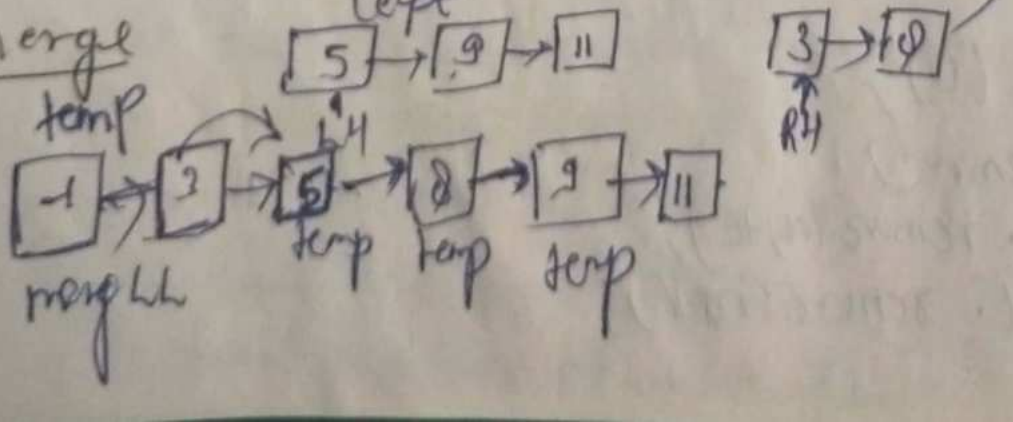
mid.next = null

MS (head) // left half

MS (right) // right half



③ merge



Code:

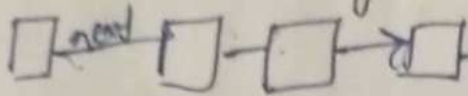
```
public Node mergeSort(Node head){
```

Zig-Zag linked list

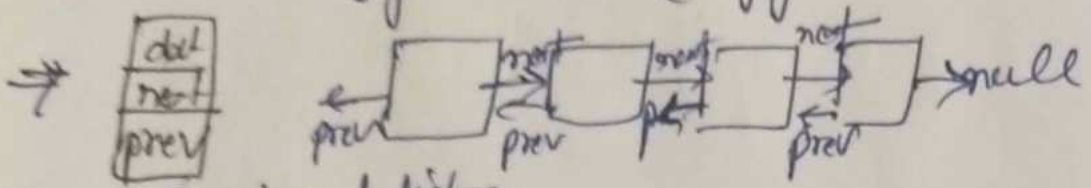
For a linked list of the form: L(1) -> L(2) -> L(3) -> L(4) ... L(n-1) -> L(n)
Convert it into a zig-zag form ie. L(1) -> L(n) -> L(2) -> L(n-1) -> L(3) ...

Doubly linked list

Approach:



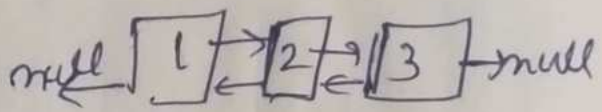
single direction. (Singly linked list)



doubly linked list

```
Node{
  int data;
  node next
   Node prev
}
```

```
public class DoubleLL{
   public class Node{
        int data;
        Node next;
         Node prev;
   public Node (int data){
     this.data = data;
       this.next = null;
        this.prev = null;
   }}
public static Node head;
  public static Node tail;
  public static int size;
 // add
public void addFirst(int data){
     Node newNode = new Node(data);
      if(head == null){
                head = tail = newNode;
```



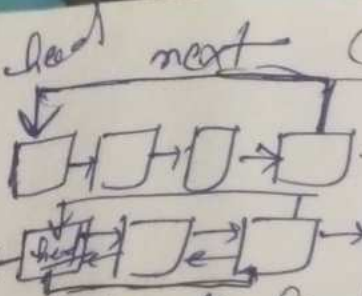
① Create node

② newNode.next = head

head.prev = newNode

head = newNode.

```
return;
}
newNode.next = head;
head.prev = newNode
head = NewNode;
}
//print
public void print(){
  Node temp = head;
while(temp != null){
  Soy(temp.data + "<->");
        temp = temp.next;
}.Soy();
}

public static void main(String args[]){
    DoubleLL dll = new DoubleLL();
        dll.addFirst(3);
        dll.addFirst(2);
        dll.addFirst(1);
       dll.print();
       dsoy(dll.size);
       }
}
```



Circular linked list

Singly linked list

Doubly linked list

जहां last node first node से connect हो जाता है वह Circular linked list बनाता है।

add
remove

Stack Data structure

STACK

Real life

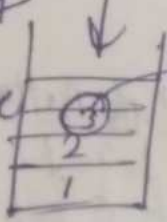
Carrom.

Coin

Stone.

Operation:

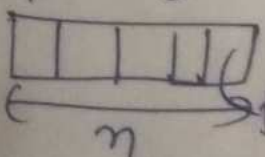
① push O(1) add
② Pop O(1) remove
③ Peek O(1)



LIFO Last in First out.

Array

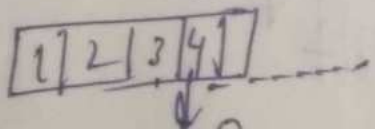
fixed size

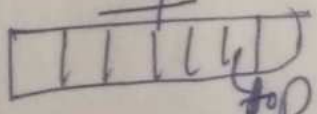


Stack == full

Arraylist

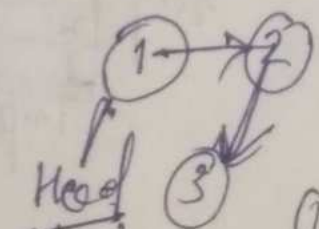
variable size





linked list

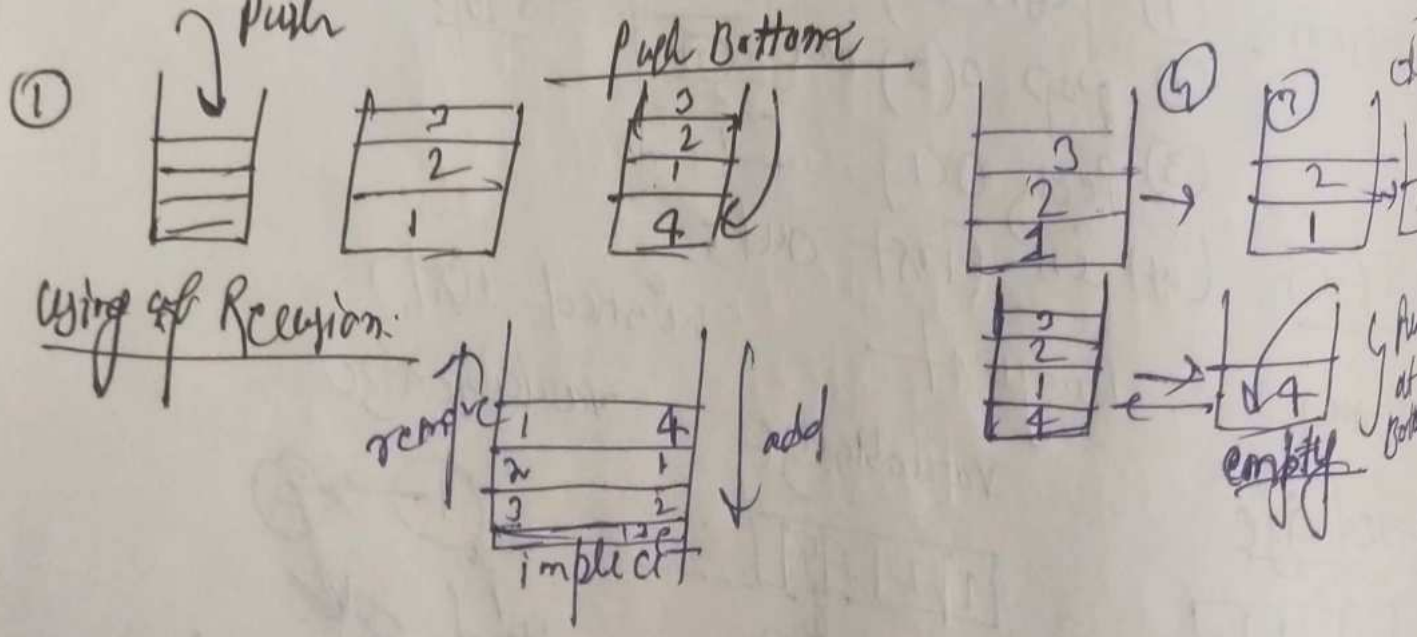
variable size



O(1)

Stack using Array

```
import java.util.*;
public class Stack{
public static void main(String[]args){
   Stack<Integer> S = new Stack<>();
    S.push(1);
    S.push(2);
    S.push(3);
while(!S.isEmpty()){
      Sop(S.pop());
    S.pop();
}
}
```



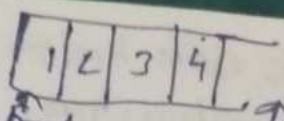
Queue

FIFO

Line में/में

Counter

First in first out

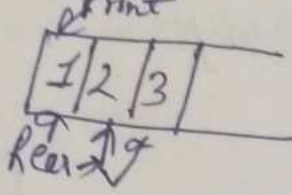


operation

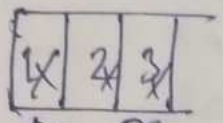
① Add O(1)
(Enque)

② Remove O(1)
Dequeue

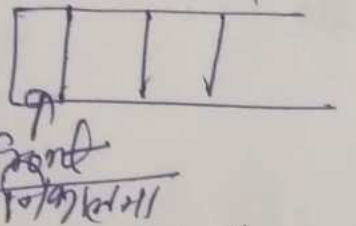
③ Peek O(1)
Front



[Deque]



1 2 3



Front
Rear+1

Implementation

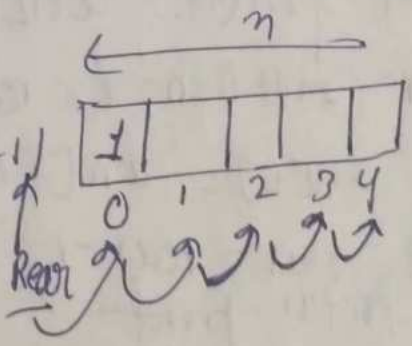
Array

- fixed size "n"
- Remove O(n)

Linked list

Stack

CircularQueue



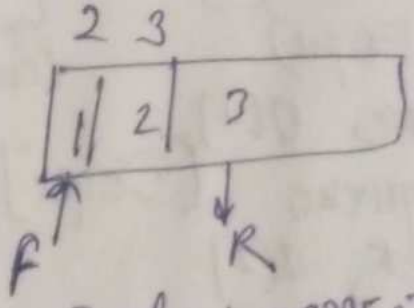
Code:

```
public class Queue
    static class Queue
static int arr[];
static int size;
static int rear;
Queue (int n){
    arr = new int [n];
    size = n;
    rear = -1;
}
```

```
public static boolean isEmpty(){
    return rear == -1;
}
```

// Add :—

```
public static void add(int data){
    if (rear == size-1){
        say("Que is full");
        return;
    }
    rear = rear+1;
    arr[rear] = data;
}
```



Front = 1 = arr[0]

// Remove

```
public static int remove(){
    if (isEmpty()){
        System.out.println("empty queue");
        return -1;
    }
    int front = arr[0];
    for (int i=0; i<rear; i++){
        arr[i] = arr[i+1];
    }
    rear = rear-1;
    return front;
}
```

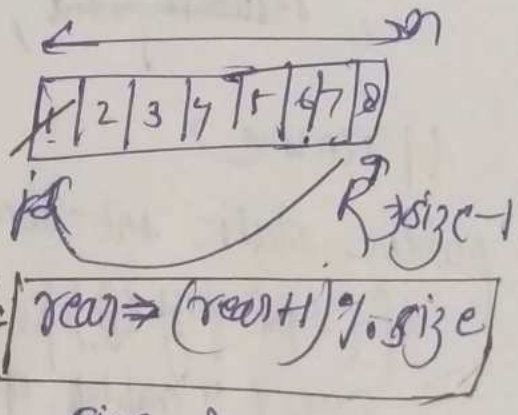
// peek

```
public static int peek(){
    if (isEmpty()){
        say("empty queue");
```

```
    return -1;
  }
  return arr[f];
 }
}

public static void main(String args[]){
  Queue q = new Queue(5);
    q.add(1);
    q.add(2);
    q.add(3);
 while(!q.isEmpty()){
  Sop(q.peek());
  q.remove();
 }
}
```

add O(1)
remove O(n)

Circular Que



formula: rear = (rear+1) % size

size = 8

rear = (7+1) % 8

= 0

Front = (front+1) % size

```
static class Queue{
 static int arr[];
 static int size;
 static int rear;
 static int front;
Queue(int n){
 Queue arr = new int[n];
  size = n;
    rear = -1;
 }  front = -1;
  public static boolean isEmpty(){
```

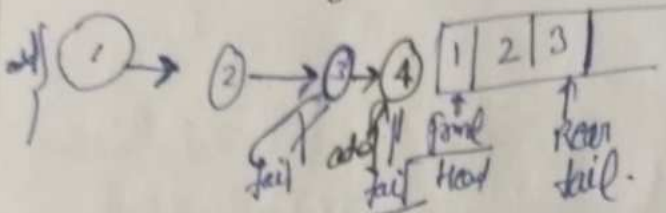
```
return rear == -1 && front == -1;

}
public static boolean isFull(){
    return (rear+1)%.size == front;
}
//add
public static void add(int data){
    if(isFull()){
    Soy("queue is full");
     return;
    }
    if(front == -1){
      front=0;
    }
     rear=(rear+1)%.size;
      return front; arr[rear] = data;
    }
//remove
public static int remove(){
    if(isEmpty()){
     Soy("empty que");
      return -1;
    }
int result = arr[front];
   front = (front+1)%.size;
     return result;
}
```

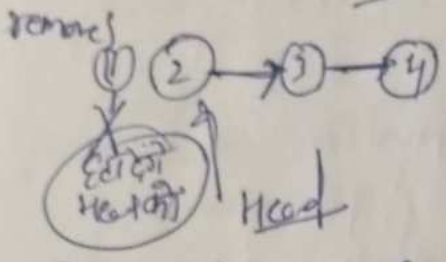
```
if(rear == front){
    rear = front = -1;
}else{
    front = (front+1)%.size;
}
return result;
}
```

```
// peek
public
  if(i
   Soy
   ret
 }
 re
}}

Soy
q.
Soy(
q..
//
while
}
```

```
// peek
public static int peek(){
  if(isEmpty()){
   Say("empty queue");
   return -1;
  }
   return arr[front];
  }

   public static void main(String arg[]){
      Queue q = new Queue(3);
        q.add(1);
        q.add(2);
        q.add(3);
  Say(q.remove());
  q.add(4);
Say(q.remove());
  q.add(5);
  // 123
while(!q.isEmpty()){
    Say(q.peek());
      q.Rremove();
    }
  }
```

Question — Queue using linked list





Code: public class Queue {
  static class Node {
    int data;
    Node next;
  Node (int data) {
  this.data = data;
  this.next = null;
  }
  }

  static class Queue {
  static Node head = null;
  static Node tail = null;
public static boolean isEmpty() {
  return head == null && tail == null;
  }

  //add
  public static void add(int data) {
    Node newNode = new Node(data);
  if (head == null) {
      head = tail = newNode;
      return;
  }

```
    tail.next = newNode;
    tail = newNode;
  }
// remove
public static int remove(){
  if (isEmpty()){
    Sop("empty queue");
    return -1;
  }
  int front = head.data;
    return if (rear == front){
                tail = head = null;
      } else {
                head = head.next;
      }
      return front;
  }
// peek  public static int peek(){
          if (isEmpty()){
            Sop("empty que")
            return -1;
          }
          return head.data;
        }
public static void main(String[] args){
  Queue q = new Queue(3);
    q.add(1);                while(!q.isEmpty()){
    q.add(2);                  Sop(q.peek());
    q.add(3);                  q.remove();
                             }
}
```

Java Collection framework

```
import java.util.*;
public class Queue{
public static void main(String args[]){
    // Queue q = new Queue();
Queue<Integer> q = new LinkedList<>(); //Array Deque
    q.add(1);
    q.add(2);
    q.add(3);
while(!q.isEmpty()){
  Sop(q.peek());
    q.remove();
}
}
}
```

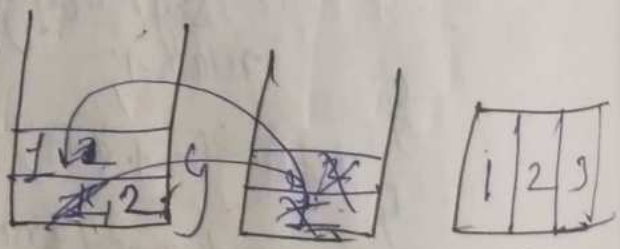
output: 1.
2
3

Question 2

Queue using 2 Stacks → (LIFO)

Push(O(n)) or Pop O(n)
add
remove

remove
peek
add O(1)



① Add
S1 → S2
S1 pop

② S2 → S1

② remove

S1 → pop

Code:
```
import java.util.*;
public class QueueUS{
static Stack<Integer> S1 = new Stack<>();
static Stack<Integer> S2 = new Stack<>();
public static boolean isEmpty(){
    return S1.isEmpty();
}
//add
public static void add(int data){
  while(!S1.isEmpty()){
    S2.push(S1.pop());
  }
  S1.push(data);
  while(!S2.isEmpty()){
    S1.push(S2.pop());
  }}
public static int remove(){
  if(isEmpty()){
    System.out("queue is empty");
    return -1;
  }
    return S1.pop();
}
    //peek.
public static int peek(){
```

```
if (isEmpty()){
System.out.println("queue is Empty");
return -1;
}
return st.pop();
}
}
public static void main(String arg[]){
Queue q = new Queue();
q.add(1);
q.add(2);
q.add(3);
while(!q.isEmpty()){
System.out.println(q.peek());
q.remove();
}
}
}
```

output

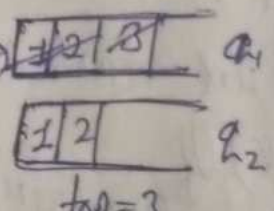
1
2
3

Question 3

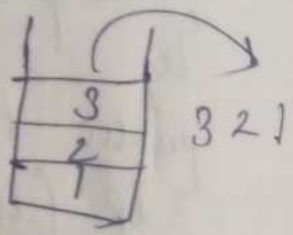
Stacks using 2 Queues:

push O(n)
add - O(1)

pop O(n)



top=3
top=2
top=1



Code:
```
import java.util.*;
public class Queues{
  public class Stacks{
static Queue<Integer> q1 = new LinkedList<>();
  static Queue<Integer> q2 = new LinkedList<>();
public static boolean isEmpty(){
    return q1.isEmpty() && q2.isEmpty();
}
public static void push(int data){
 if(!q1.isEmpty()){
   q1.add(data);
} else{ q2.add(data);
 }
    public static int pop(){
  if(isEmpty()){
     System.out.println("empty stack");
       return -1;
      }
      int top=-1;
  // case1
if(!q1.isEmpty()){
    while(!q1.isEmpty()){
        top = q1.remove();
      if(q1.isEmpty()){
       break;
```

```
    q2.add(top);
}else{ // case2
while(!q1.isEmpty()){
    top=q.removes();
    if(q2.isEmpty()){
        break;
    }q.add(top);
    }
    }return top;
    }
    public static int pop(){
    if(isEmpty()){
    Soy("Empty stack");
    return-1;
}  int top=-1;
// case1
if(!q1.isEmpty()){
while(!q1.isEmpty()){
    top=q.remove();
}  q2.add(top);
}else{//case2//
while(!q2.isEmpty()){
top=q2.remove();
    q1=add(top);
}
```

```
=> return top;
}}
public static void main(){
stack S=new stack();
S.push(1);
S.push(2);
S.push(3);
while(!isEmpty()){
Soy(S.pop());
S.pop();
}}
```

Output

3
2
1

Question 4

character → lower case

First non-repeating letter in a stream of character.

→ aabccxb → a -1 b b b b x

a	a
aa rep	-1
aab	b
aabc	b
aabcc	b
aabccx	b
aabccxb	b x

map / count — 0 1 2 3 4 5 ... 25 — arr — Count[26] = — freq ≥ 1

a	b	c	d	---	z
0	1	2	3		25

Queue :- Front [a | c | c | x | b] Rear

1. "a" → a
2. "aa" → -1
3. "aab" → b
4. "aabc" → b
5. aabcc → b
6. "aabccx → b
7. aabccxb → x

Logic
1. freq[] = [26] a-z
2. Queue<Character>

```
for(int i=0; str.length()){
   ch = str.charAt(i);
   q.add(ch)
   freq[ch-'a']++
}
   q.remove() still find non repeating
   if q.empty() → -1
   freq[q.peek()]=1
```

Code

```
import java.util.*;
public class classroom{
public static void printRepeating(String str){
  int freq[] = new int[26];
  Queue<Character> q = new LinkedList<>();
  for (int i=0; i<str.length(); i++){
      char ch = str.charAt(i);
      q.add(ch);
   freq[ch-'a']++;
  while (!q.isEmpty() && freq[q.peek()-'a']>1){
      q.remove();
  }
  if (q.isEmpty()){
      soy(-1+" ");
  } else{ soy( q.peek()+" ");
  }
  }
      soy();
}
  public static void main( ){
    String str = "aabccxb";
```

Question 5

Interleave 2 Halves of a Queue (even length) O(n)

1 2 3 4 5 | 6 7 8 9 10

1 6 2 7 3 8 4 9 5 10

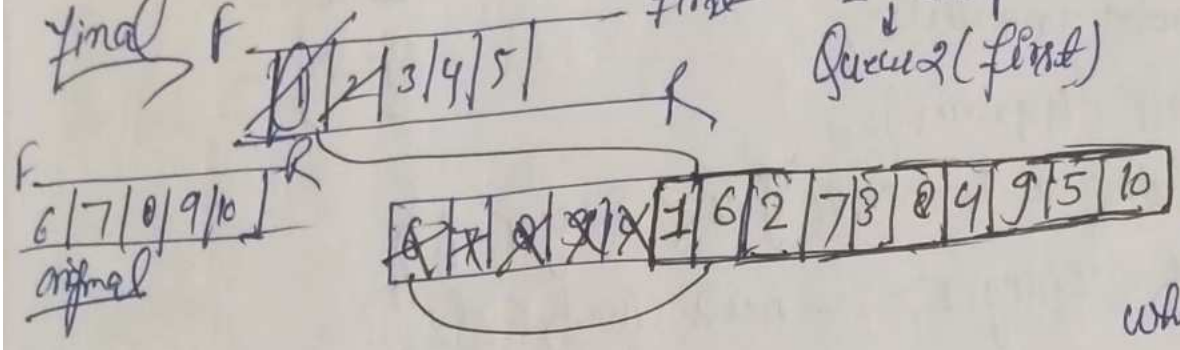
Logic: size = n = 10

size/2 ⇒ 5

1st half

Queue2 (first)

final



① size = q.size()

Que first → size/2

while (first.isEmpty()){

q.add(first.remove())

q.add(q.remove())

```
import java.util.*;
public class classroom{
  public static void interleave(Queue<Integer> q){
    Queue<Integer> firstHalf = new LinkedList<>();
    int size = q.size();
    for(int i=0; i<size/2; i++){
      firstHalf.add(q.remove());
    }
    while(!firstHalf.isEmpty()){
      q.add(firstHalf.remove());
      q.add(q.remove());
    }
  }
  public static void main(String args[]){
    Queue<Integer> q = new LinkedList<>();
    q.add(1);
    q.add(10);
    interleave(q);
    // print q
    while(!q.isEmpty()){
      System.out.print(q.remove()+" ");
    }
    System.out.println();
  }
}
```

output

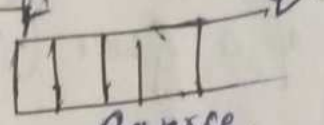
1 6 2 7 3 8 4 9 5 10

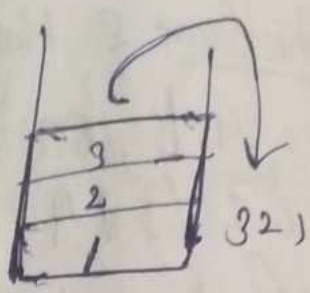
Question 6

Que Reversal: 1 2 3 4 5

Property: final = 5 4 3 2 1

Stacks:-



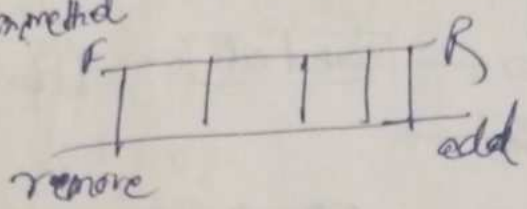


Code:

```
import java.util.*;
public class classroom{
public static void reverse(Queue<Integer>q){
    Stack<Integer>s=new Stack<>();
while(!q.isEmpty()){
    s.push(q.remove());
}
while(!s.isEmpty()){
    q.add(s.pop());
}}
public static void main(String args[]){
Queue<Integer>q=new LinkedList<>();
  q.add(1);
  q.add(5);
   reverse(q);
  //print
while(!q.isEmpty()){
    Sop(q.pop()+" ");
}
    Sop
}
```

Dequeue → next
dchronmethod

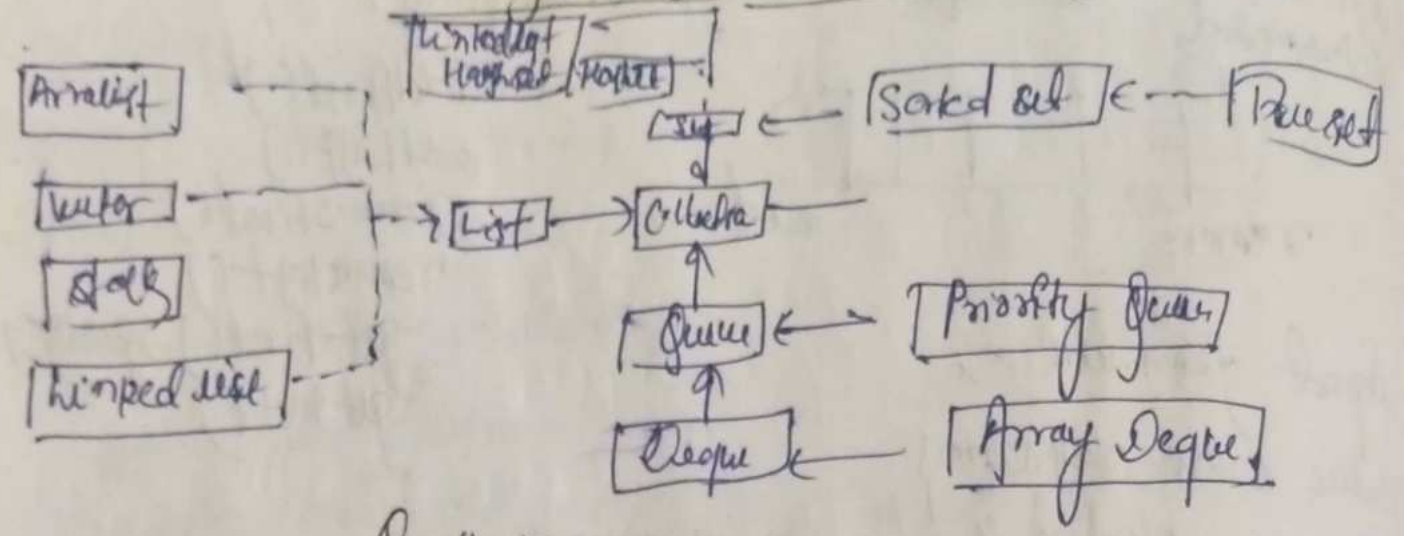
Deque (Double ended queue)



addFirst()
addLast()
removeFirst()
removeLast()
getFirst() peek()
getLast()

```
import java.util.*;
public class classroom{
  public static void main(String args[]){
    Deque<Integer> deque = new LinkedList<>();
      deque.addFirst(1);  // 1
      deque.addFirst(2);  // 2 — 1
    Soy(deque);
deque.removeFirst();
  Soy(deque);

}
```

Java Collection Frame Work

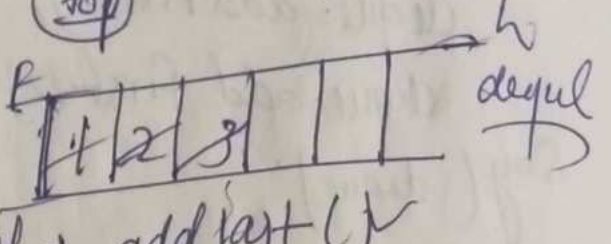


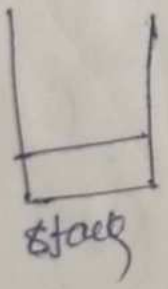
Question 7

Stack & Queue using Deque.

- push O(1) → add
- pop O(1) → remove } O(1)
- peek O(1) → peek

Stack :-





push → addlast() ✓
pop → removelast() ✓
peek → getlast().

Code:

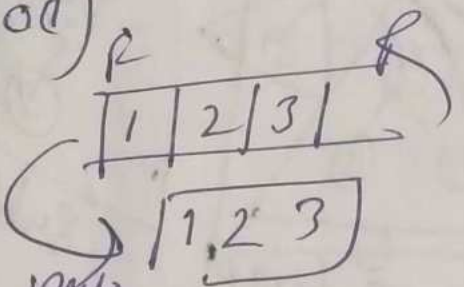
```
import java.util.*;
public class DayRoom{
static class Stack{
Deque<Integer> deque = new LinkedList<>();
public void push(int data){
    deque.addLast(data);
}
public int pop()
    return deque.removeLast();
}
public int peek(){
  return deque.getLast();
}
```

```
public static void main(String arg[]){
  Stack s = new Stack();
   s.push(1);
   s.push(2);
   s.push(3);
Sop("peek is "+s.peek);
   Sop(s.pop());
   Sop(s.pop());
   Sop(s.pop());
}
```

Stack & Queue using Deque

Queue | FIFO

add
remove
peek



deque

add → addLast
remove → removeFirst
peek → getFirst

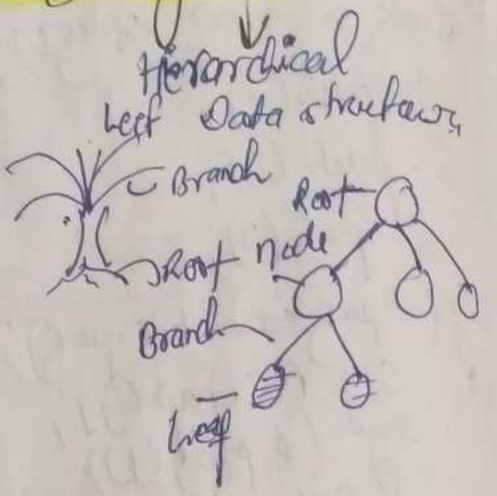
```
Static class Queue{
Deque<Integer> deque = new LinkedList<>();
   deque.public void add(int data){
deque.addLast(data);
}
   public int remove(){
   return deque.removeFirst();
   }
   publi int peek(){
      return deque.getFirst();
}
      psvm(){
```

```
Queue q = new Queue();
  q.add(1);
  q.add(2);
  q.add(3);
Sop("peek = "+q.peek());
  Sop(q.remove());
  Sop(q.remove());
Sop(q.remove());
}}
```

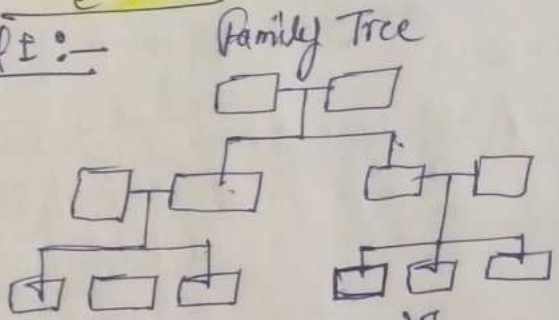
Advanced Data Structure

Binary Tree

Hierarchical
Leaf Data structure

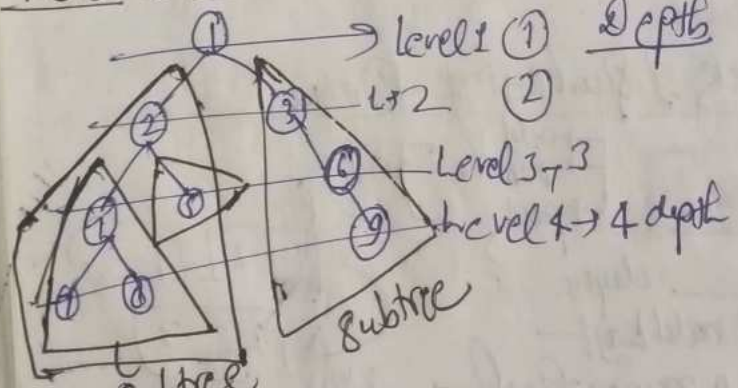


Leaf :-

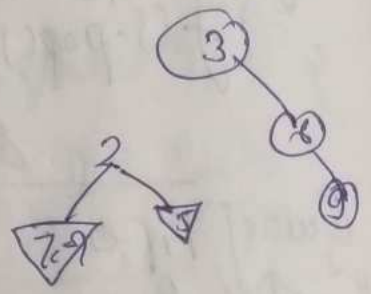


Binary tree [at max 2 children]

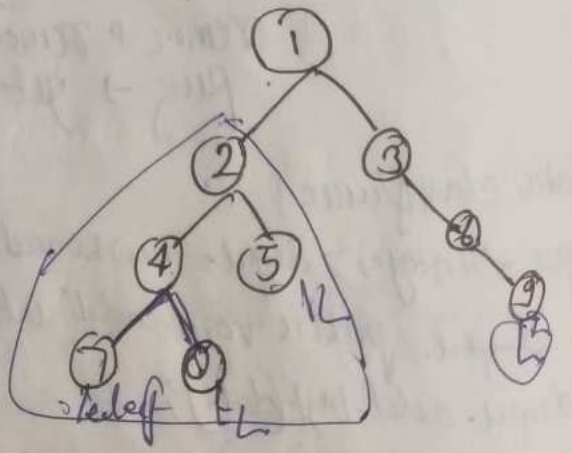
level & subtree in a tree



Depth



Pop Quiz:



① Children of 4: 7, 8
No. of leaves: 4
Parent of 6: 3
level of 2: 2
subtree of 1 of 2: - (2, 4, 5, 7, 8) (3, 6, 9)
Ancestor of 8: { 4, 2, 1 }

Build Tree Pre

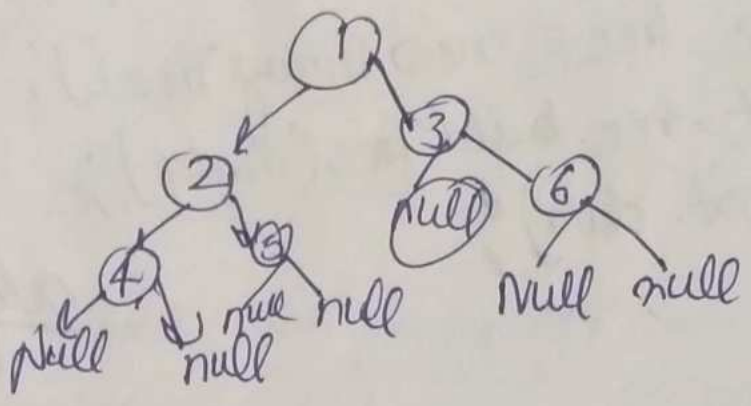
node
left · right

Code of Bin

public cla
stati
int
n
N
Node
this. da
this.
thi
}
sta
publ
if
}
Pu

Build Tree Pre order :- 1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1

node
left . right



Code of Binary tree:-

```
public class BinaryTree {
    static class Node {
        int data;
        Node left;
        Node right;
    Node (int data) {
    this.data = data;
      this.left = null;
      this.right = null;
    }
    static class BinaryTree {
    static int idx = -1;
    public static Node buildTree (int nodes[]) {
          idx++;
    if (nodes[idx] == -1) {
          return null;
      }
        Node newNode = new Node (nodes[idx]);
        newNode.left = buildTree (nodes);
        newNode.right = buildTree (nodes);
    }    return newNode;
    public static void main (String args[]) {
```

```
int nodes[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -}
  BinaryTree tree = new BinaryTree();
  Node root = tree.buildTree(nodes);
    Soy( root.data);
  }}
```

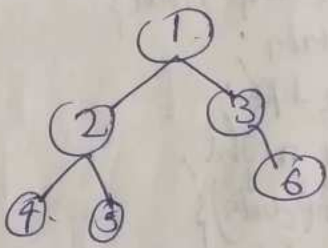
output 1

T.C O(n)

Tree Traversal

(1) Inorder → In between

Rule - (1) Left subtree
(2) Root
(3) Right subtree



print: 4, 2, 5, 1, 3 6

T.C ⇒ O(n).

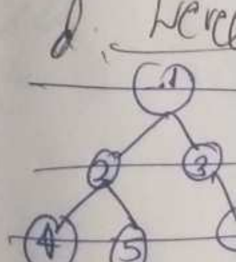
```
public static void inorder(Node root){   Base: if(root == null){
   inorder (root.left);                          return;
   Soy( root.data + " ");                  }
   inorder (root.right);
}}

public static void main (String args[]){
 int node[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};
   BinaryTree tree = new BinaryTree();
   Node root = tree.buildTree(Node);
        tree.inorder(root);
   }}
```

(1) Left Subtr
(2) Right subtr
(3) Root

Code: publ
part
}
}}

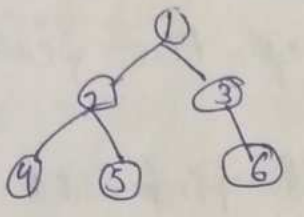
in
Bin
N
}

2. Level



C. Post order Reversals

① left subtree
② Right subtree
③ Root



4 5 2 3 6 3 1

T.C ⇒ O(n)

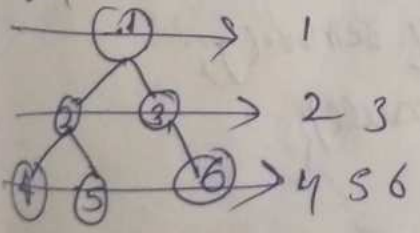
Code:

```
public static void postorder(Node root){
    postorder(root.left);
    postorder(root.right);
    Soy(root.data+" ");
}
public static void main(String arg[]){
int nodes[] = {1,2,4,-1,-1,5,-1,-1,3,-1,6,-1,-1};
BinaryTree tree = new BinaryTree();
Node root = tree.buildTree(nodes);
tree.postorder(root);
}
```

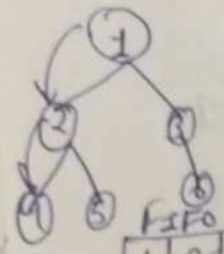
```
if(root==null){
    return
}
```

Output
4 5 2 6 3 1

d. Level order Reversal:-

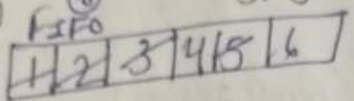


Level order



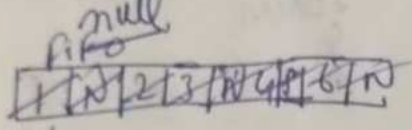
DFS Depth First Search

BFS: Breadth First search → FIFO - Queue



1 2 3 4 5 6 → 1n
2 3n
4 5 6n

Time Complexity: O(n).



1n
2 3n
4 5 6n

Code:

```
// Level order Traversal
import java.util.*;
public static void levelorder (Node root){
  if( root == null){
     return;
  }
    Queue <Node> q = new LinkedList<>();
     q.add(root);
     q.add(null);
while (!q.isEmpty()){
  Node currNode = q.remove();
    if (currNode == null){
        Soy(.);
    if (q.isEmpty()){
       break;
    } else{
      q.add(null);
    } else { Soy(currNode.data + " ");
```

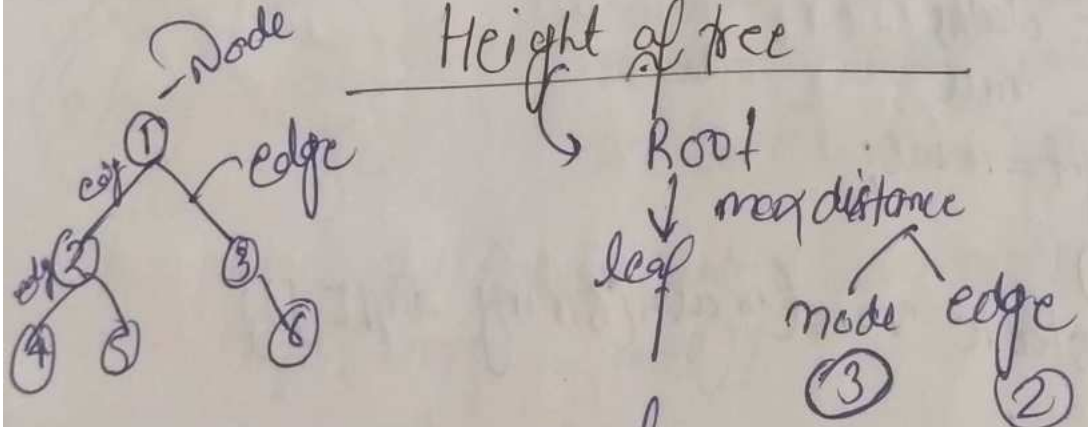
```
if (currNode.left != null){
  q.add(currNode.left);
}
if (currNode.right != null){
  q.add(currNode.right);
}
}
}
public static void main(String args[]){
int node[] = {1,2,3,4,-1,-1,5,-1,-1};
BinaryTree tree = new BinaryTree();
Node root = tree.buildTree(node);
tree.levelOrder(root);
}
}
```

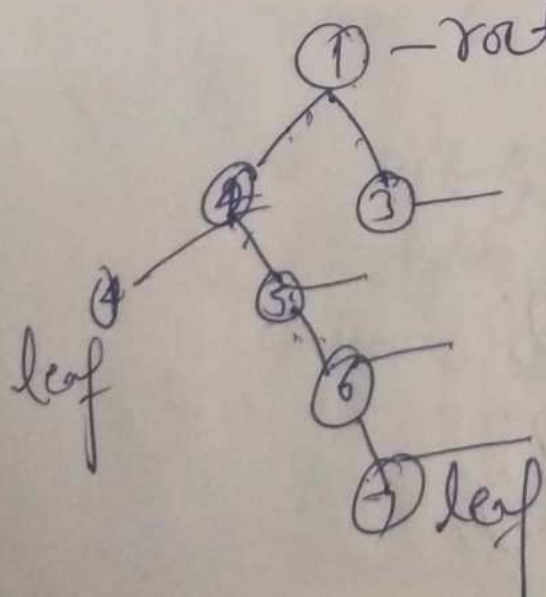
output:-

1

2 3

4 5 6

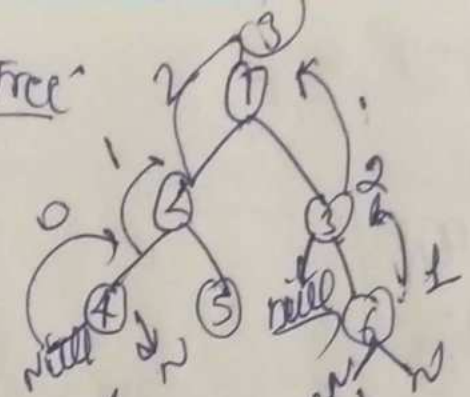




height = 5 in term of Node.

height in term of edge = 4

Height of Tree



① if (root == null)
return 0;

② lh = height (root.left)
rh = height (root.right)

height = max (lh, rh) + 1

⇒ max (lh, rh) + 1

Height = 3

height = 3

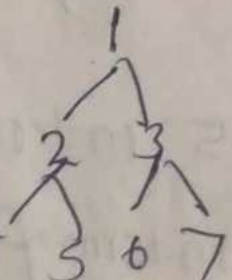
T.C = O(n)

```
public class classroom {
    static class node {
        int data;
        Node left, right;
    public Node (int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

①

```
public static void main (String args[]) {
```

/*



*/

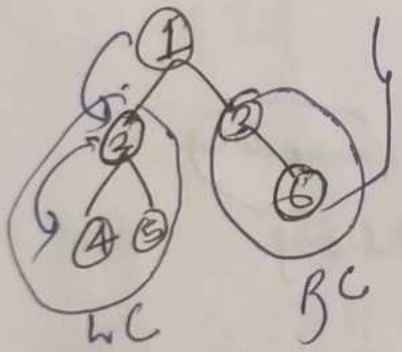
```
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
```

root.left.left = new Node(4);
root.left.right = new Node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);
}}

① public static int height (Node root){
if (root == null){
return 0;
}
int lh = height (root.left);
int rh = height (root.right);
return Math.max (lh, rh) + 1;
} ②

Count of Nodes

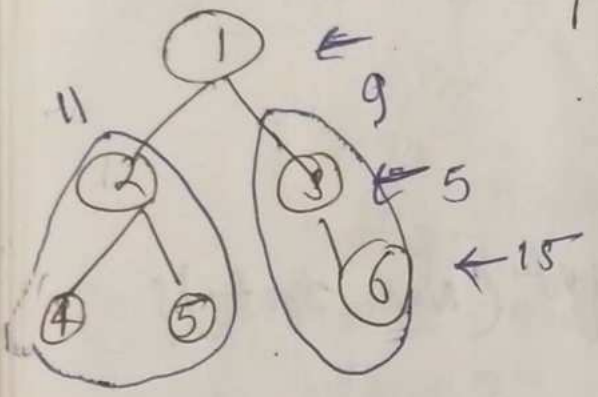
Count = 6



lcount = Count (root.left)
rcount = Count (root.right)
tree count = lc + rc + 1

② public static int count (Node root){
if (root == null){
return 0;
}
int leftCount = Count (root.left);
int rightCount = Count (root.right);
return leftCount + rightCount + 1;
}

Sum of Node

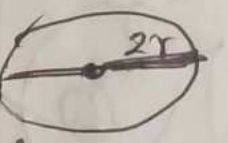


Sum of Node = 21

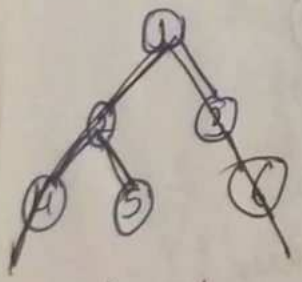
Recursive

```
if (root == null)
    return 0;
leftsum = sum(root.left);
rightsum = sum(root.right);
treesum = ls + rs + root.data;
```

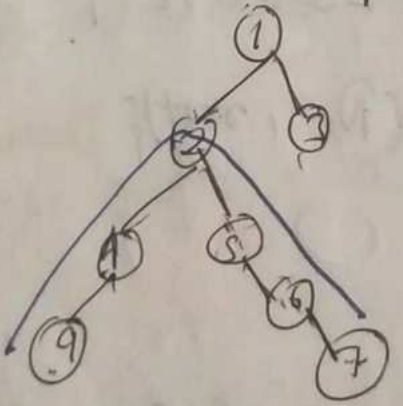
Diameter of a Tree



(no. of largest path b/w 2 leaves)



diameter = 5

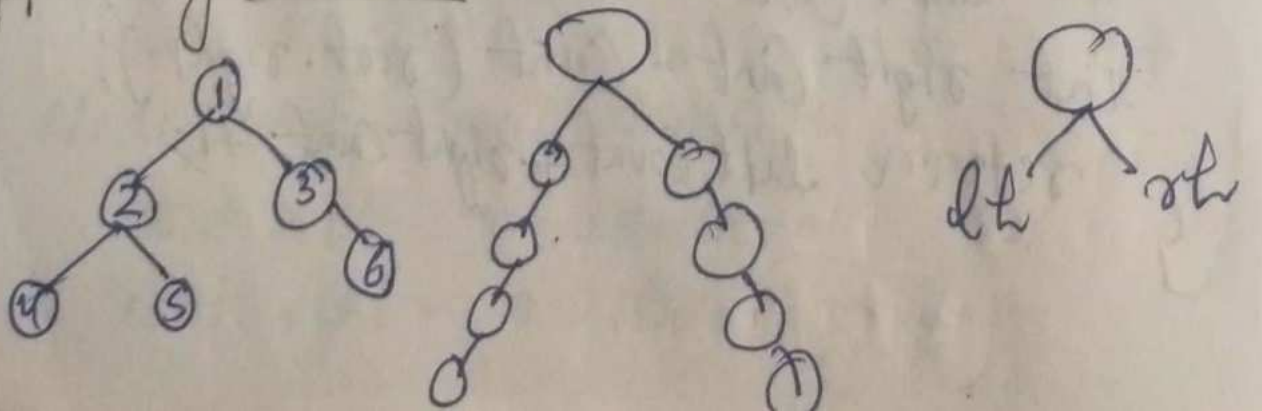


7, and 3 ∄ 6 = no. of node

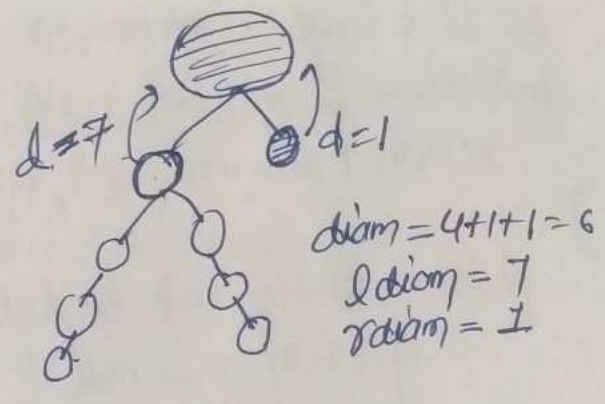
9, 3 ∄ 5

9, 7 ∃ 6

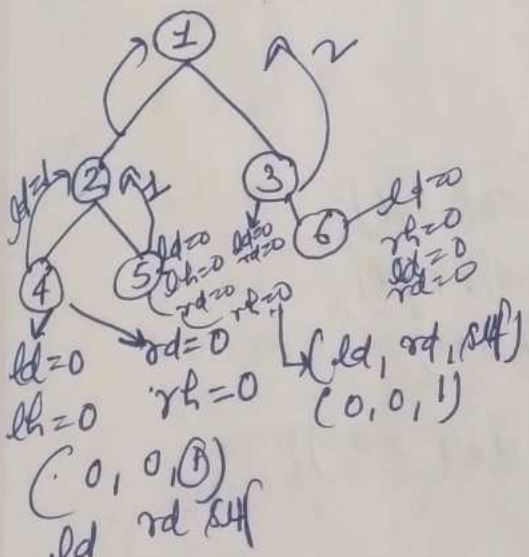
Approach of a tree :- 1 :



// diam pass through root
diam = lh + rh + 1
// diam does not pass
ldiam ✓
rdiam ✓



diam = 4+1+1 = 6
ldiam = 7
rdiam = 1



(ld rd self) self = l+r+1
(0, 0, 1)
(1, 1, 3)
(0, 1, 2)
(3, 2, 5)
d = 5

② ldiam = diam(root.left)
③ rdiam = diam(root.right)
• lh = height(root.left)
• rh = height(root.right)
① self diameter = lh + rh + 1
max(ldiam, rdiam, selfdiam)

```
if(root == null)
    return 0;
```

```
public class classroom {
    static class Node {
        int data;
        Node left, right;
    public Node (int data) {
    this.data = data;
    this.left = null;
    this.right = null;
    }
    public static int height (Node root) {
        if (root == null) {
            return 0;
    }
```

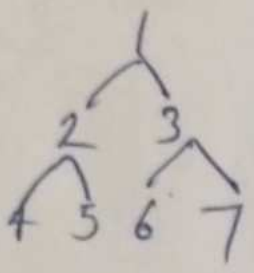
```
int lh = height(root.left);
int rh = height(root.right);
return Math.max(lh, rh)+1;
}

public static int height(Node root){
    if(root == null){
        return 0;
    }
int leftCount = count(root.left);
int rightCount = count(root.right);
return leftCount + rightCount + 1;
}
public static int sum(Node root){
    if(root == null){
        return 0;
    }
int leftSum = sum(root.left);
int rightSum = sum(root.right);
return leftSum + rightSum + root.data;
}
public static int diameter(Node root){
    if(root == null){
        return 0;
    }
    int leftDiam = diameter(root.left);
    int leftHt = height(root.left);
int rightDiam = diameter(root.right);
int rightHt = height(root.right);
```

int selfDiam = leftHt + rightHt + 1;

Math.max(selfDiameter, Math.max(leftDiam, rightDiam));

} public static void main (String args[]){

/*



*/

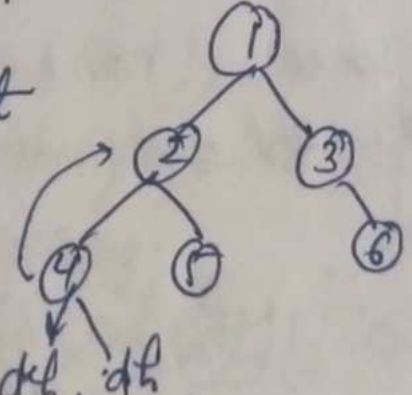
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new Node(4);
root.left.right = new Node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);

}

Diameter Approach 2

return (int) → diameter

return (Info) → diam, ht



class info {
int diam
int ht
}

leftD, lh

rD, rh

max (ld, rd, lh+rh+1)
selfdiameter

height = max(lh, rh)+1

dh, dh

} Same function के अंदर करने वाले है।

Pseudo (diam) → lInfo = diam (root.left)

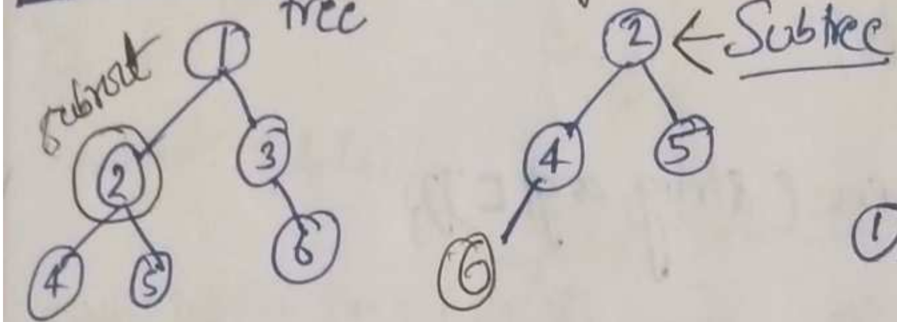
(diam) rInfo = diam (root.right)

final Diam = max[lInfo.d, rInfo.d, lh+rh+1]

finalHt = max(lh, rh) + 1
return newInfo(Diam, ht)

Subtree of another Tree

Given the root of two binary tree root and subroot, return true, if there is a subtree of root with the same structure and node value of subRoot and false otherwise.



Q subtree → tree → true
subtree ↛ tree → false

① root के लिए चेक करना पड़ेगा

Step identical

① Find subroot in tree
② Check identical (subtree, node subtree)

➤ traversal
root.data = subroot.data

Non identical

① node.data != subroot.data
② node = null || subroot = null
③ left subtree → non identical
④ right subtree →
or

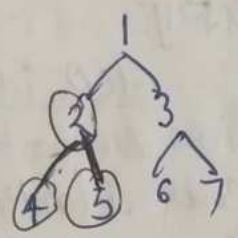
```
if(root==null){
    return false;
}
```

```
public static boolean isSubtree(Node root, Node subroot){
  if (root.data==subRoot.data){
  if (isIdentical(root,subRoot))
      return true;
  }}
boolean leftAns = isSubtree(root.left, subRoot); //ls->node
boolean rightAns = isSubtree(root.right, subRoot);
      return leftAns || rightAns;
return isSubtree(root.left,subRoot)||isSubtree(root.right,subRoot);
}
```

```
① public static boolean isIdentical(Node node, Node subRoot){
    if (node==null && subroot==null){
          return true;
    }
    else if (node == null || subRoot == null){
      return false;
    }}                    //node.data!=subroot.data
```

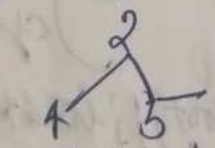
```
if ( !isIdentical(node.left, subRoot.left)){
    return false;
}
if (!isIdentical(node.right, subRoot.right)){
    return false;
}
return true
}
public static void main (String args[]){
```

/*




```
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new Node(4);
root.left.right = new node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);
```

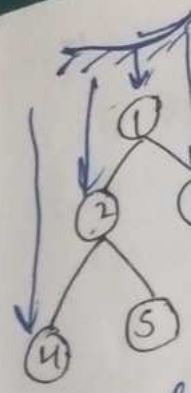
/**




```
Node subroot = new Node(2);
subRoot.left = new Node(4);
subRoot.right = new Node(5);
Soy (isSubtree(root, subroot));
}
```

output

true



output

HashMa

key =

value =

Syntax

① Create:

Add

remov

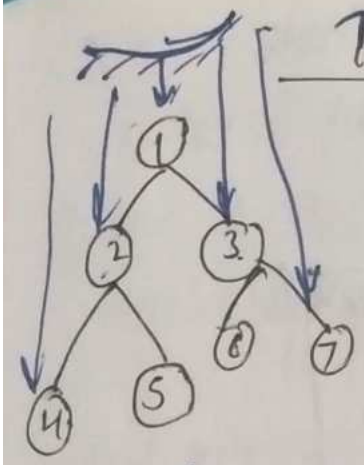
get

② Ad

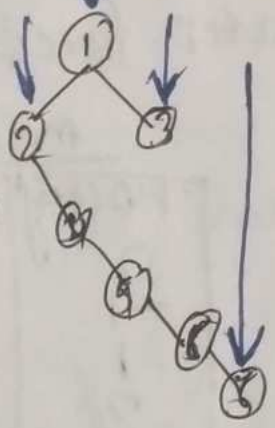
> Horiz

-- ②

Top View of a Tree



output 2,1,3,6



output: 4,2,1,3,7

Top View of Tree

HashMap in Java (Brief) map (Key, Value)

Key = unique Value

Value = Change ~~की~~ ~~सकती~~ है।

Country	Population
China	125
India	100
"US"	50

Key

Syntax `import java.util.*;`

① Create: `Hashmap<String, Integer> map = new Hashmap<>();`

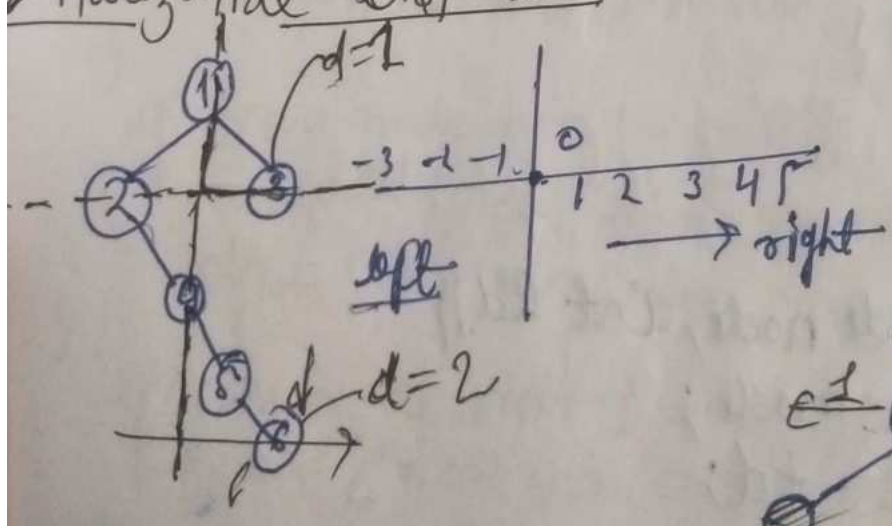
Add T.C O(1) } hashtable (hashing)
remove TC O(1) }
get TC O(1) }

② Add:- (put)
`map.put(Key, Value)`

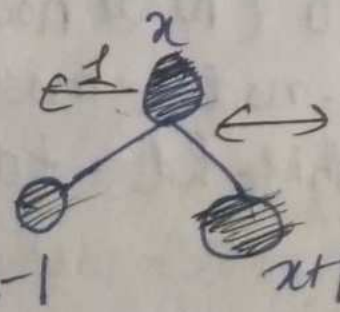
③ Get:-
`map.get(Key)`

Ex `map.put("India", 100)`

Horizontal Distance:-



HD	node
0	1
-1	2
0 1	3
2	6

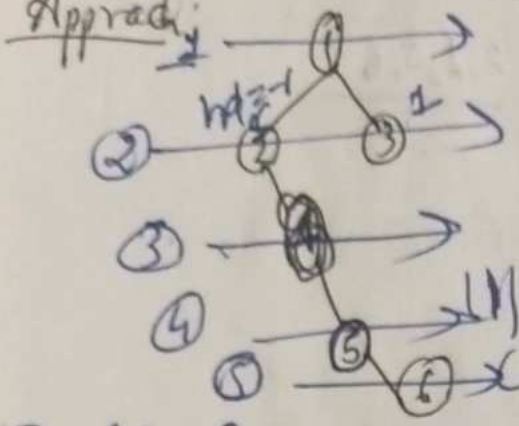


Root HD = 0

Top View of a Tree

(level order traversal)

Approach:



map

HD key	Node val
0	1
-1	2
1	3
2	(6)

① (level order traversal)

min = 0 -1

max = 0 1, 2 map.get(0)

loop (from min to max) value

-1 → map.get(-1)	2
0	1
1	3
2	6

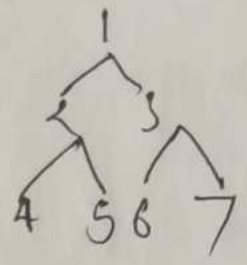
Coding

```
import java.util.*;
Static class Info {
    Node node;
    int hd;

    public Info (Node node, int hd){
        this.node = node;
        this.hd = hd;
    }
}
```

```
Static class Pair {
    Node node;
    int hd;

    public Pair (Node node, int hd){
        this.node = node;
        this.hd = hd;
    }
}
```

```
public static void topview(Node root){
 //level order
Queue <Info> q = new LinkedList<>();
 HashMap <Integer, Node> map = new HashMap<>();

int min=0, max=0;
 q.add(new Info(root,0));
  q.add(null);
while(!q.isEmpty()){
  Info curr= q.remove();
 if (curr==null){
   if(q.isEmpty()){
      break;
  } else{
   q.add(null);
   }
  }
            curr.hd
  if(map.containskey.(key)){ //true.    //first time my hd is occurring
   map.put(curr.hd, curr.node);
 }
  if (curr.node.left != null){
      q.add(new Info(curr.node.left, curr.hd-1));
    min= Math.min(min, curr.hd-1);
 }
 if(curr.node.right != null){
   q.add(new Info(curr.node.right, curr.hd+1));
    max=Math.max(max, curr.hd+1);
 }
```

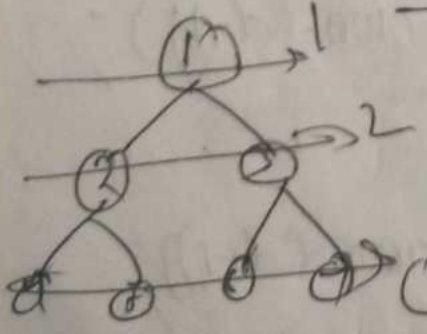
```
}
for (int i=min; i<=max; i++){
    Sop(map.get(i).data+" ");
}
}
Sop();
public static void main(String args[]){
/*
```




```
*/
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new Node(4);
root.left.right = new Node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);
topView(root);
}
}
```

output: 4, 2, 1, 3, 7

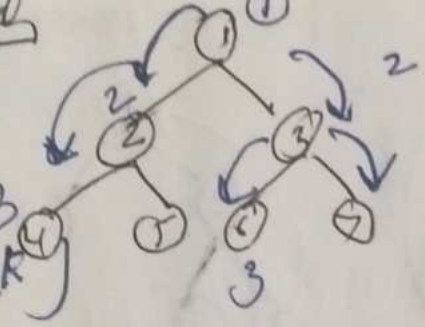
Kth level of a Tree



printallcurrent

k=3
output = 4 5 6 7

① level order Reversal } Approach
② Recursive way }



Preorder
4 5 6 7

Pseudo Code Klevel(root, level, K)
if (level == K){
print(root.data)
}
Klevel(root.left, level+1, K)
Klevel(root.right, level+1, K)
}

T.C = O(N)

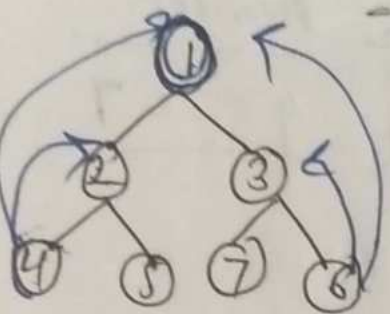
Code: public static void Klevel(Node root, int level, int K){
if(root == null){
return;
}
if(level == K){
Soy(root.data + " ");
return;
}
K.level(root.left, level+1, K);
Klevel(root.right, level+1, K);
public static void main(String args[]){

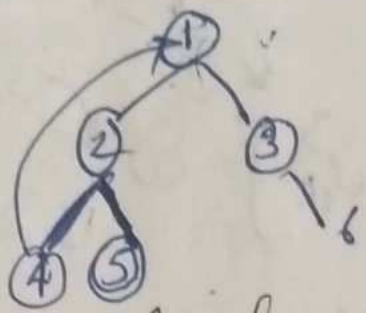
/*

Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new Node(4);
root.left.right = new Node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);
int l5 = 2
Klevel(root, l5)

output - 2, 3

Lowest Common Ancestor





$n_1 = 4$ $n_2 = 5$

LCA = 2
(Common Node)

$n_1 = 4$ $n_2 = 6$

LCA = 1

Approach

path 1 (n_1) = [1 | 2 | 4]

path 2 (n_2) : [1 | 3 | 6]

LCA = 1

$n_1 = 4$ $n_2 = 5$

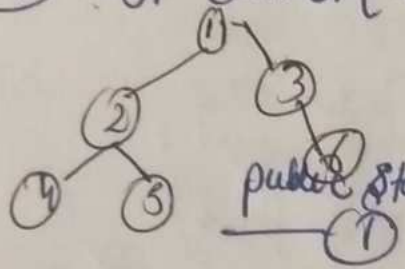
path 1 = [1 | 2 | 4]

path 2 = [1 | 2 | 5]

LCA = 2

(1) Root

(2) Last common node → LCA



[1 | 2 | 5]

Step (1) Root to Node

getpath (root, node A, L path)

[1 | 2 | 5]

(2) Loop (i = 0 to path.length || path2.length)

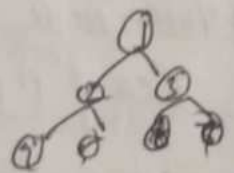
(Last common Ancestor)

Code:

```
public static Node LCA (Node root, int n1, int n2){
    Arraylist <Integer> path1 = new Arraylist <>();
    Arraylist <Integer> path2 = new Arraylist <>();
    getPath (root, n1, path1);
    getPath (root, n2, path2);
    // last common Ancestor
    int i = 0
    for (; i < path1.size() && path2.size(); i++){
        if (path1.get(i) != path2.get()){
            break;
        }
    }
    // Last equal node -> i-1
    Node Lca = path1.get(i-1);
}
```

① public static boolean getPath(Node root, int n, ArrayList<Node> path){
path.add(root);
if(root.data == n){
return true;
}
boolean foundLeft = getPath(root.left, n, path);
boolean foundRight = getPath(root.right, n, path);
if(foundLeft || foundRight){
return true;
}
path.remove(path.size()-1);
return false;
}
public static void main(String args[]){

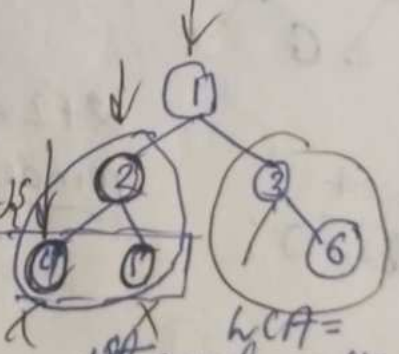
if(root == null){
return false;
}



int n1=4, n2=5;
System.out.println(lca(root, n1, n2).data);

Lowest Ancestor : Approach 2

public static Node lca2(Node root, int n1, int n2){
if(root.data == n1 || root.data == n2){
return root;
}
Node leftLca = lca2(root.left, n1, n2);
Node rightLca = lca2(root.right, n1, n2);
if(rightLca == null){
return leftLca;
}
if(leftLca == null){
return rightLca;
}
return root;



LCA = 1st parent → subtree n1 & n2
ancestor

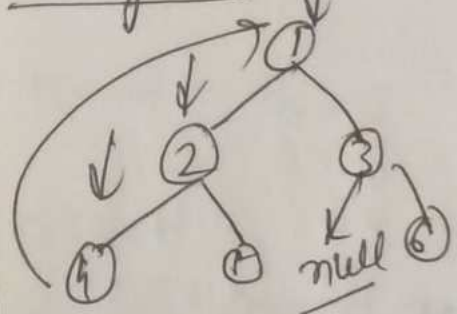
public static void main (String Ar[])

int $n_1 = 4$, $n_2 = 7$;

Sop(lca (root, n_1, n_2).data);

$n_1 = 4$ $n_2 = 6$

Dry Run:-



Condition ① root = null
root = n1
root = n1

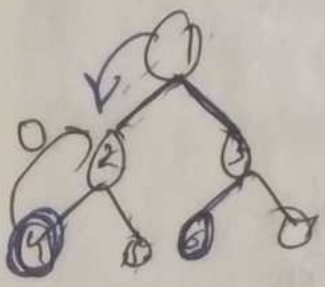
left LCA
right lca.

② right LCA = null → left lca
left LCA = null → right LCA
right left = null → return root

Minimum Distance between Node

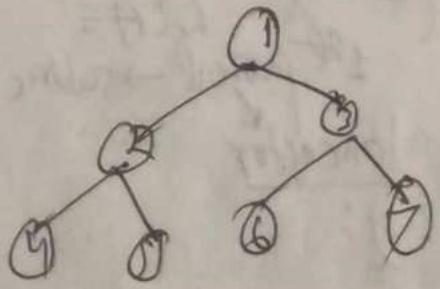
$n_1 = 4$ $n_2 = 6$

dist = 4 (node) (edge)



2+2=4
mindist = 4

$n_1 = 4$
return 0;



lca of n_1 & n_2

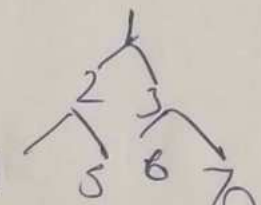
Approach ① dist1 lca to n1
② dist 2 lca to 2
dist1 + dist2 = min. dist n1 n2

if (root.data = n)
return 1
if (root.data = n)
return 0.
left n search
right n search
-1, -1, → -1
Valid (>-1) → Value+1

```
public static Node Lca2(Node root, int n1, int n2){
if (root == null || root.data == n1 || root.data == n2){
  return root;
}
  Node leftLca = Lca2(root.left, n1, n2);
  Node rightLca = Lca2(root.right, n1, n2);
  //
If (leftLCA == null){
 return rightLCA;
}
   return root;
}
public static int LcaDist(Node root, int n){
   if (root == null){
   }  return -1;
    if (root.data == n){
    }  return 0;
int leftdist = LcaDist(root.left, n);
int rightDist = LcaDist(root.right, n)
if (leftDist == -1 && rightDist == -1){
}      return -1;
 else if (leftDist == -1){
      return rightDist + 1;
  } else{
    }      return leftDist + 1;
  }
```

```
public static int minDist(Node root, int n1, int n2){
    Node lca = lca2(root, n1, n2);
    int dist1 = lcaDist(lca, n1);
    int dist2 = lcaDist(lca, n2);
    return dist1 + dist2;
}
public static void main(String args[]){
```

/*

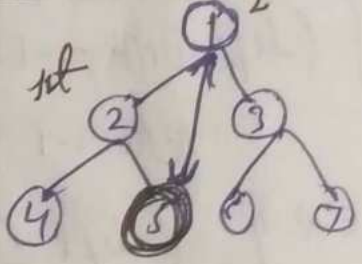



```
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new node(4);
root.left.right = new node(5);
root.right.left = new Node(6);
root.right.right = new Node(7);
int n1=4, n2=6;
Sop(minDist(root, n1, n2));
}}
```

Kth Ancestor of Node

node = 5 k=2

ans = 1

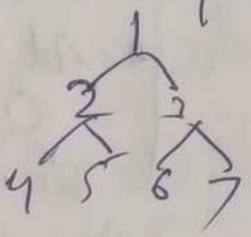


Approach:- ① find my node
② if (root.data = node)
return 0.

left Dist
right Dist
-1, +1 → -1

valid value → (dist + 1) == k print root.data

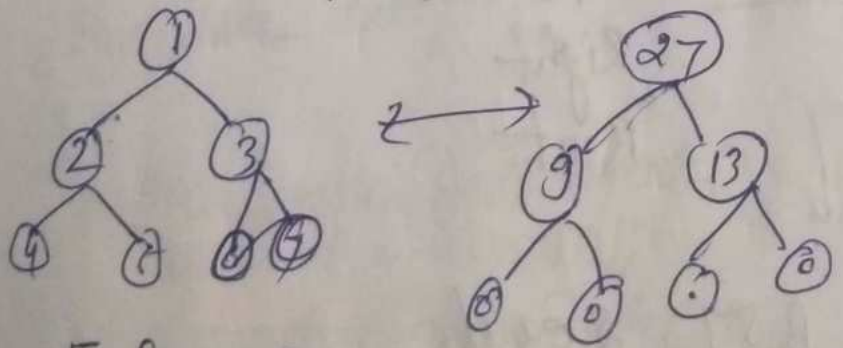
```
public static int kAncestor(Node root, int n, int k){
  if (root.data == n){
    return 0;
  }
  int leftDist = kAncestor(root.left, n, k);
  int rightDist = kAncestor(root.right, n, k);
  if (leftDist == -1 && rightDist == -1){
    return -1;
  }
  int max = Math.max(leftDist, rightDist);
  if (max+1 == k){
    sop(root.data);
  }
  return max+1;
}
public static void main(String arg[]){
  /*
```

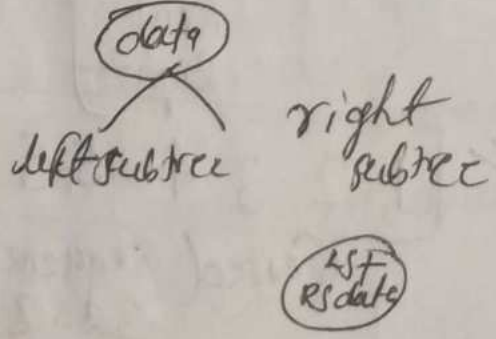


output = 1

```
  int n=5, k=2;
  kAncestor(root, k, n);
}}
```

31.6 Transform to Sum Tree





Each node = sum of left & right subtree

Approach

left subtree sum =
right subtree sum =
data = root.data
root.data = lsum + RS
return data;

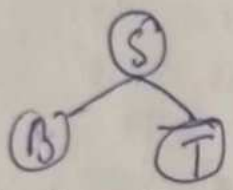
or

if (root == null)
return 0

leftchild = transform(root.left)
rightchild = transform(root.right)
data = root.data
root.data = root.left + left child + root.right child
return data;

32.1 Binary Search Tree (BST)

* fast lookup → O(H) - height
O(N) no. of nodes

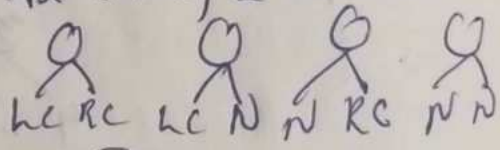


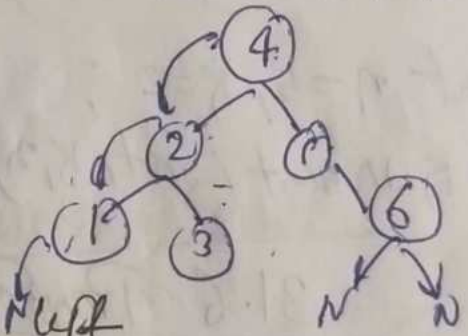
*

What is a BST?

ⓐ Left Subtree Node < Root
ⓑ Right Subtree Node > Root
ⓒ Left & Right Subtree are also BST with no duplicate.

Binary Tree
- At most 2 children





Key greater

Preorder → Root Left Right

* InOrder → Left Root Right

Postorder → Left Right Root

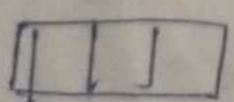
1 2 3 4 5 6

Sorted Sequence

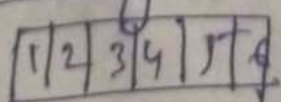
32.2 BST Search

Key = 3

Linear



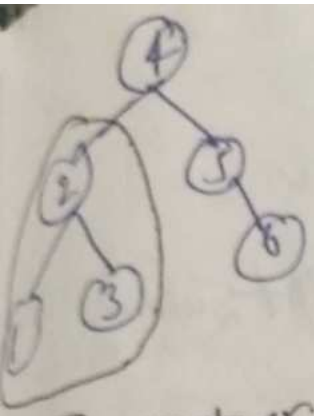
Binary



Key = 6

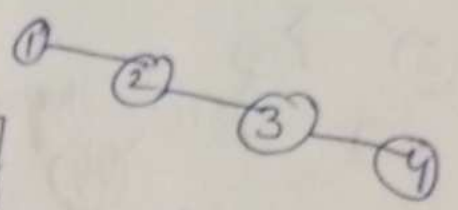
K < data → go left
Key = 6 = data
return data

K > data → go right



Key=3 (T.C → O(H))

*Skew Tree



O(H) = O(N)

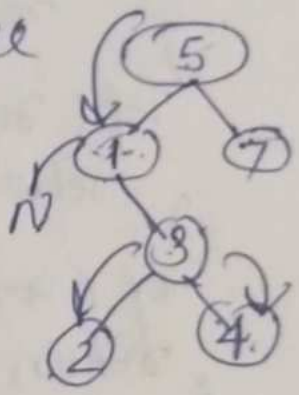
Remember: BST make search efficient

Strategy: Most problem will be solved using recursion ie. by dividing into subproblem & making recursive call on subtree.

32.3 Build a BST

Value[] = {5, 1, 3, 4, 2, 7}

Root = null



1 2 3 4 5 7

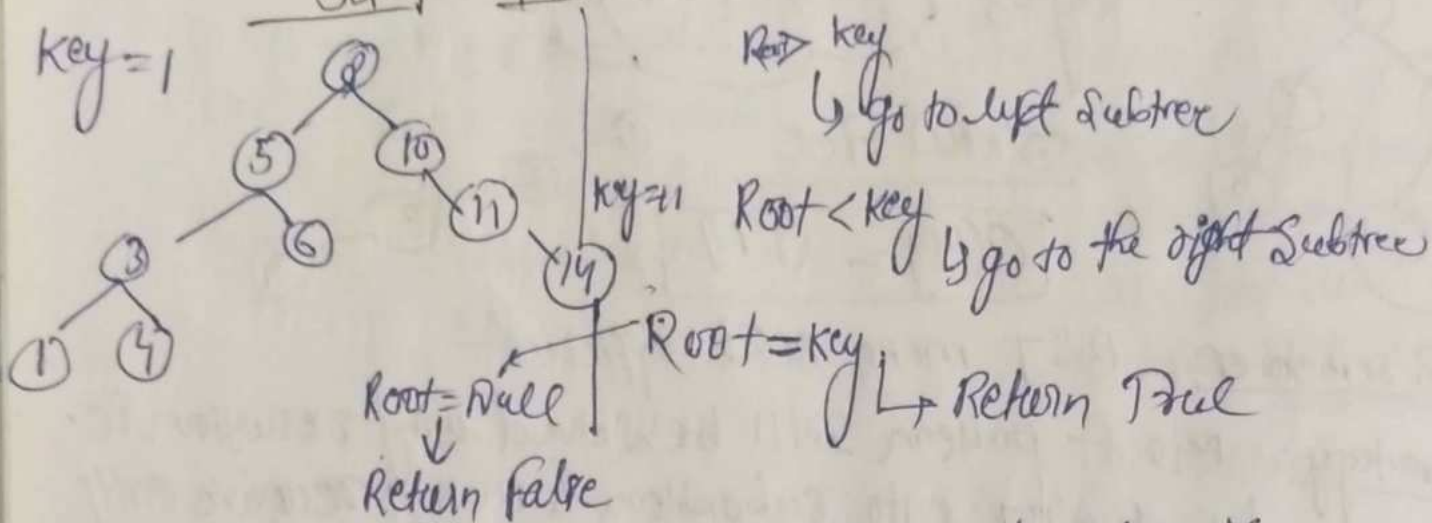
value[i] > root → Right Subtree

value[i] < root → left Subtree

Code:

```
public class BST{
  static class node{
    int data;
    Node left;
    Node right;
  Node(int data){
  this.data = data;
}}

public static Node insert(){
  if(root == null){
    root = new Node(val);
    return root;
  }
  if(root.data > val){
  // left subtree
root.left = insert(root.left, val);
  }else{ // right subtree
root.right = insert(root.right, val);
  } return root;
  }
  public static void main(String[]
int value[] = {5,1,3,4,2,7}
  Node root = null
  for(int i=0; i<value.length; i++)
root = insert(root, value[i]);
  }
```

32.4 Search a BST



Code:

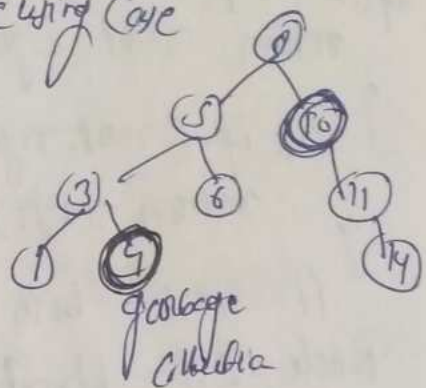
```
public static boolean search (Node root, int key){
    if (root == null){
        return false;
    }
    if (root.data == key){
        return true;
    }
    if (root.data >= key){
        return search (root.left, key);
    }
    else{
        return search (root.right, key);
    }
}
public static void main (String arg[]){
int value[] = {5, 1, 3, 4, 2, 7};
  Node root = null;
for (int i=0; i<value.length; i++){
    root = insert (root, value[i]);
    inorder (root);
    Soy ();
if (search (root, 1)){
    Soy ("Found");
} else{
    Soy ("Not Found");
}
```

output: Found

32.5 Delete a Node

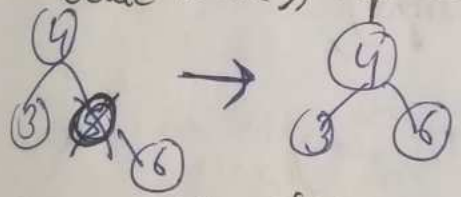
Case ① No child (leaf Node)
② One child (10, 11)
③ Two children (8, 3, 5)

① Search
② Delete using Case



Case1: No child (Leaf Node) Key=4
Delete Node
& Return Null to parent

Case2 One child
Delete Node & replace with child node



Case3 Two children
- Replace value with inorder successor
- Delete the node for inorder successor

inorder successor in BST ⇒ left most node in Right Subtree

In Order successor always has 0 or 1 child.

Approach ① Find inOrder successor
② Node ka value replace
③ Delete कर दो

```
public static Node Delete (Node root, int val){
    if (root.data < val){
        root.right = delete (root.right, val);
    } else if (root.data > val){
        root.left = delete (root.left, val);
    } else {  // voila Case 1
        // Case 2 - leaf node
        if (root.left == null && root.right == null){
            return null;
```

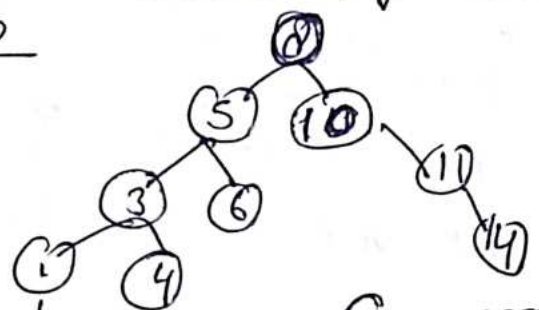
13 4 6 D—
134 6 D :—

```
// Case - Single Child
if(root.left == null){
    return root.right;
} else if (root.right == null){
    return root.left;
}
// Case 3 - both children
Node DS = findInOrderSuccessor(root.right);
root.data = DS.data;
root.right = delete(root.right, DS.data);
return root;
}
public static Node findInorderSuccessor(Node root){
while (root.left != null){
    root = root.left;
}
return root;
}
public static void main(String args[]){
int value[] = {8,5,3,6,4,6,10,11,14};
    Node root = null;
for (int i=0; i<value.length; i++){
    root = insert(root, value[i]);
}
inorder(root);
say();
    root = delete(root, 1);
    say();
    inorder(root);
}}
```

32.6 Print in Range

k1 = 5 k2 = 12

5, 6, 8, 10, 11



~~root~~ =

Case I k1 <= root <= k2

↳ LS

↳ RS

C3 ⇒ root < k1

↳ Left Recursive (left Subtree)

C2 root > k2

↳ RS

Code

```
public static void printrange(Node root, int k1, int k2){
if (root.data >= k1 && root.data <= k2){
printInRange(root.left, k1, k2);
Sop(root.data + " ");
printInRange(root.right, k1, k2);
}
else if (root.data < k1){
printInRange(root.left, k1, k2);
}else{
printInRange(root.right, k1, k2);
}}
public static void main(String args[]){
int value[] = {8,5,3,1,4,6,10,11,14};
Node root = null;
for(int i=0; i<value.length; i++){
root = Insert(root, value[i]);
}
inorder(root);
Sop();
printInRange(root, 5, 12);
}}
```

```
if(root == null){
return;
}
```

output

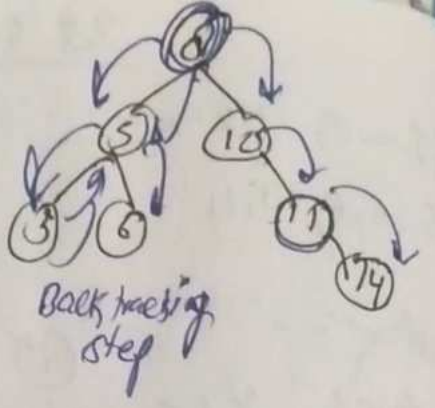
5, 6, 8, 10, 11

3d:] Root to Leaf Path

ArrayList → path

0	5	3			

P1 { 0-5-3
P2 { 0-5-6
P3 { 0-10-11-14



Backtracking step

① Add (Node) to path
② Left Subtree
③ Leaf → print path; Right Subtree

{preorder Reversal

Base Case

Code:

```
public static void printRoot2Leaf(Node root, ArrayList<Integer> path){
    path.add(root.data);
    printRoot2Leaf(root.left, path);
    printRoot2Leaf(root.right, path);
    path.remove(path.size()-1);
}

if(root==null){
    return;
}
path.add(root.data);
if(root.left==null && root.right==null){
    printPath(path);
}

public static void main(String args[]){
    int values[] = {0,5,3,1,4,6,10,11,14};
    Node root = null;
    for(int i=0; i<values.length; i++){
        root = Insert(root, values[i]);
    }
    Inorder(root);
    Sop();
    printRoot2Leaf(root, new ArrayList<>());
}
```

32.8 Validate BST

Approach

Compare with left & right

small → valid BST ✓

Root

Left Right

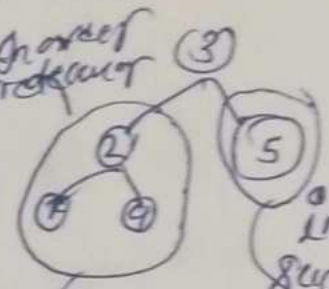
large. ~~राईट~~ Right side ~~में~~ ~~चाहिए~~

Approach Check if max value in left subtree < node
and min value in right subtree > node

Inorder Traversal

↓

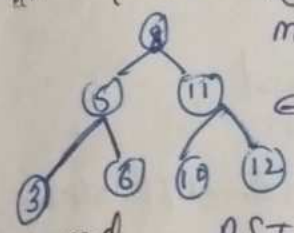
Sorted



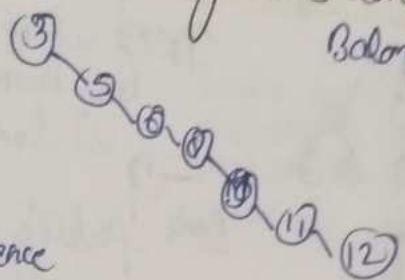
$[-\infty < \text{val} < 5 \quad 5 < \text{val} < \infty]$

33.1 Sorted array to Balanced BST

arr = {3, 5, 6, 8, 10, 11, 12} (start, end, mid)



← Height



Balanced min possible

Approach BST Inorder sequence → Sorted

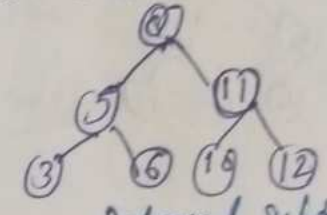
{Left subtree Node + Curr/root node + Right Subtree node}

arr = {3, 5, 6 | 8 | 10, 11, 12} mid = (start+end)/2

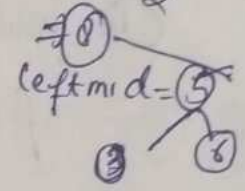
(Left | mid | Right)

mid = (start+end)/2

mid = Current node



Balanced Subtree



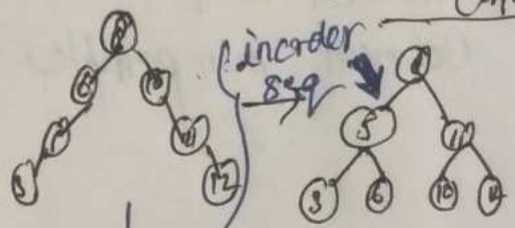
leftmid = 5

```
Node createBST(arr, st, end)
    if (str > end)
        return null
    int mid = (str+end)/2
    Node root = new Node(arr[mid])
    root.left = createBST(arr, str, mid-1);
    root.right = createBST(arr, mid+1, end);
    return root
```

T.C = O(n).

Linear

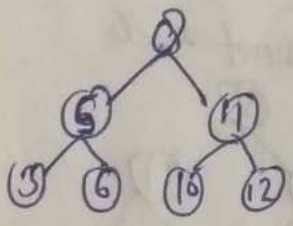
33.2 Convert BST to Balanced BST



Inorder Sequence :- 3, 5, 6, 8, 10, 11, 12

root root root

mid



Approach Sorted → Balanced BST

① Inorder Sequence

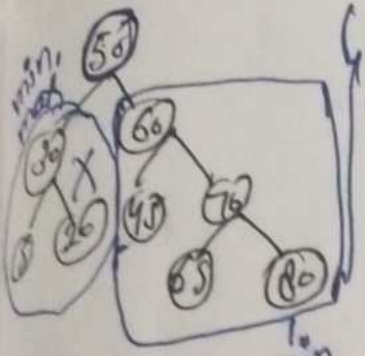
② Sorted arr → balanced BST

Code

```
public static Node balanceBT(Node root,
// inorder seq
ArrayList<Integer> inorder = new Arra
getInorder(root, inorder);
```

```
public static void getInorder(Node root, Arr
                                <Integer> ino
if (root == null) {
    return
getInorder(root.left, inorder)
inorder.add(root.data);
getInorder(root.right, inorder
}
```

33.3 Size of Largest BST in BT



BT → is valid BST

min = 45
max = 50

root == null → True ✓
min → min possible value
max → max possible value

root < max(left) || (root > min, right)

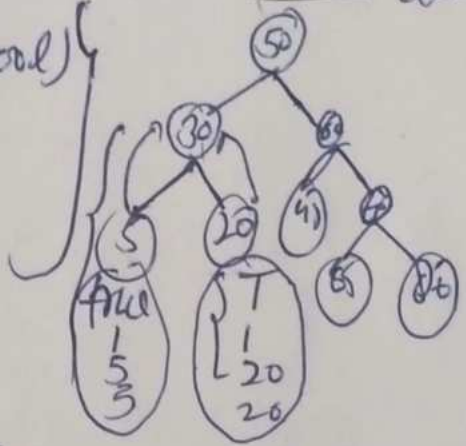
false.

root.data > left && root.data

True → isBST = true, size = 0, min = +∞, max = -∞

null

Approach:
1. isBST (bool)
2. size
3. min
4. max



① Invalid BST
↳ Root = null
left max > root
right min < root
⇒ false = isBST

left info
right info
self info
↓
max size

size = left.s + right.s + 1
min = min (root.data, leftmin, rightmin)

Graph

- Introduction to graph
- Graph Terminology
- Graph Implementation
- graph Traversal
- Weighted And Directed graph
- Minimum Spanning tree
- Cycle detection in graph
- Kruskal Algorithm
- Prim algorithm
- Dijakstra algorithm
- Bellman Ford Algorithm & Related Question

Dynamic Tree

- Segment Tree
  - (1) what are segment tree
  - Creation of segment tree
  - Solving range Query

Graph

- Introduction to graph
- Graph Terminology
- Graph Implementation
- graph Traversal
- Weighted And Directed graph
- Minimum Spanning tree
- Cycle detection in graph
- Krushal Algorithm
- Prim algorithm
- Dijakstra algorithm
- Belmen Ford Algorithm & Rotf Question

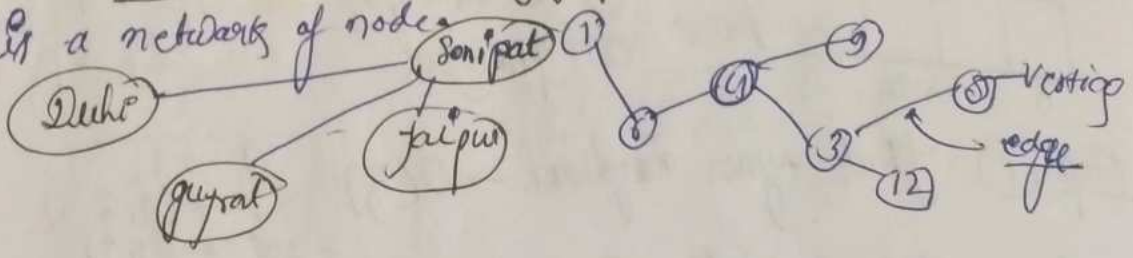
Dynamic Tree

- Segment Tree
  - ① what are segment tree
  - Creation of segment tree
  - Solving range Query

37.1 Introduction of graph

> Graph is a network of node

Example



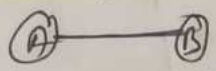
37.2 Type of graph

Edge

① Uni-Directional

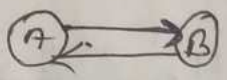
* Directed graph

② Un-Directed



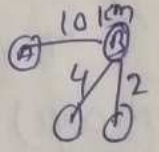
[Undirected graph]

③ Bi directional

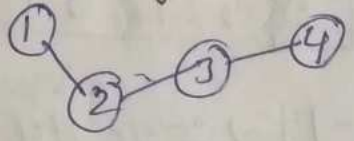


> Weight: property Associate

↳ 10



② Non Weighted



37.3 Storing a Graph → Structure Represent

Ex Linked list

> Adjacency list
> Adjacency Matrix
> Edge List
> 2D matrix (Implicit graph)

Adjacency list:-

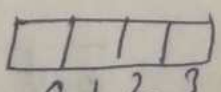
V = 4 | undirected graph
E = 4 | unweighted graph

cycle detection

- List of list

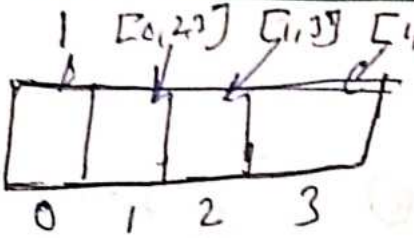
List 0 0 → (1)
List 1 1 → (0, 2, 3)
List 2 2 → (1, 3)
List 3 3 → (1, 2)

→ ArrayList <ArrayList>



Array <ArrayList>

HashMap < Int, List >
& Vertex (key) Value



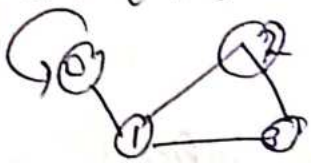
① extra info
② optimized

Vertex = 4
E = 4

undirected
unweighted

Step ① find your neighbour → O(E)

0 → 1
1 → (0, 2, 3)
2 → (1, 3)
3 → (1, 2)



Array of Arraylist [Edge]
src. destination → wt

② Adjacency matrix V = 4 E = 4 O(V²)

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

(i, j) → 1 (i-j)
0 (i-j → No edge)

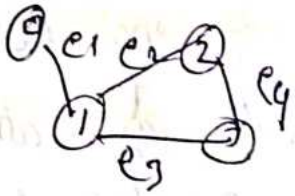
V = i
2

O(V)
n²

(2-1) → edge ya ⇒ 1
(3-3) No edge → 0

Edge list

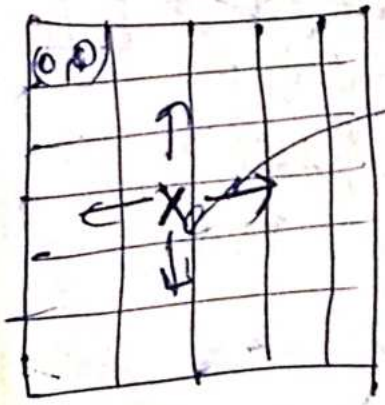
Edge = {{0,1}, {1,2}, {1,3}, {2,3}}
e1 e2 e3 e4



{0, 1, 4} {1, 2, 2} {1, 3, 3} {2, 3, -1}
e1 e2 e3 e4

{edge → Sorting}
{minimum spanning tree (MST)}

Implicit Graph



(i, j)
(i-j-1) ← (i, j) → (i, j+1)
(i+1, j)

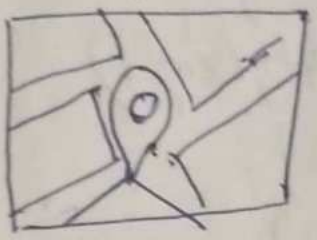
graph (n+1)*2D matrix

{Flood Fill Algorithm}

2D matrix

Application of graph

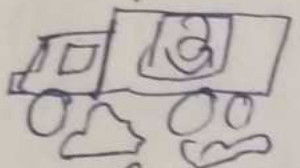
① Maps (Sortest path)



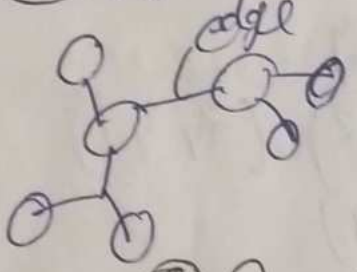
② Social Network

in f X Quora

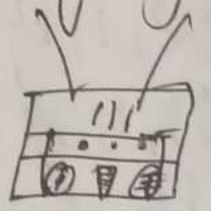
③ Delivery Network (Shortest Cyclic Route)



④ Physics & Chemistry

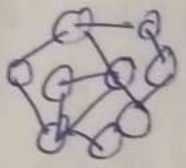


⑤ Routing Algorithm

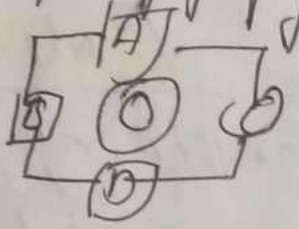


⑥ Machine Learning:

Computational graph

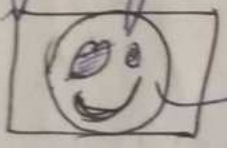


⑦ Dependency graphy



⑧ Computer vision

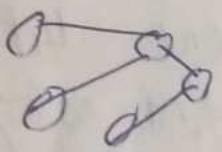
• Image segmentation AI/ML



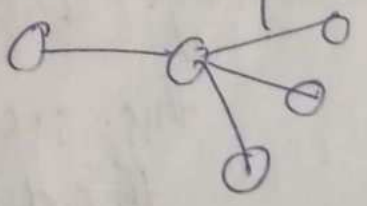
Flood fill Algorithm

(बाल्टी) इसमें को Color fill करें

⑨ Graph Database: Nebula neo4j

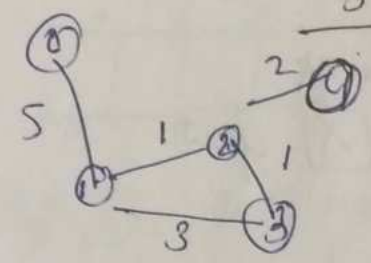


⑩ Research:- Neuron (human)



{Fraud Detection

37.5 Creating a graph (Adjacency list)



ArrayList<Edge> graph

Edge = (src, dest, weight)

→ (2, 4, 2)

(4, 2, 2)

0 → {0, 1, 5}

1 → {1, 0, 5}

2 → {2, 1, 1} {2, 3, 3}

2 → {2, 1, 1} {2, 4, 2} {2, 5, 1}

3 → {3, 1, 3} {3, 2, 1}

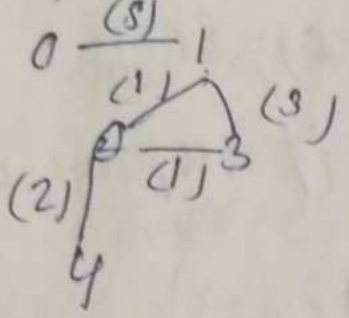
4 → {4, 2, 2}.

Code.

```
import java.util.*;
public class classroom {
    static class Edge {
        int src;
        int dest;
        int wt;
    public Edge(int s, int d, int w){
        this.src = s;
        this.dest = d;
        this.wt = w;
    }
}
```

public static void main (String args[]){

/*



*/

int V = 5

ArrayList<Edge>[] graph = new ArrayList[V]; //null

for(int i=0; i<V; i++){

graph[i] = new ArrayList<>();

}

// 0 - vertex

graph[0].add(new Edge(s: 0, d: 1, w: 5));

// 1 vertex

graph[1].add(new Edge(s: 1, d: 2, w: 2));

graph[1].add(new Edge(s: 1, d: 2, w: 1));

graph[1].add(new Edge(s: 1, d: 3 w: 3));

// vertex 2:

graph[2].add(new Edge(s: 2, d: 1, w: 1));

graph[2].add(new Edge(s: 2, d: 3, w: 1));

graph[2].add(new Edge(s: 2, d: 4; 4));

// 3 vertex

graph[3].add(new Edge(s: 3, d: 1, w: 3));

graph[3].add(new Edge(s: 3, d: 2, w: 1));

// 4 vertex

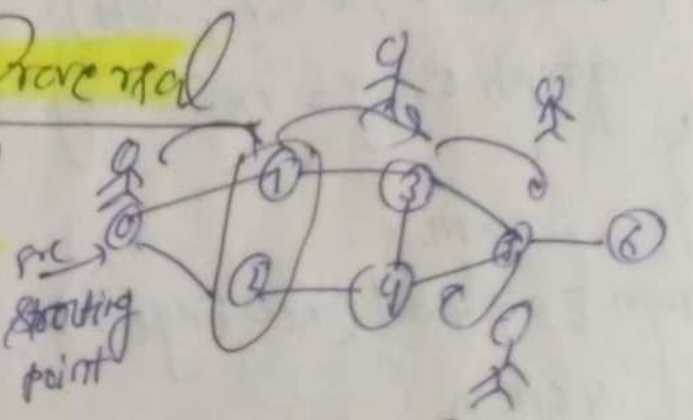
graph[4].add(new Edge(s: 4, d: 2, w: 4));

```
//2's neighbour.
for(int e=0; i<graph[2].size(); i++){
    Edge e= graph[2].get(i); // src, dst, wt
    System.out.println(e.dst);
}
//
```

output: 1 3 4

Graph Traversal

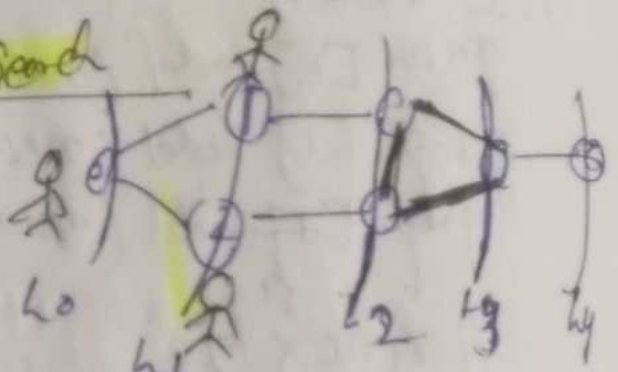
① Breadth first search (BFS)
② Depth first search (DFS)



(level order traversal)

Breadth First Search

- go to immediate neighbors first



Queue: [0 | 1 | 2 | 3 | 4 | 1 | 4 | 5 | 2 | 3 | 5]

FIFO

```
q.remove() → curr
visit[curr] = print(curr)
if(!vis[curr])
```

T	T	T	T	T	T	T	
visited 0	1	2	3	4	5	6	7

visited → each node visit once

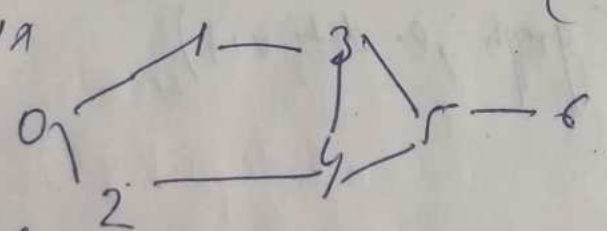
F → ✗
T → ✓

① visited[curr] = T
② print
③ Neighbour add kr

[0 1 2 3 4 5 6]
output

Code

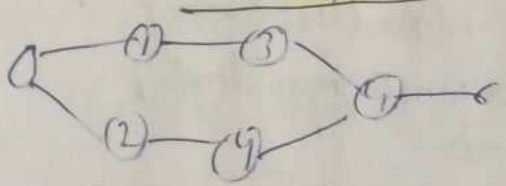
```
public static void bfs(ArrayList<Edge>[] graph){
    Queue<Integer> q = new LinkedList<>();
    boolean vis[] = new boolean[graph.length];
    q.add(e: 0); //source=0
    while(!q.isEmpty()){
        int curr = q.remove();
        if(!vis[curr]){ //visit curr
            say(curr+" ");
            vis[curr] = true;
            for(int i=0; i<graph[curr].size(); i++){
                Edge e = graph[curr].get(i);
                q.add(e.dest);
            }
        }
    }

public static void main(String args[]){
```




```
    int v=7;
    ArrayList<Edge> graph[] = new ArrayList[v];
    createGraph(graph);
    bfs(graph);
    }
}
```

DFS Depth First Search

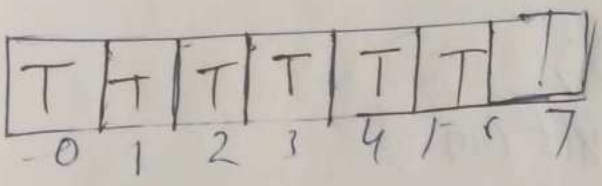
Use Recursion (Stack)

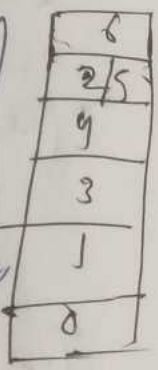


0, 1 3 4 2 5 6

① cur → visit — print vis[cur] = T

② for(int i=0 to k)
if(!vis[curr])
dfs(neighbour)





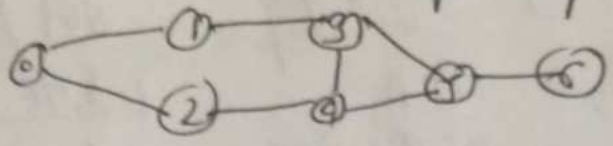
Code:

```
public static void dfs(ArrayList<Edge>[] graph, int cur,
                       boolean vis[]){
    // visit
    sop(cur + " ");
    vis[cur] = true;
    for(int i=0; i<graph[cur].size(); i++){
        Edge e = graph[cur].get(i);
        if(!vis[cur]){
            dfs(graph, e.dst, vis);
        }
    }
}

public static void main(String args[]){
    int v=7
    ArrayList<Edge> graph[] = new ArrayList[v];
    createGraph(graph);
    dfs(graph, 0, new boolean[v]);
}
```

for given dst

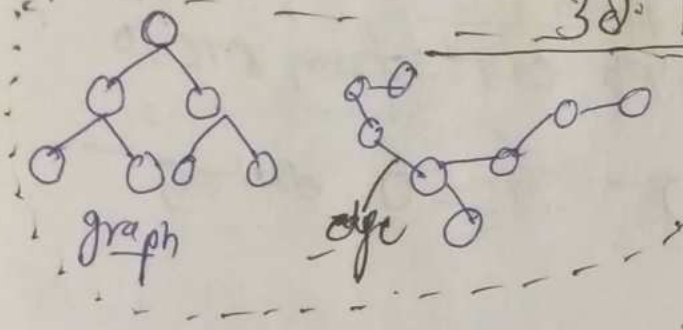
Question Has Path

for given src, dest, tell if a path exist from src to dest



src = 0 dest = 5

38.1 Connected Component



single graph

DFS
BFS

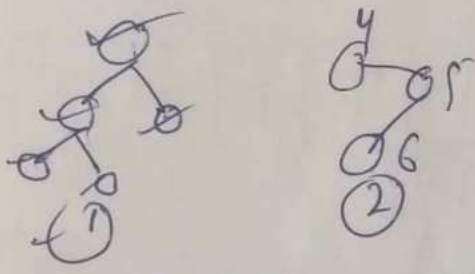
① DFS
vis[] → boolean

```
for(int i=0 ; graph.length)
  if (!vis[i])
    dfsUtil
```

dfsUtil

DFS
helper

dfs.Util



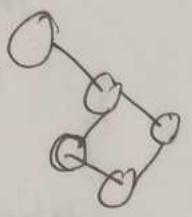
Code-

```
public static void bfs (ArrayList<Edge>[] graph){
    boolean vis[] = new boolean[graph.length];
    for(int i=0; i< graph.lenght ; i++){
        Edge e = graph[curr].get(i);
        q.add(e.dest);
    }
}
```

```
public static void dfs (ArrayList<Edge>[] graph){
    boolean vis[] = new boolean[graph.length];
    for(int i=0; i<graph.length; i++){
        dfsUtil(graph, i, vis);
    }
}
public static void dfsUtil(ArrayList<Edge>[] graph, int curr, boo
System.out.(curr+" ");
    vis[curr] = true;
for(int i=0 ; i<graph[curr].size(); i++){
```

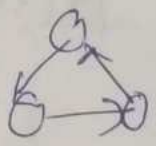
```
Edge e = graph[curr].get(i);
if (!vis[e.dest]){
    dfsUtil(graph, e.dest, vis);
}
}
}
```

38.2 Cycle in graph



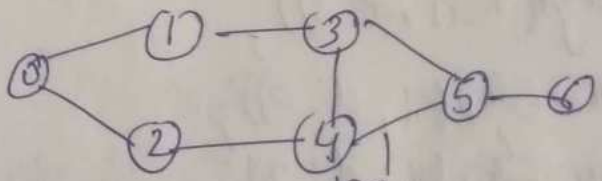
Undirected graph → DFS
→ BFS
→ DSU (Disjoint Set Union)

Directed → DFS
→ BFS
→ Topological sort (Kahn's Algorithm)



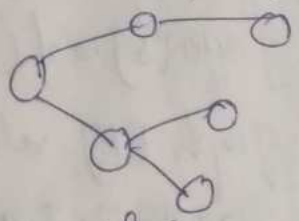
Cycle Detection

Undirected graph → DFS



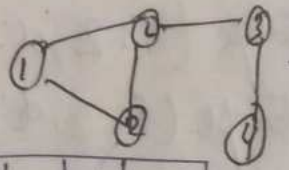
true

no cycle



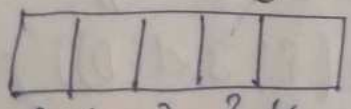
false

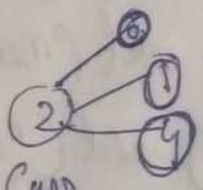
Example



step ① vis

vis:





curr

Case 1:- vis[neigh] = true
parent ✓

Case 2: vis[neigh] ✓
par ✓ continue

Case 3: vis[neigh] ✗
→ normal dfs
call (vis, ff)
cycle → true

dfs
loop
dfsUtil
Rcall

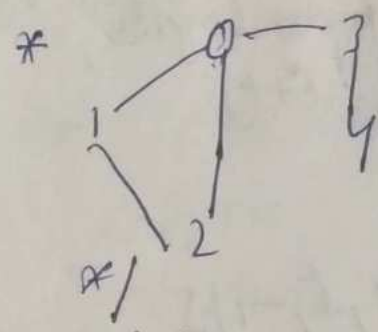
Code:

```
import java.util.*;
public class classroom{
    static class Edge{
        int src;
        int dst;
public Edge (int s, int d){
    this.src=s;
    this.dst=d;
}

static void createGraph(ArrayList<Edge> graph[]){
for(int i=0; i<graph.length; i++){
    graph[i]= new ArrayList<>();
}

    graph[0].add(new Edge(s:0; d:1));
    graph[0].add(new Edge(s:0; d:2));
    graph[0].add(new Edge(s:0, d:3));
    graph[1].add(new Edge(s:1, d:0));
    graph[1].add(new Edge(s:1; d:2));
    graph[2].add(new Edge(s:2, d:0));
    graph[2].add(new Edge(s:2, d:1));
    graph[3].add(new Edge(s:3, d:D));
    graph[3].add(new Edge(s:3, d:4));
    graph[4].add(new Edge(s:4, d:3));
}
```

① Code of cycle detection

```
public static boolean detectCycle(ArrayList<Edge>[] graph){
    boolean vis[] = new boolean[graph.length];
    for(int i=0; i<graph.length; i++){
        if(!vis[i]){
            if(detectCycleUtil(graph, vis, i, -1)){
                return true; // cycle exist in one part
            }
        }
    }
    return false;
}

public static boolean detectCycleUtil(ArrayList<Edge>[] graph, boolean vis[],
                                      int curr, int par){
    vis[curr] = true;
    for(int i=0; i<graph[curr].size(); i++){
        Edge e = graph[curr].get(i);
        if(!vis[e.dest] && detectCycleUtil(graph, vis, e.dest, curr)){
            return true;
        }
        // case 1
        else if(vis[e.dest] && e.dest != par){
            return true;
        }
        // case 2 -> do nothing -> continue
    }
    return false;
}
```

public static void main(String args[]){



*/

int v = 5;

ArrayList<Edge> graph[] = new ArrayList[v];

Create Graph(graph);

Sop(detectCycle(graph));

}}

Dry Run

Case1 vis[neigh]

par → T

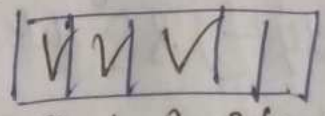
Case2 vis[neigh]

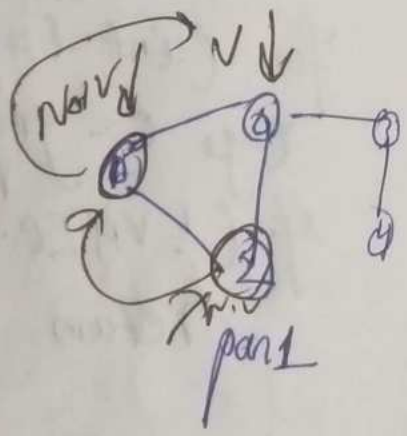
par

Case3: vis[neigh] → normal DFS

Cycle → T

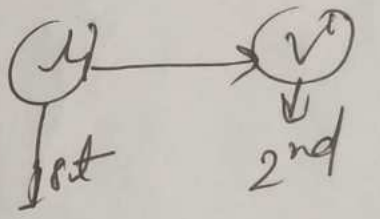
vis

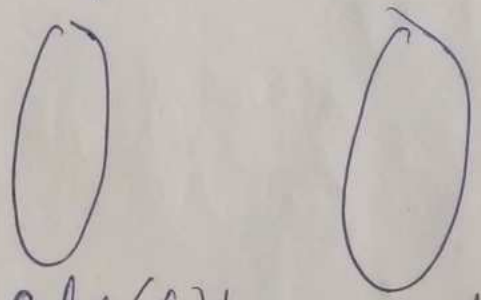


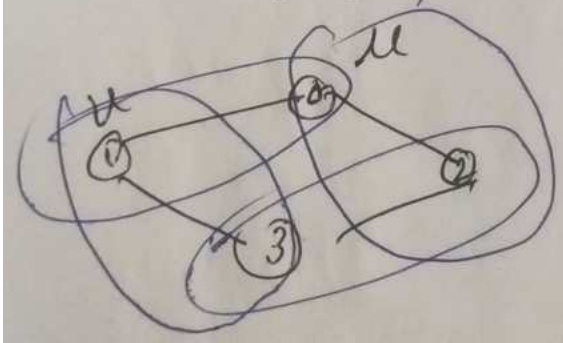


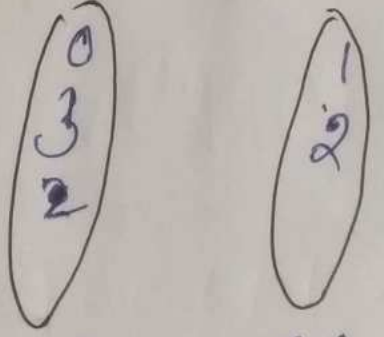
Bipartite graph

A bipartite graph is a graph whose vertices can be divided two independent set, U & V such that every edge (u,v) either connect a vertex from U to V or a vertex from V to U. In other word, for every edge (u,v), either u belong to U & v to V or u belong to V and v to V. We can also say that there is no edge that connect vertices of same set.



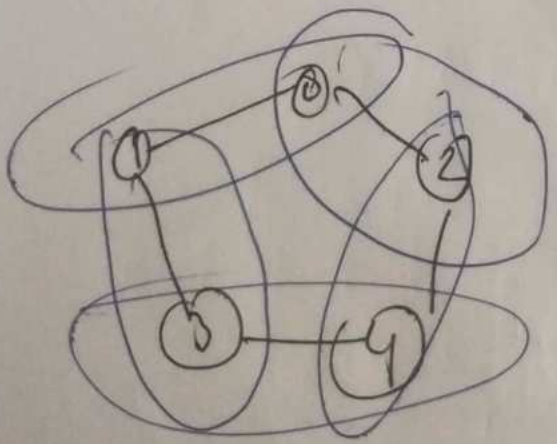




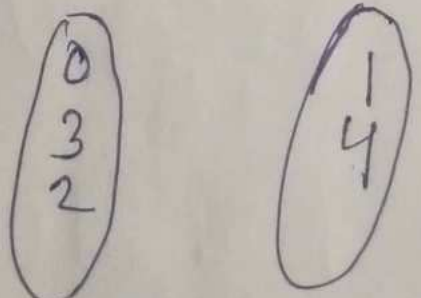


⟹ Bipartite graph

Set1(U) Set2(V)



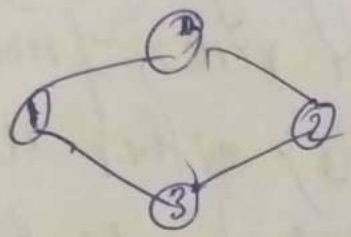
⇒



✗ partite

Set1: U Set2: V

Algorithm ⇒ Coloring



set 1

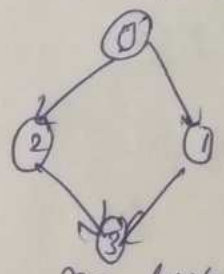
yellow
(0)

set 2

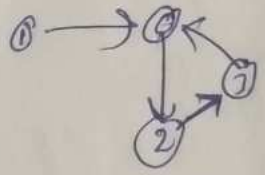
(blue)
(1)

Cycle Detection in Directed graph

Directed graph (DFS)

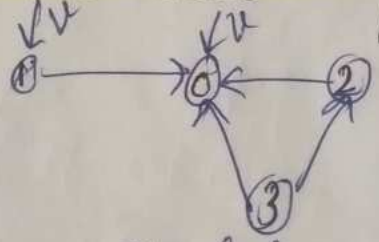


no cycle



cycle

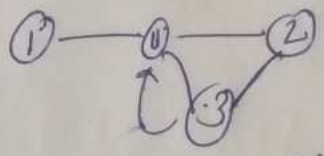
Why undirected DFS approach fail?



Directed
no cycle

par with DFS
+ fail

neigh → vis → ✗ par



undirected
cycle (DFS)

par 2

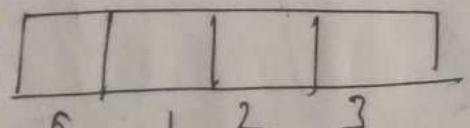
Undirected ~~undirected~~

neighbour (vis)
+
par

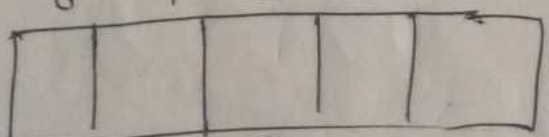
Approach: DFS
+
Stack → recursion
(vis.) Stack (explicit)
(boolean)

① process
② Appraoch
③ dry run

Approach Array: vis



0 1 2 3



0 1 2 3

Stack
call
node visit

dfs (curr)

vis [curr] = true ——— stack [curr] = true;

for (all nigh)

if (! vis [nigh] → visit

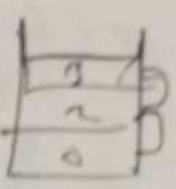
if (stack [nrigh] = true)

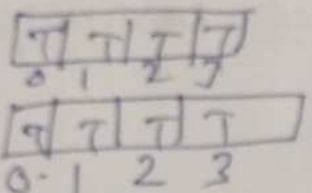
→ cycle exist True

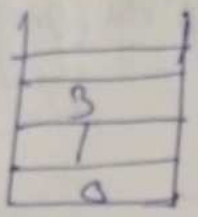
if (! vis [nigh])

is cycle (nerigh) → True

stack [curr] = false







Call Stack

① Static void createGraph (ArrayList<Edge> graphs[]){

for (int i=0; i< graphs.length; i++){

graphs[i] = new ArrayList<>();

}

graphs[0].add(new Edge(s:0, d:2));

graphs[1].add(new Edge(s:1, d:0));

graphs[2].add(new Edge(s:2, d:3));

graphs[3].add(new Edge(s:3, d:0));

}

public Static boolean isCycle(ArrayList<Edge>[] graphs){

boolean vis[] = new boolean[graphs.length];

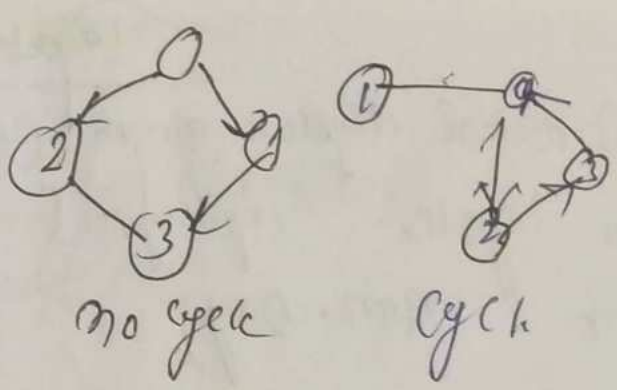
boolean stack[] = new boolean[graphs.length];

for(int i=0; i<graphs.length; i++){

if (!vis[i]){

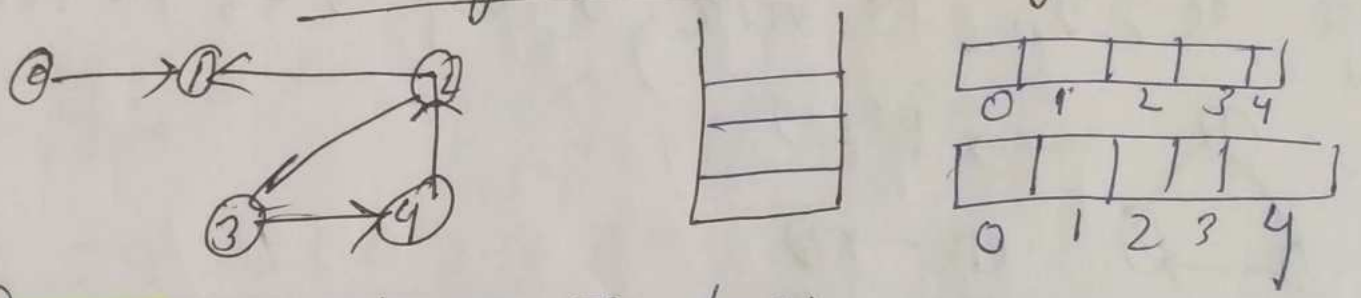
```
if (isCycleUtil(graph, e, vis, stack)){
        return true;
    }
    }
  return false.
}
public static boolean isCycleUtil(ArrayList<Edge>[] graph, int
                                  boolean vis[], b
    vis[curr] = true;
    stack[curr] = true;
for( int e=0; e<graph[curr].size(); e++){
    Edge e = graph[curr].get(i);
    if (stack[e.dst]){
        return true;
    }
    if (!vis[e.dst] && isCycleUtil(graph, e.dst, vis, stac
      return true;
    }
    stack[curr] = false;
      return false;
    }
    public static void main (String arg[]){
          int v = 4;
        ArrayList<Edge> graph[] = new ArrayList<>
            createGraph(graph);
```

boy (is cycle (graph));
}
}



no cycle Cycle

Cycle Detection Dry Run.



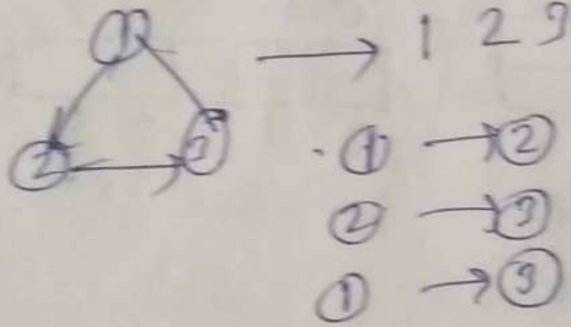
Main Code:

```
vis[curr] = true;
stack[curr] = true;
for(int i=0; i<graph[curr].size(); i++){
  Edge e = graph[curr].get[i];
  if(stack[e.dst]){
      return true;
  }
  else if( !vis[e.dst] && isCycleUtil(graph, e.dst
                                      vis, stack)){
      return true;
  }
}
stack[curr] = false;
return false;
}
```

Topological Sort

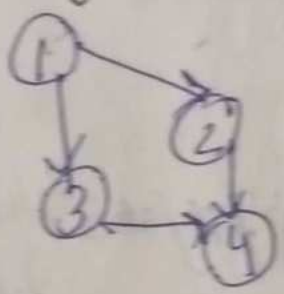
Directed Acyclic graph (DAG) is a directed graph with no cycle. Topological sorting is used for DAG (not for non-DAG)

- It is a linear order of vertices such that every directed edge $u \to v$, the vertex u comes v in the ordering



Dependency graph

Action 1 - boil water
Action 2 - add masala
3 → add maggi
4 → serve maggi

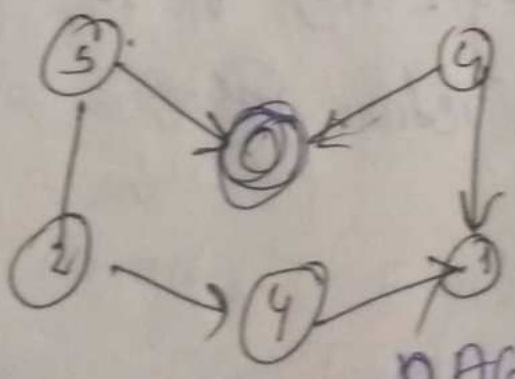


{1 2 3 4}
{1 3 2 4}

Software
OS
VS Code
extension

Cycle ✗ Not dependency

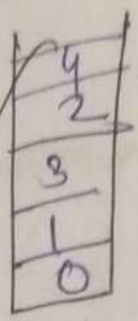
using DFS



DAG

possible Ans

5 4 2 3 1 0
4 5 2 3 1 0



Sorting dfs(graph, curr, vis, stack){

vis[curr] = true;

for(int i=0; i<graph[curr].size(); i++) nigh → call (unvisited)

if (unvisited)

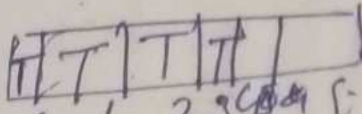
dfs

stack.add(curr)

}

}

trace what come first



5 4 2 3 1 0

O(V+E) dfs

O V

①

}

public static void ~~main~~ topSort(ArrayList<Edge>[] graph){

boolean vis[] = new boolean[graph.length];

Stack<Integer> s = new Stack<>();

for(int i=0; i<graph.length; i++){

if (!vis[i]){

topSortUtil(graph, i, vis, s);

}

}

while (!s.isEmpty()){

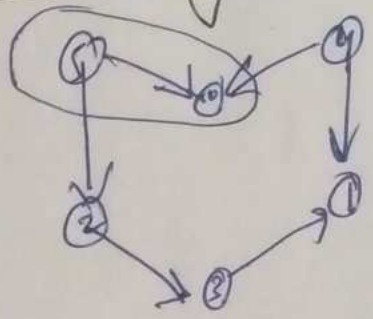
System.out.print(s.pop() + " ");

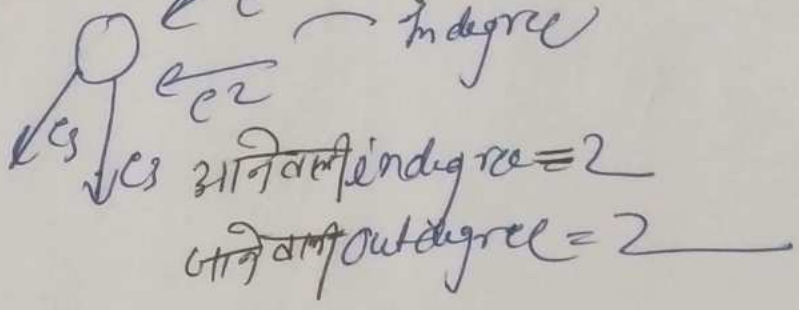
}

```
}
public static void topSortUtil(ArrayList<Edge>[] graph, int curr,
                               boolean vis[], Stack s){
    vis[curr] = true;
for(int i=0; i<graph[curr].size(); i++){
    Edge e = graph[curr].get(i);
    if(!vis[e.dest]){
        topSortUtil(graph, e.dest, vis, s);
    }
}
    s.push(curr);
}
public static void main(String args[]){
    int v = 6;
    ArrayList<Edge> graph[] = new ArrayList[v];
    createGraph(graph);
}
```

39.1 Topological Sorting BFS

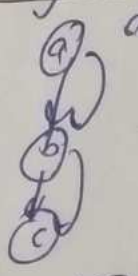
Kahn's Algorithm :-

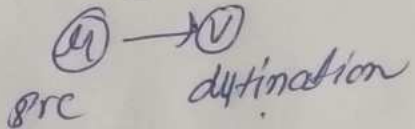




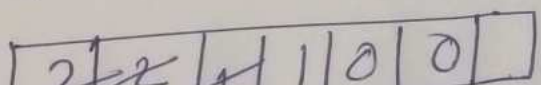
Fact: A DAG has at least one vertex with in degree 0 and one vertex with out degree 0.
(Directed Acyclic graph)

Dependency graph :-



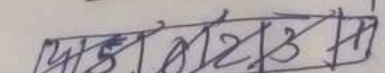


Array



Indegree

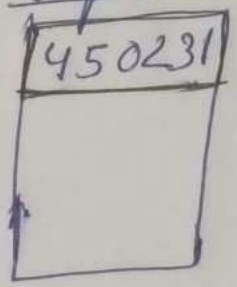
BFS (Inp on Queue)



FIFO

indegree → 0
add in Queue.

Output



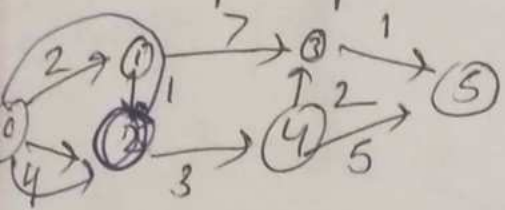
Top Sort logic

Code :-

39.2 Code.

greedy Algo Dijkstra Algorithm $O(n \log n)$

Shortest path from the source to all vertices (weighted graph)

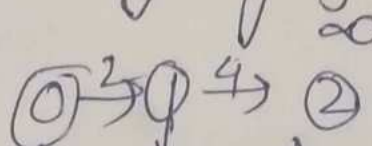


Shortest path 2+1=3

0 → 2 → 4

(optimization)

greedy Algorithm



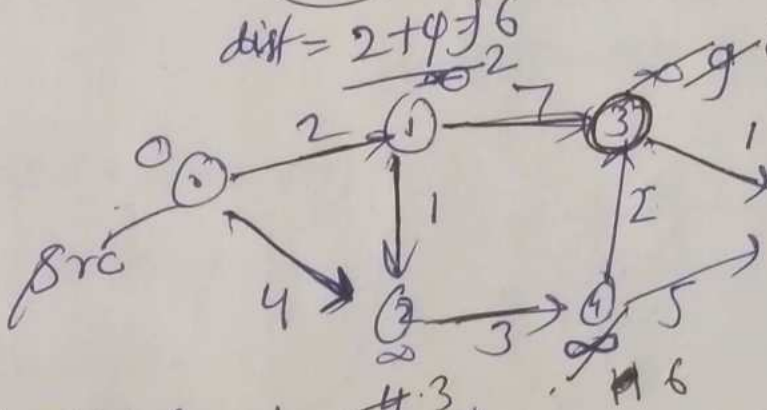
src

dist[i] = src to i

dist[u] = src to u

- dist[u] + wt(u,v) < dist[v]
- dist[v] = dist[u] + wt(u,v)

→ Priority Que



dist = 2+4+6

BFS

visit ✗

shortest ✓

dist[u] + wt(u,v) < dist[v]

dist[v] = dist[u] + wt(u,v)

Final Distance

0 → 0 → 0

0 → 1 = 2

0 → 2 = 3

0 → 3 = 8

0 → 4 = 6

0 → 5 = 9

Algorithm

① Initialize distance

② Priority Que <pair> (n, dist)

③ while (Pq is not empty) {

curr → visit

if (vis[curr] = false) {

neighbours

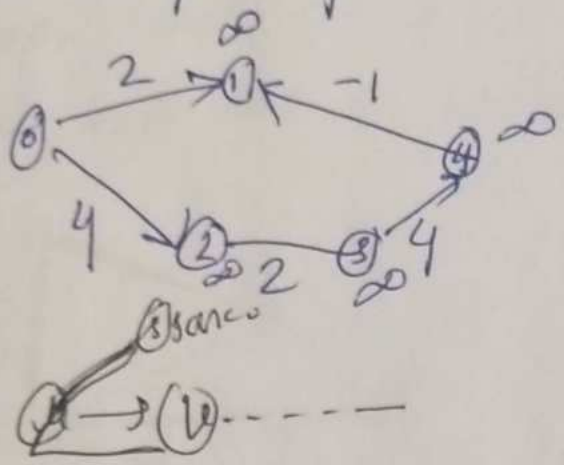
dist[u] + wt(u,v)

< dist[v]

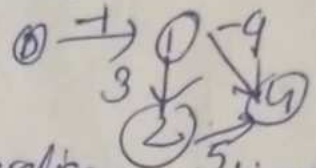
update dist[v]

Bellman Ford Algorithm (Dynamic Programming)

- Shortest path from the source to all vertices (Negative edge)



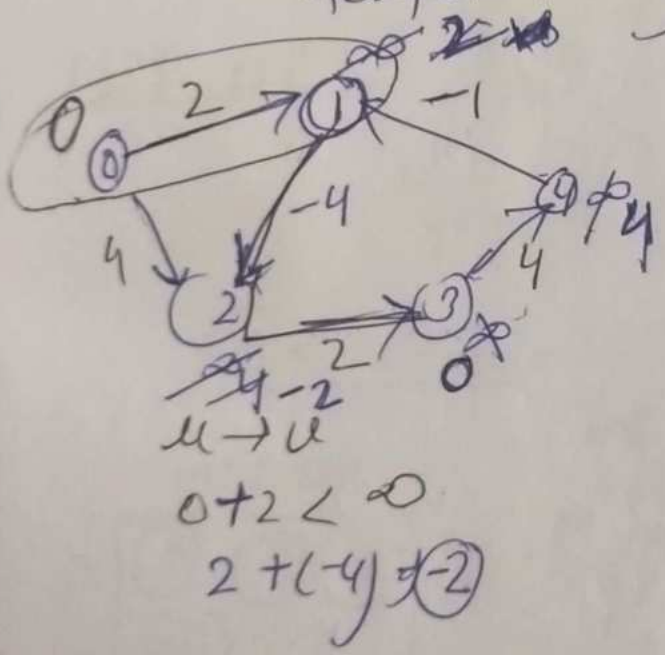
Dijistra ← shortest path



- Perform this operation V-1 times for all edges (u,v)

if dist[u] + wt(u,v) < dist[v] { relaxation
dist[v] = dist[u] + wt(u,v)

Logic for (int i = 0 to v-1) {
edge (u → v)
Relaxation }



	0	1	2	3	4
	0	∞	∞	∞	∞
1	0	2	4-2	0	2
2	0	2	-2	0	4
3	0	2	-2	0	4
4	0	2	-2	0	4

Code: dist[] = new int[v]
for (int i = 0 to v-1)
edge (u → v)
Relaxation

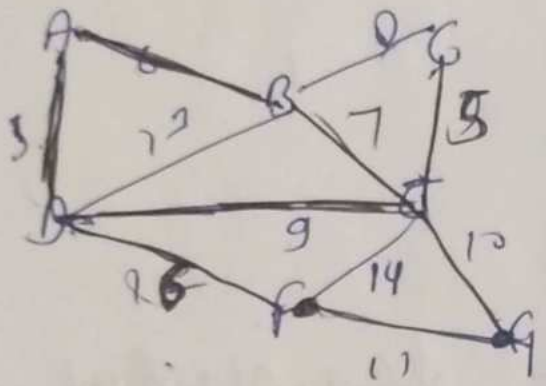
u → v
loop (v-1)th

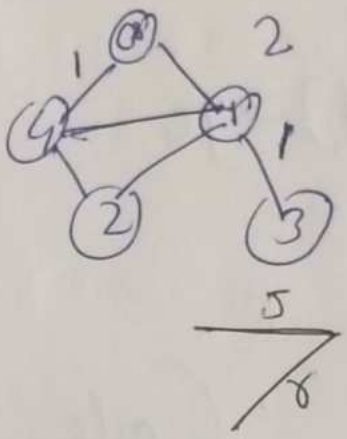
Note Doesn't work for Negative weight cycle

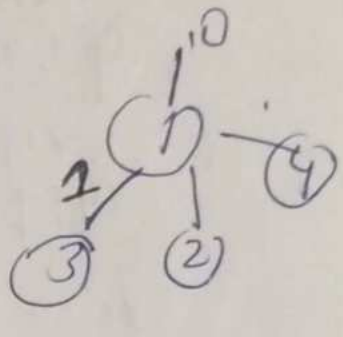
-4+2+4 = 1 positive

Minimum Spanning Tree (MST)

- A minimum spanning tree (MST) or minimum spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connect all the vertices together, without any cycle and with minimum possible total edge weight.



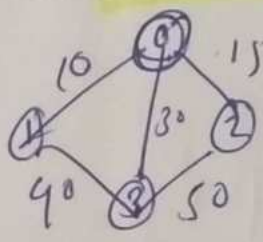




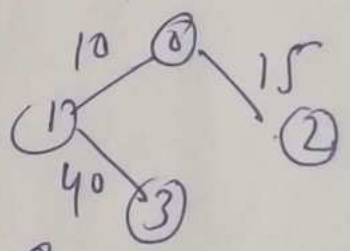
11 < 6+11

Prim's Algorithm

MST set

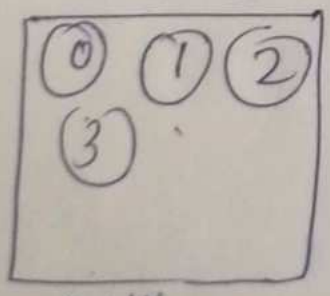


- Undirected + Weighted MST

MST: ①  cost = 65

55 < 65 So 2nd right

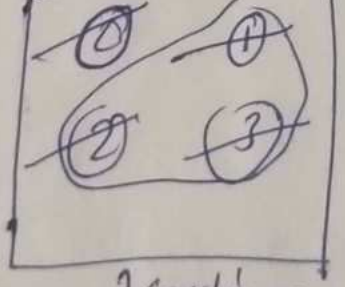
MST set



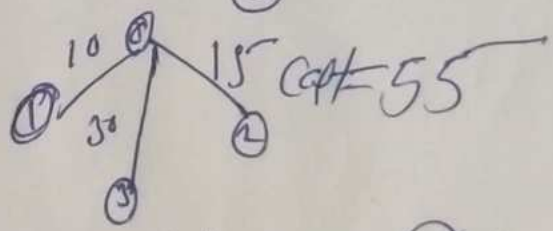
vertices

MST ②

Not MST set

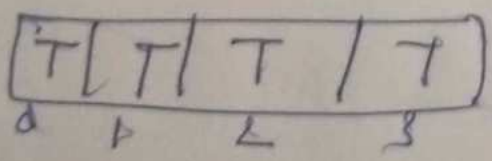


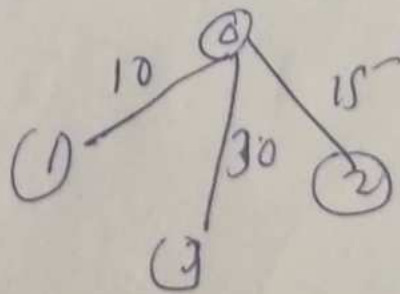
Vertices



{ minimum cost को देखना है left node को लेंगे }

visit





using priority queue < vertex, cost > → minimum pair (cost)

```
pq (0,0)
while (pq.is not empty){
  curr → (v, cost)
if (not vis){
  visit v
MST (arr) edge
}
}
```

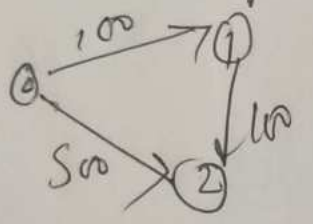
Code · 40.5 Prims Algorithm

41.1 Cheapest flight within K stop

There are n cities connected by some number of flight. You are given an array flight where flight[i] = [from, to, price] indicates that there is a flight.

You are also given three integers src, dst, K, return the cheapest price from src to dst with at most K stop if there is no such route, return -1.

All value are positive:



flight = [[0,1,100],[1,2,100],[0,2,500]]

src = 0, dst 2 K = 1

ans = 200

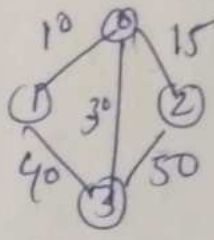
41.3 Connecting cities with minimum cost

Find the minimum cost for connecting all cities on the map

```
cost[][] = {{0,1,2,3,4},
            {1,0,5,0,7},
            {2,5,0,6,0},
            {3,0,6,0,0},
            {4,7,0,0,3}}
```

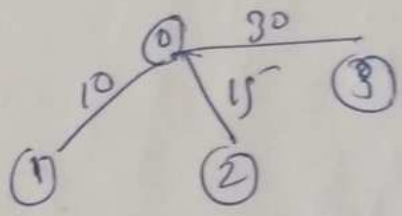
41.6 Kruskal Algorithm

MST greedy



Step:- ① sort edge
② Take min cost edge
(x form cycle
→ include any.

Sorted edge	Weight
(0,1)	10
(0,2)	15
(0,3)	30
(1,3)	40
(2,3)	50



* Cycle नहीं
अलग चाहिए
→ union find

Arraylist <edge> → Collection.Sort()
↓
cost/wt → Count MST

for (i=0 to v-1)

for (count=0 to V-1){

Edge e ← a src
b dest

parA parB

same → x cycle

diff → union (parA, parB)

0 → 1
parent of 0 parent of 1
same
cycle

O(V+ElogE)

Code 41.6

Flood Fill Algorithm

Given a m×n integer grid image where image[i][j] represents the pixel value of the image. You are also given three integers sr, sc, color. You should perform a flood fill on the image starting from the pixel image[sr][sc].

To perform a flood fill, considered the starting pixel, plus any pixel connected 4 directionally to the starting pixel of the same color as the starting pixel, plus any pixel connected 4 directionally to those pixel (also with the same color), and so on. Replace the color of all of the aforementioned pixel with color.

image = [[1,1,1],
[1,1,0],
[1,0,1]]

sr = 1 sc = 1 color = 2

ans = [[2,2,2],
[2,2,0],
[2,0,1]]

42.1 Strongly Connected Component

SCC is a component in which we can reach every vertex of the component from every vertex in that component.

42.2

Bridge En Graph

Tarjans Algorithm:-

- Bridge Ep on edge whose deletion increase the graph's number of connected component