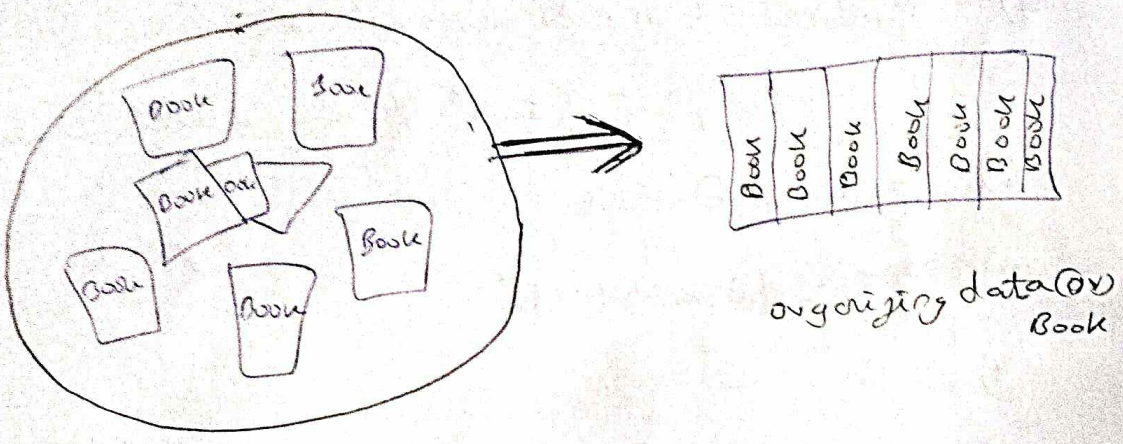
DATA STRUCTURE AND ALGORITHM CONCEPT

1) INTRODUCTION.
2) RECURSION
3) BIG O NOTATION.
4) ARRAYS (One dimension and Two Dimensional Array)
5) LINKED LIST (Single / circle, Double / circle)
6) STACK / QUEUE
7) TREE / BINARY TREE
8) AVL TREE
9) BINARY HEAP
10) TRIE
11) HASHING
12) SORT ALGORITHM
13) SELECTION SORT
14) SEARCH ALGORITHM
15) GRAPH ALGORITHM
16) GREEDY ALGORITHM
17) DIVIDE AND CONQUER ALGORITHM
18) Dynamic PROGRAMING
19) WILD WEST

What is datastructure?

Datastructure:-

→ Different ways of organizing data on your computer



What is algorithm?

ALGORITHM:-

→ Set of Instruction to performa task

Example:- Step1: Go to bus stop

Step2: Take a bus

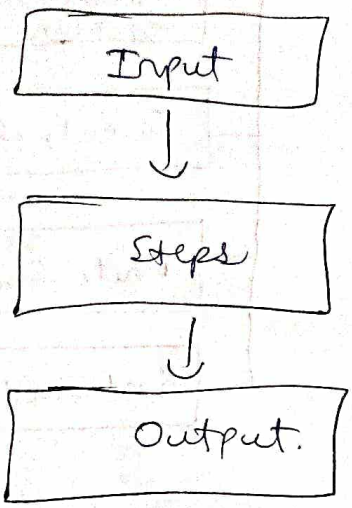
Step3: Go to office

Example:-

Step 1: Go to Starbucks

Step 2: Pay money

Step 3: Take a coffee.

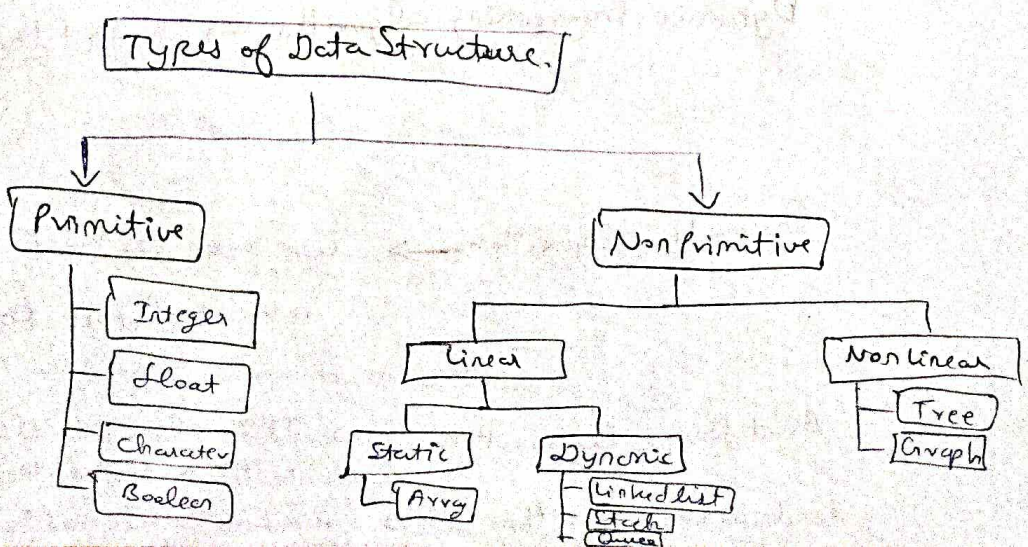


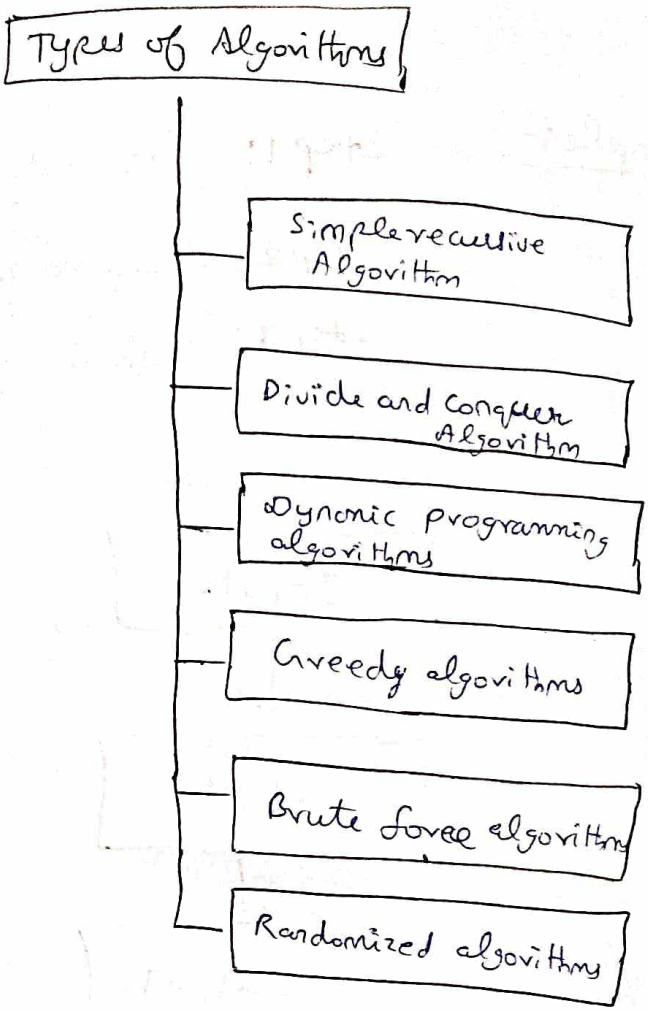
ALGORITHM:- NASA, ISRO, SPACE X, TESLA.

Google Maps ⟶ Data Structure and algorithm

Why are Data Structure and Algorithm in Interviews?

⟶ Improves Problem Solving Skills





Simple recursive Algorithm →

Divide and conquer algorithm → Divide the problem into smaller subproblems.
Eg:- Quicksort and mergesort.

Dynamic Programing algorithm → To find the best solution

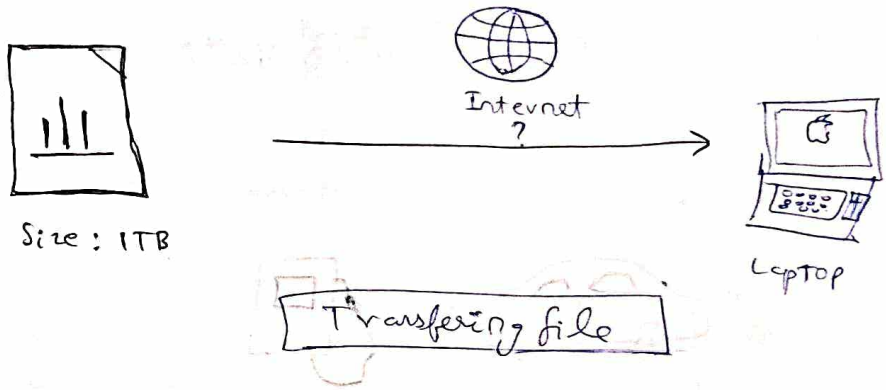
Greedy algorithm → we take the best without worrying about future consequences.

Brut force algorithms → It simply tries all possibilities until a satisfactory solution is found

Randomize algorithm → Random number atleast once to make decisi

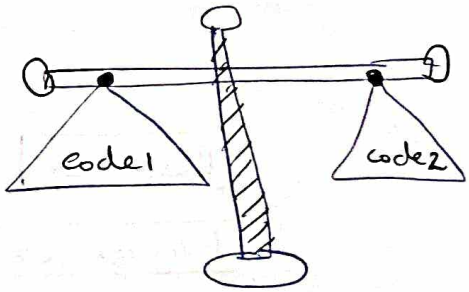
Big O Notation

Big O → Big O is the language and metric we use to describe the efficiency of algorithms



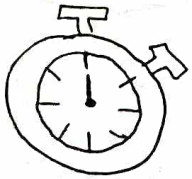
O(1) & O(n)

Big O → efficiency of code



code1 → easy to read

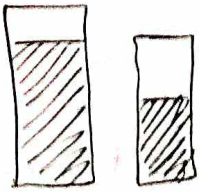
code 2 → difficult to read



code1 → 30 sec
code 2 → 60 sec
} Time Complexity

code 1 faster than code 2

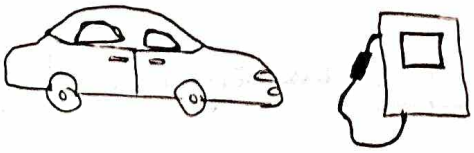
code 1 → better than code 2



Code 1 → More space
Code 2 → Less space
} Space complexity

Code1 > code2

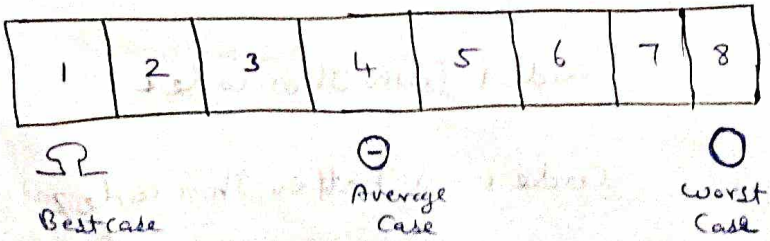
$O(n)$ (On^2) $O(2^N)$



CityTraffic → 20 litres (Worst Case)
Highway → 10 litres (Best case)
Mixed condition → 10 litres (Average Case)

Best Case → Code executes 5 seconds
Average Case → Code executes in 1 min
Worst Case → Code execute in 5 min

Array



Big O — Worst.

Big Ω — Best

Big Θ — Average

Runtime Complexity

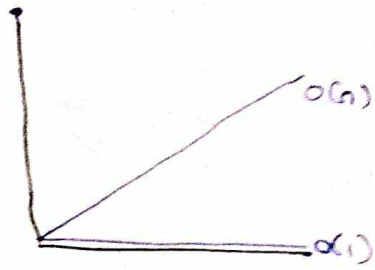
O(1) → order of 1

o(1) → Constant

o(n) → Linear

O(log N) → Logarithmic

O(N²) → Quadratic



O(1) → execution time will not change / constant

O(n) → time takes to execute a code (for loop)

O(log N) → Time complexity

O(N²) → Nested for loop

Faster computer → Faster memory access

Slower computer → Slower memory access.

One for loop → (n) Number

Two for loop → (n^2) Square of Number

Three for loop → (n^3) Cube of Number

Time Complexity $O(\log n)$

Code1 → 30 sec

Code2 → 60 sec

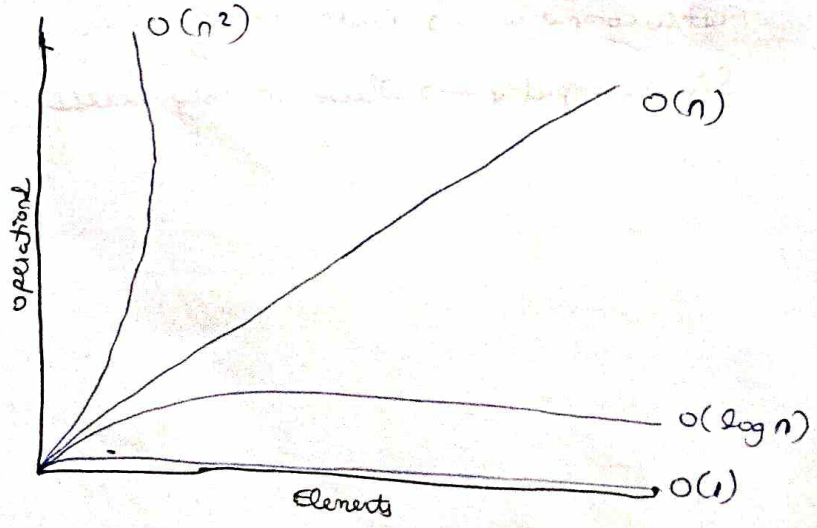
$2^3 = 8$

$\log_2 8 = 3$

$2^? = 8$

$\log_2 8 = ?$

$? = 3$



Add vs Multiply

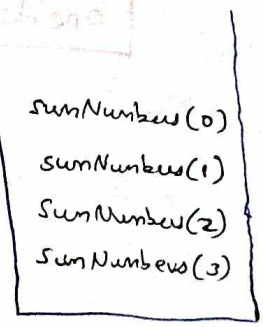
Add the Runtimes : $O(A+B)$

Multiply the Routines : $O(A*B)$

Space Complexity:-

```
public static int sumNumbers (int n) {
    if (n <= 0) {
        return 0;
    }
    return n + subNumbers(n-1);
}
```

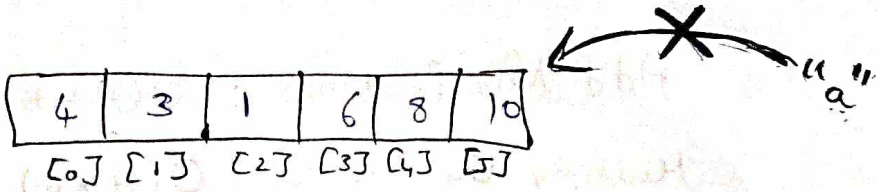
Java Visualizer → Executing Code. Step by Step



for(; ;) → $O(n)$

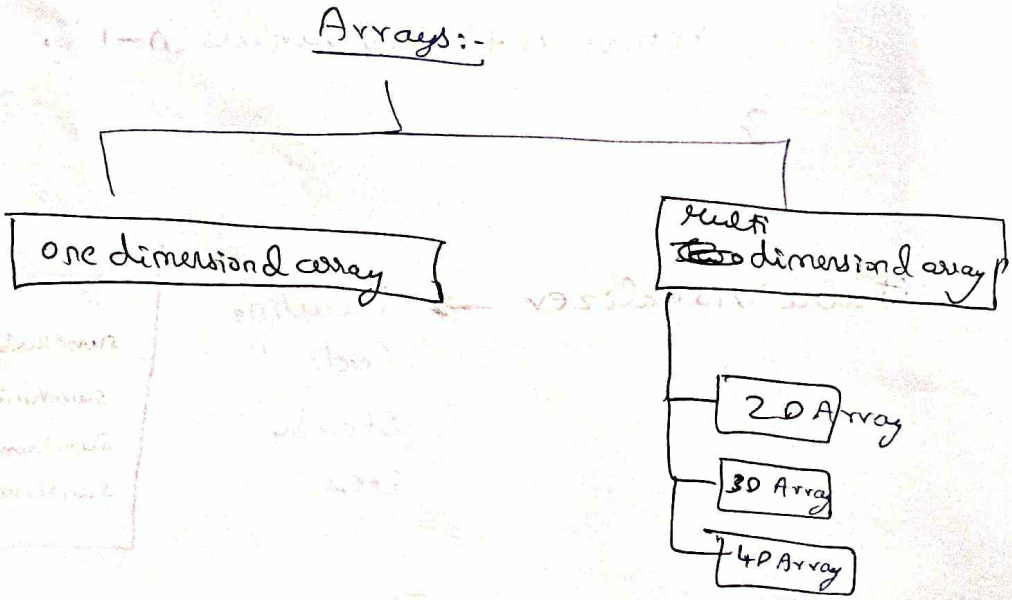
S.O.P(" ") → $O(1)$ // constant

Arrays :-

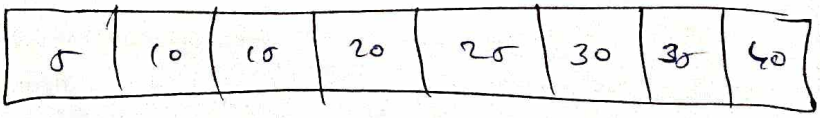


Array → An array is a datastructure consisting of collection of elements

Types of Arrays:-



One dimensional array



Multidimensional array :-

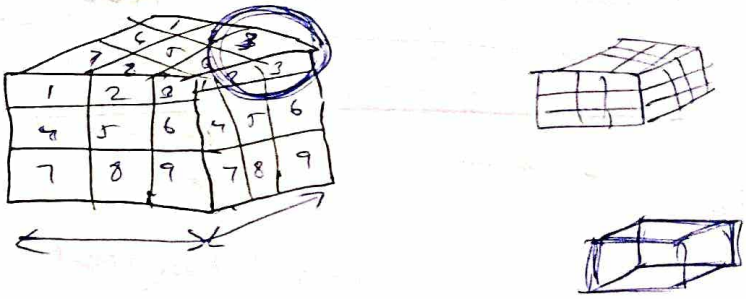
Two dimensional array :-

	0	1	2	3	4	5	6	7
0	5	10	15	20	25	30	35	40
1	6	12	18	24	30	36	42	48
2	7	14	21	28	35	42	49	56
3	8	16	24	32	40	48	56	64

a[0][4] = 25

a[2][5] = 42

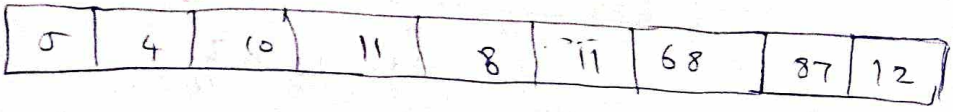
Three dimensional array :-



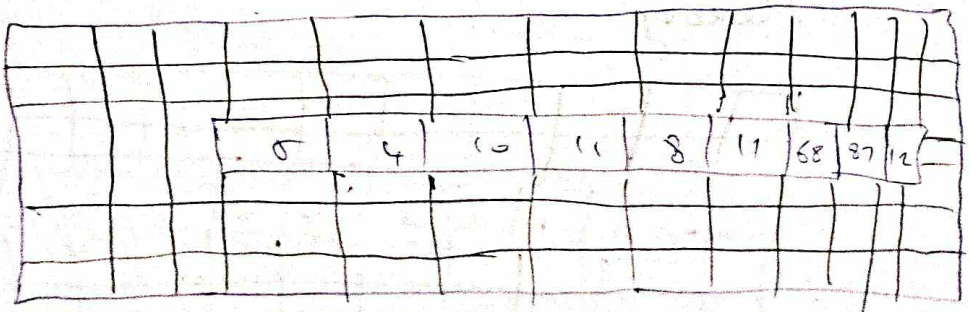
a[0][0][0]

Arrays in Memory :-

One dimensional



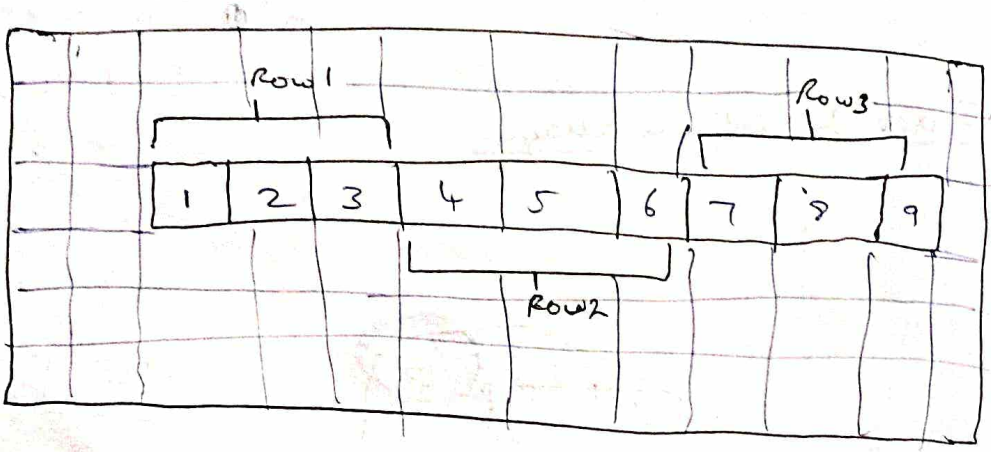
Memory.



Two dimensional

1	2	3
4	5	6
7	8	9

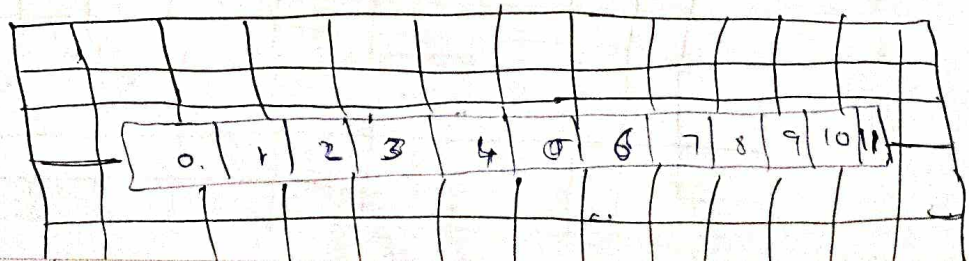
Memory



Three dimensional array

```
{{{0, 1, 2},
  {3, 4, 5},
  {6, 7, 8},
  {9, 10, 11},
  {12, 13, 14},
  {15, 16, 17}}};
```

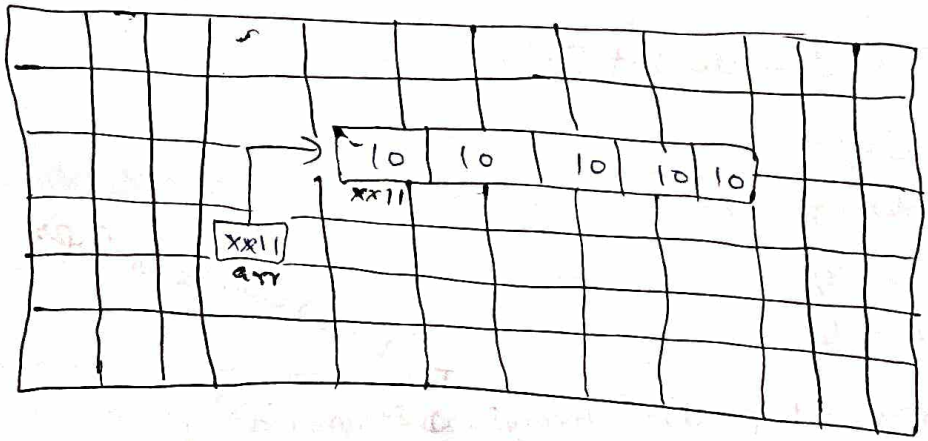
Memory



Creating an array :-

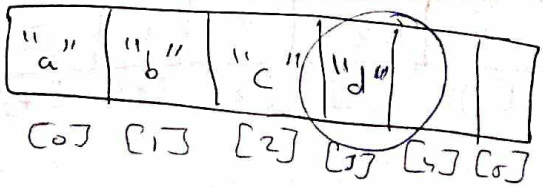
How we create an array.

Memory :-



Syntax : datatype[] arr

Insertion in Array

myArray = 

myArray[3] = "d"

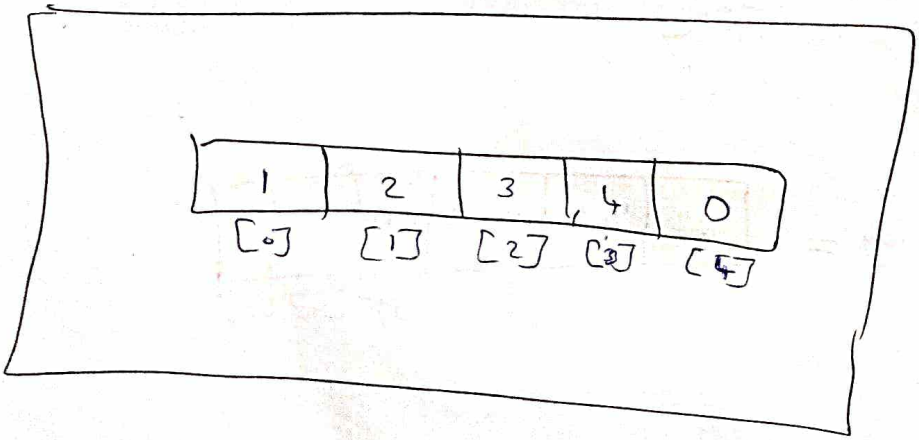
myArray[5] = "f"

Basic array program.

```
import java.util.*;

class type
{
public static void main (String [] args)
{
int a[] = new int [5];
a[0]=1;
a[1]=2;
a[2]=3;
a[3]=4;
System.out.println(Arrays.toString(a));
}
}
```

→ should be in Caps.

Memory



System.err.println ("This line represent in red colour");

myArray =

"a"	"b"	"c"	"d"	"e"
[0]	[1]	[2]	[3]	[4]

```
myArray[0] = "a";
      .
      .
      .
myArray[4] = "e";
```

Combination of One dimensional array ⟶ Two dimensional array

Creation of Array

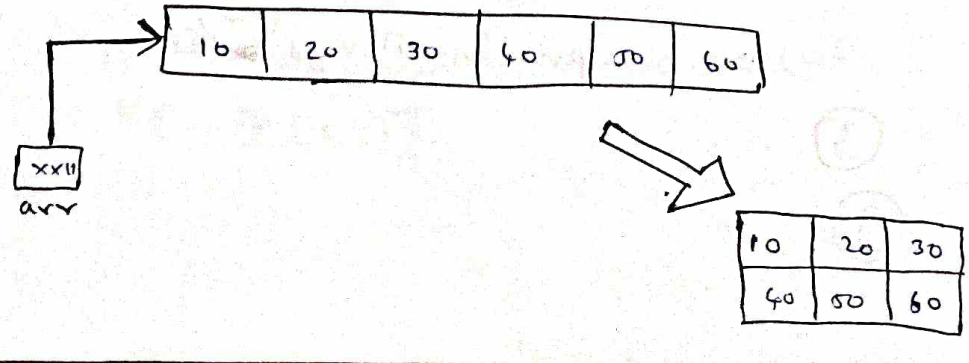
When we create an array, we:

Declare :- Creates a reference to array

Instantiation of an array :- creates an array

Initialization :- assigns values to cells in array

Memory :- (Two dimensional array)



Brackets:-

One dimensional array = [ ] → (1)

Two dimensional array = [ ][ ] → (2)

Three dimensional array = [ ][ ][ ] → (3)

Steps for Creating an array → (3) Steps.

Program for 1D array:

```
import java.util.Arrays; //Array library
class Main
{
public static void main (String [] args)
{
~~int~~
// Step1 - Declare.
~~int a = new int [5];~~
int [] a;
//Step 2 - Instantiate
a = new int [2];
// Step 3 - Initialize;
a[0] = 1;
a[1] = 2;

System.out.println(Arrays.toString(a));
}
}
```

Program for 2D Arrays :

```
import java.util.Arrays; // Array Library.
class Main
{
public static void main (String [] args)
{
// Step 1 - Declare.
int [][] int2DArray;
// Step 2 - Instantiate
int2DArray = new int[2][2];
// Step 3 - Initialize
int2DArray [0][0] = 1;
int2DArray [0][1] = 2;
int2DArray [1][0] = 3;
int2DArray [1][1] = 4;

System.out.println (Arrays.deepToString (int2DArray));
}
}
```

OUTPUT : [[1, 2], [3, 4]]

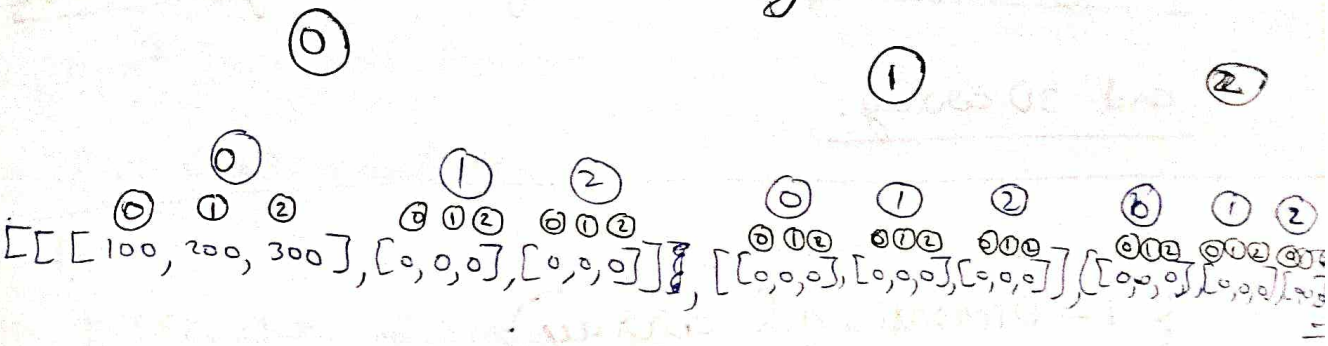
Program for 3D Array:

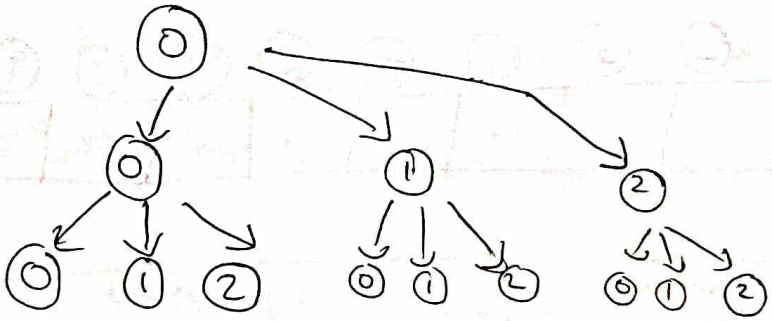
```
import java.util.Arrays; // Array libraries

class Main
{
public static void main (String [] args)
{ // Step 1 - Declare.
int [][][] array3D;

// Step 2 - Instantiate
array3D = new int[3][3][3];

// Step 3 - Initialize
array3D [0][0][0] = 100;
array3D [0][0][1] = 200;
array3D [0][0][2] = 300;

System.out.println (Arrays.deepToString (array3D));
}
}
```

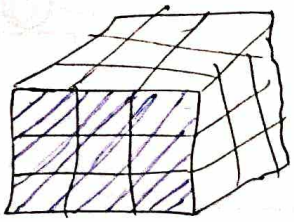
OUTPUT:-

[[[100, 200, 300], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]

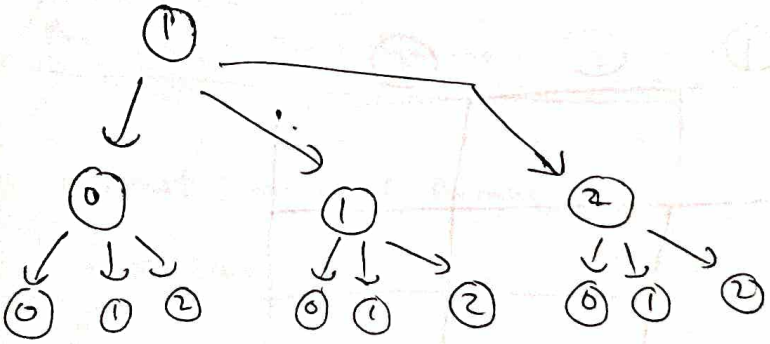
How arrays are arranged in 3D array

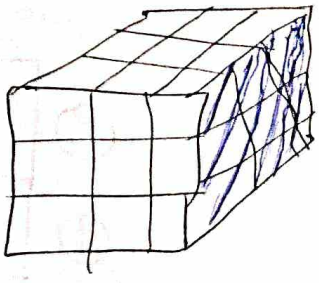




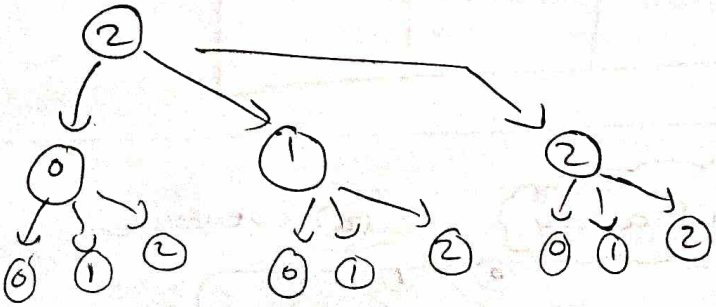


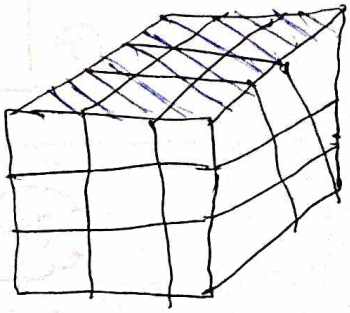
(0)





(1)

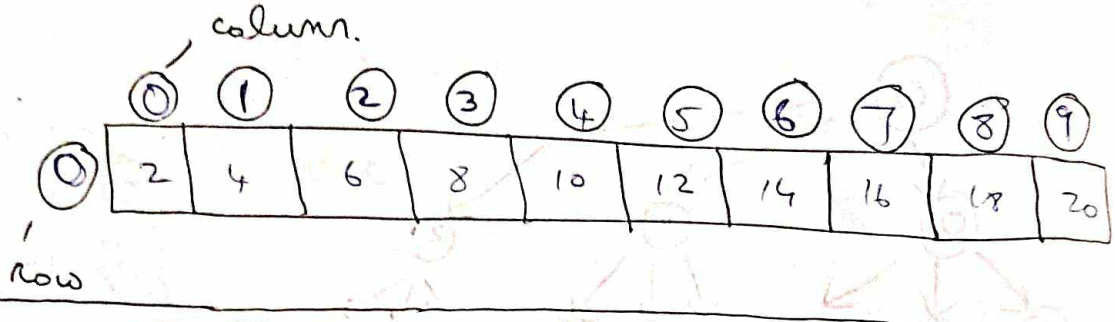




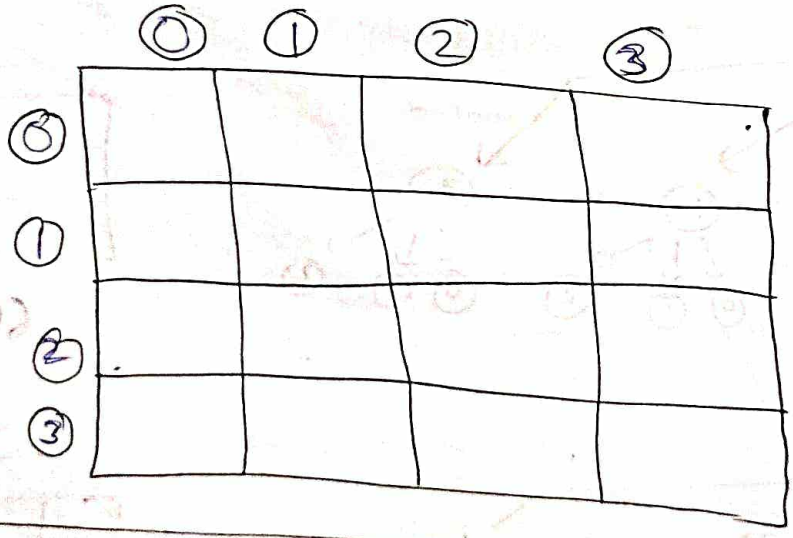
(2)

Diagramatic representation of 1D array => 2D array and 3D array.

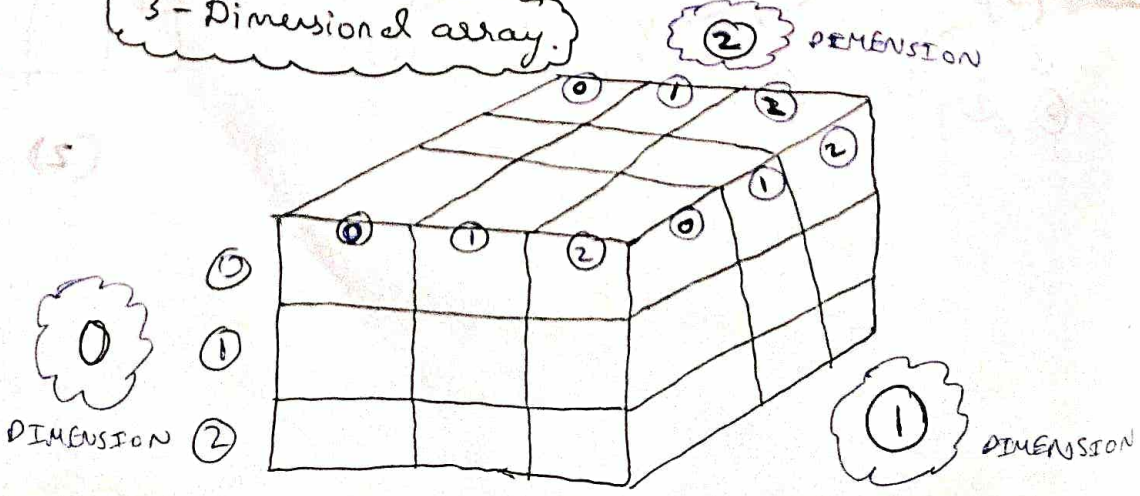
1-Dimensional array.



2-Dimensional array's.



3-Dimensional array.



Program for one dimensional array.

```
import java.util.Arrays;
class Onedimensionalarray
{
 public static void main(String[] args)
 {
  int[] a = {1, 2, 3, 4, 5, 6};
  System.out.println(Arrays.toString(a));
 }
}
```

OUTPUT : [1, 2, 3, 4, 5, 6]

Program for Two dimensional array

```
import java.util.Arrays;
class Twodimensionalarray
{
 public static void main(String[] args)
 {
  int[][] a = {{1, 2, 3, 4, 5, 6}, {1, 2, 3, 4, 5, 6}};
  System.out.println(Arrays.deepToString(a));
 }
}
```

OUTPUT : [[1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]]

If we use (Arrays.toString) then output be [I@3796 1aa, [I@cac 736d]]

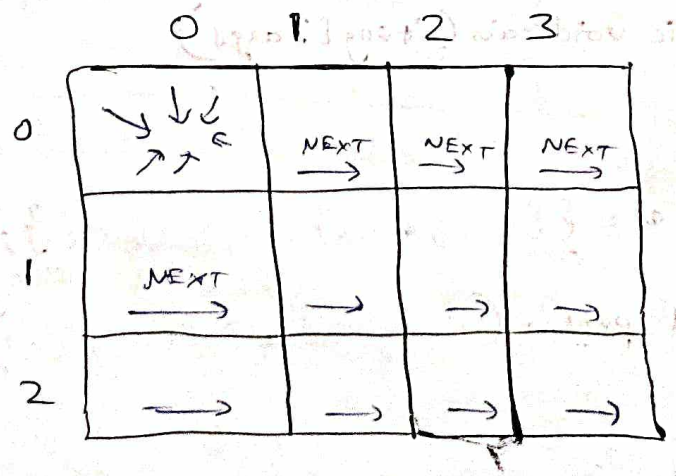
Program for Three dimensional array

```
import java.util.Arrays;

class Threedimensionalarray
{
public static void main (String [] args)
{
int [][][] a = {{{1,2,3,4,5,6},{1,2,3,4,5,6},
                {1,2,3,4,5,6}}};
System.out.println(Arrays.deepToString(a));
}
}
```

OUTPUT : [[[1, 3, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6]]]

XXXXXXXXX

TWO DIMENSIONAL ARRAY



Coding Exercise

Given 2D array calculate the sum of digonal elements.

Example

myArray2D = {{1,2,3},{4,5,6},{7,8,9}};

Program :-

```
class Exercise {

    public static int sumDiagonalElements(int[][] array)
    {
        int sum = 0;
        int numRows = array.length;
        for(int i = 0; i < numRows; i++){
        sum += array[i][i];
        }
        return sum;
    }
}
```

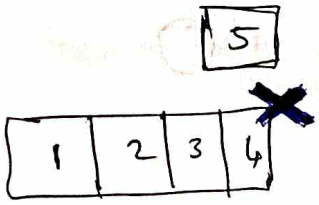
what is digonal elements in an array.

$A = [a_{ij}]$

An element of a matrix $A = [a_{ij}]$ is a digonal elements of matrix if $i=j$ when rows and columns are equal.

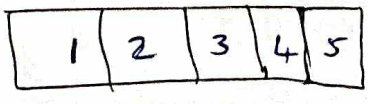
Arrays have fixed size.

Array Size ⟶ Static



int[] a = new int[5];

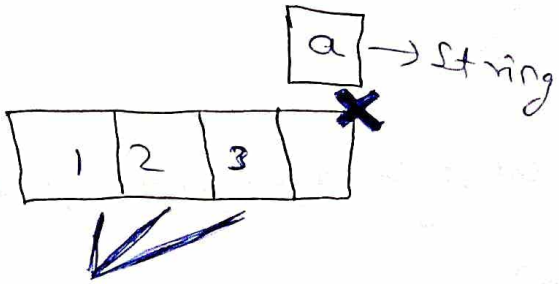
Array list Size ⟶ Dynamic



Arraylist <Integer> a =
new Arraylist <Integ

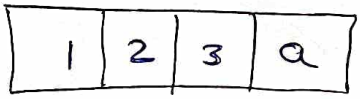
Type Safety

Array → Type Safe.




```
int[] a = new int[5];
```

Arraylist → Not Type Safe




```
Arraylist<Integer> a = new Arraylist<Integer>();
```

Array program:

```
import java.util.*;
class Main {
public static void main (String [] args)
{
int[] age = new int[5];
age [0] = 15;
age [1] = 16;
age [2] = 18;
age [3] = 23;
age [4] = 24;
System.out.println (Array.toString(a));
} }
```

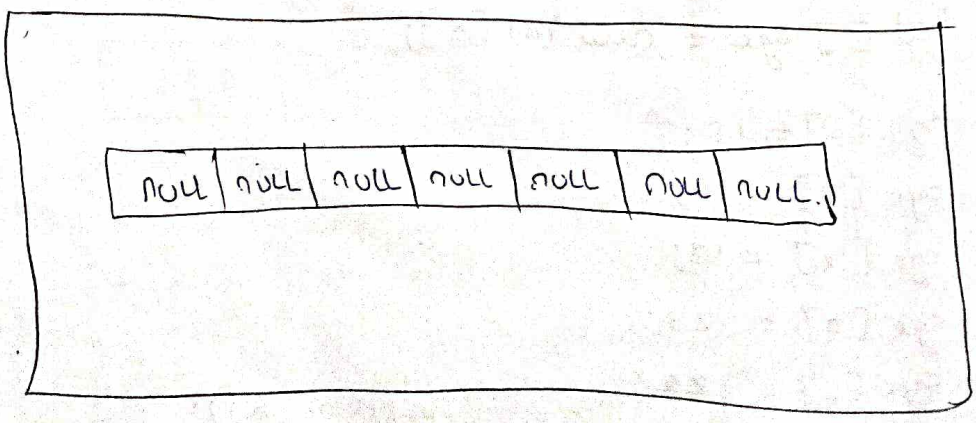
Arraylist program.

```
import java.util.*;
Class Main {

public static void main (String [] args)

{

Arraylist <Integer> a = new Arraylist <Integer> ();

 a.add (0);
 a.add (1);
 a.add (2);

 System.out.println (a);

 }
}
```

Array → Static in nature.

Arraylist → Dynamic in nature

Arraylist in Memory



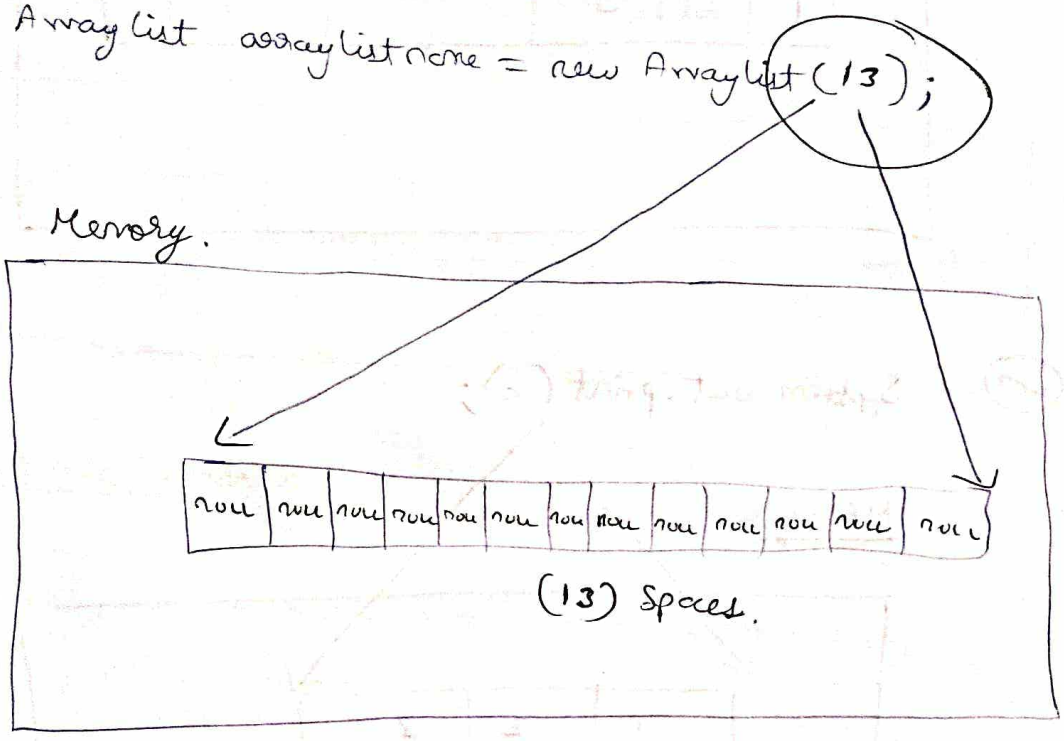
WORKING OF ARRAY LIST.

// declaring array list / Explanation of Array List

```
ArrayList <Integer> arrayListName = new ArrayList<Integer>();
```

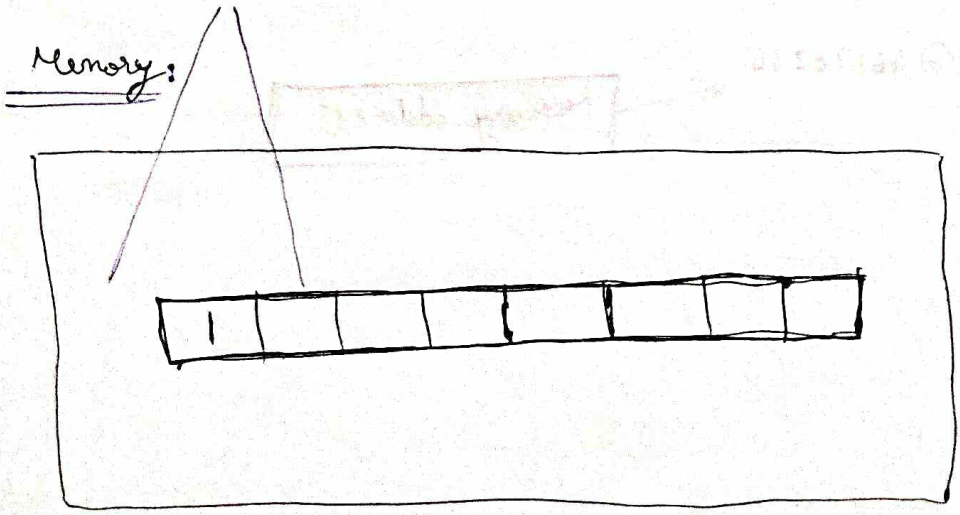
(13)

```
ArrayList arrayListName = new ArrayList(13);
```

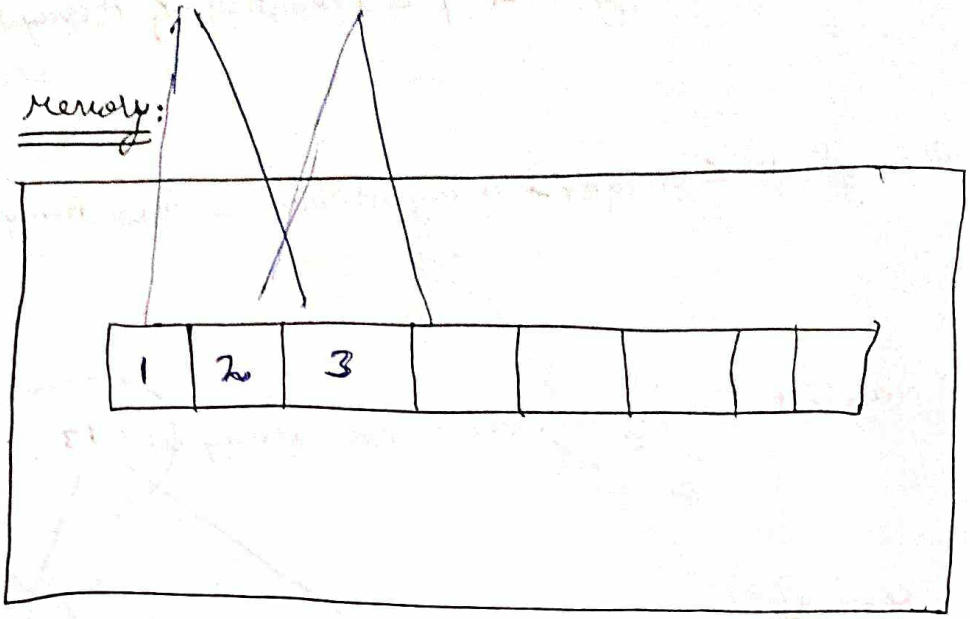



```
ArrayList<Integer> a = new ArrayList<Integer>();
a.add(1);
```

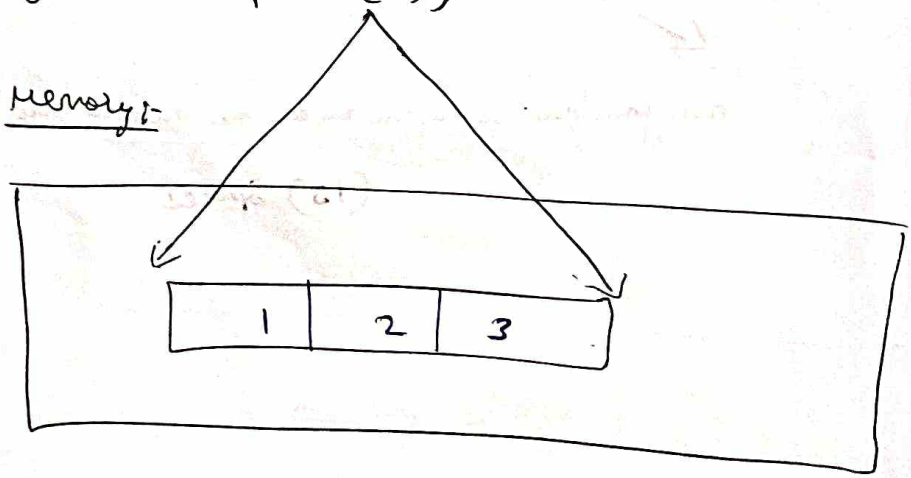
Memory:



a.add (1);
a.add (2); a.add (3);



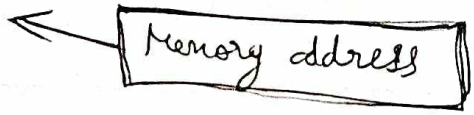
System.out.print (a);



OUTPUT:- [1, 2, 3]

---

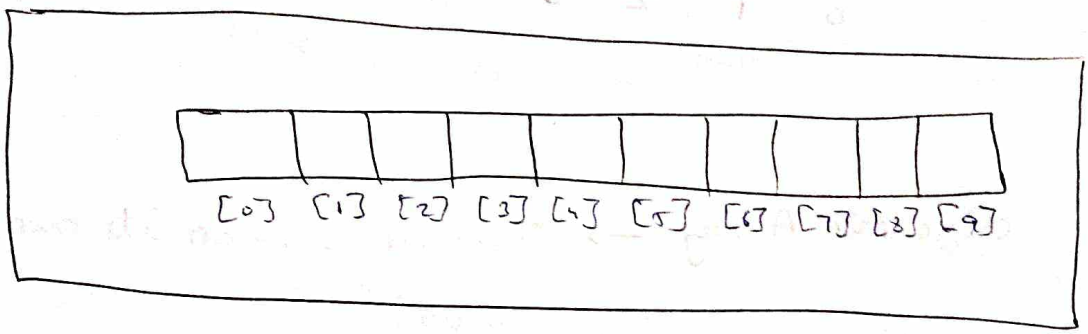
① I@4617c264



Static Memory:-

```
ArrayList<Integer> a = new ArrayList<Integer>(10);
```

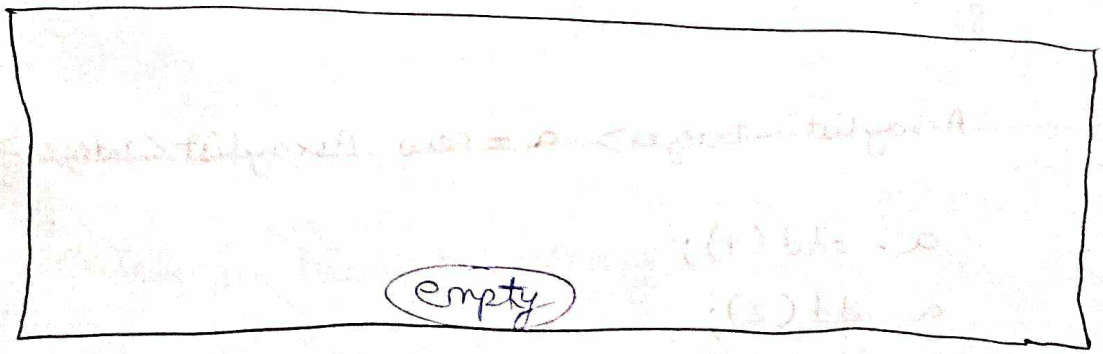
Memory : (static) → cannot be changed



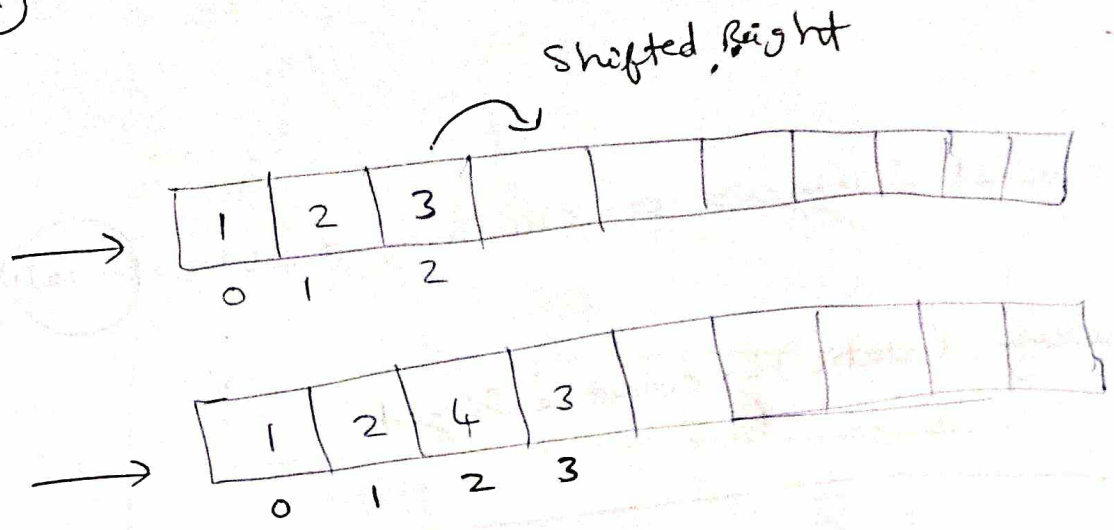
Dynamic Memory:-

```
ArrayList<Integer> a = new ArrayList<Integer>();
```

Memory (Dynamic) → can be changed



① add(2,4)



Dynamic Array → Increase size on its own.

How to get an particular array

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

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

a.add(1);
a.add(2);

System.out.println(a);
System.out.println(a.get(0));
}
}
```

OUTPUT: [1, 2]

① → getting 0 value in next line.

To display elements in array

```
String letter = a.get(i);

S.O.P(letter);
```

→ use for loop to get all element of array.

(or)

Use for each loop.

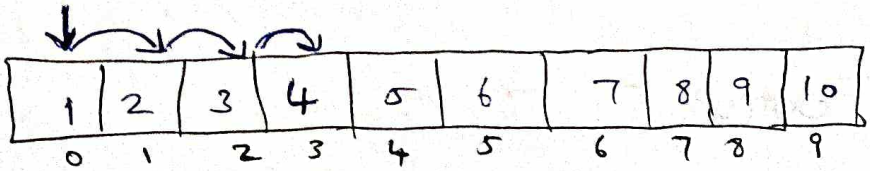
(or)

Use Iterator.

```
Iterator<String> iterator = a.iterator();
while (iterator.hasNext()) {
    String letter = iterator.next();
    System.out.println(letter);
}
```

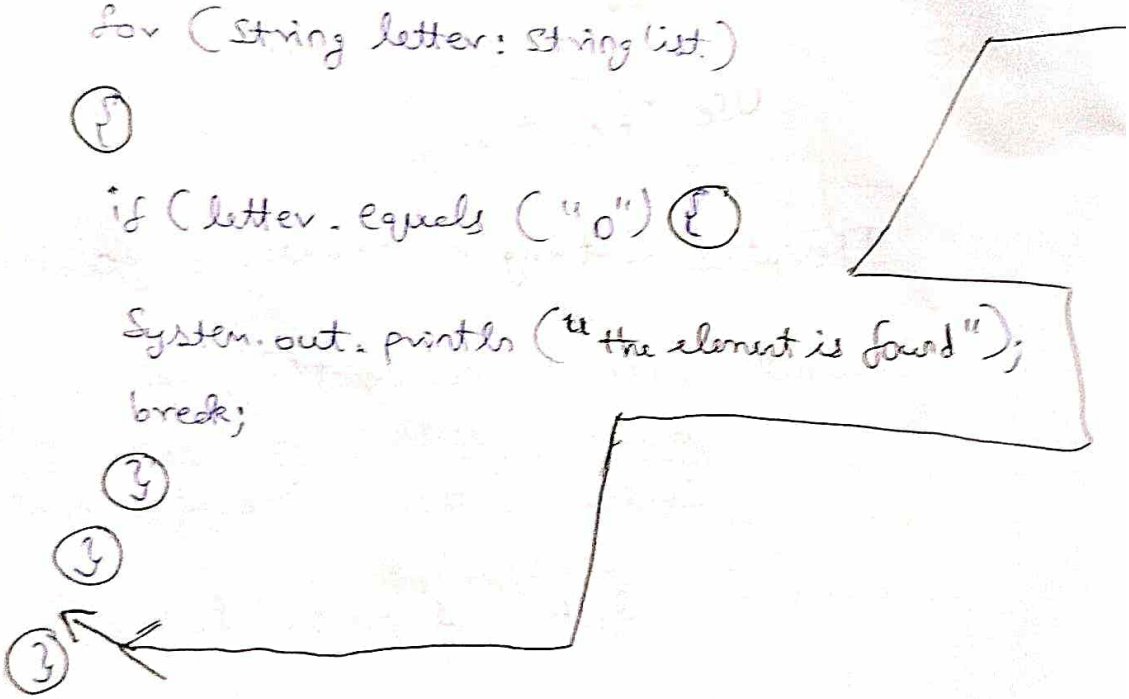
Search for Element in Array list

Search for 4



Program :

```
Class Main {

public static void main (String[] args)
{

ArrayList<String> stringlist = new ArrayList<String>
                (Arrays.asList("A","E",
                "C","O","E"));

for (String letter: stringlist)
{

if (letter.equals("O")){

System.out.println("the element is found");
break;
}
}
}
```



Array and Arraylist

→ Both Array and Arraylist output comes in [ ] this format.

OUTPUT: [ ]

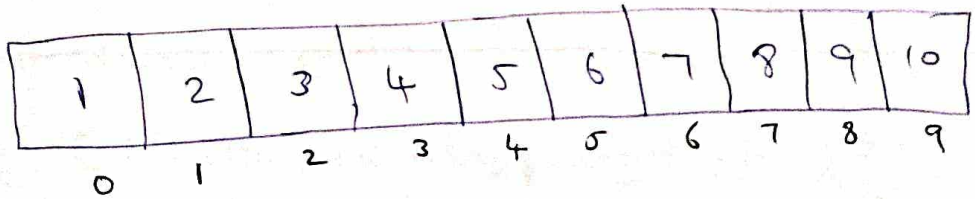
Add this line last:

```
int index = stringList.indexOf("c");
System.out.println("The element is found at
                    index :" + index);
```

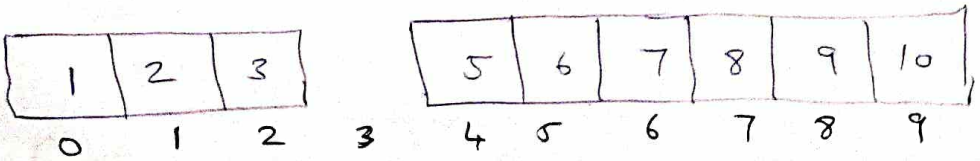
Delete Element from Array list.

remove(3) — remove(index) (or) remove(object)

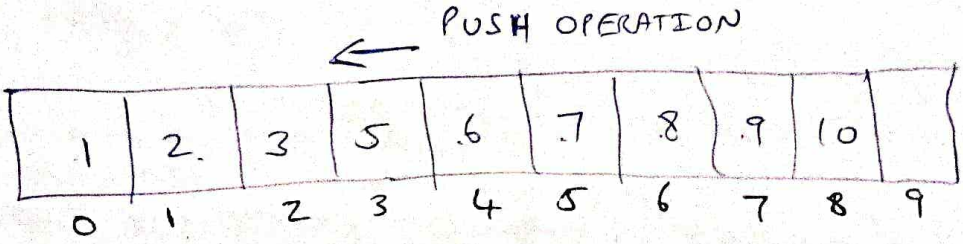
STEP ①



STEP ②



STEP ③



Program :-

```
class Main {

public static void main (String [] args)

{

ArrayList <String> Stringlist = new ArrayList
                                <String> (Arrays.asList ("A", "B",
                                "C", "D", "E"));

Stringlist.remove (2);

System.out.println (stringlist);

}

}
```

Interview question.

① Find Sum and product of array's.

```
class Main {

public static void main (String [] args)
{

Main main = new Main();
int [] customArray = {1, 3, 4, 5};
main. spotArray (customArray);
}

void spotArray (int [] array) {

int sum = 0;
int product = 1;
for (int i=0 ; i< array. length ; i++) {

sum += array [i];
}

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

product *= array [i];
}

System.out.println ( sum + " , " + product);
}
}
```

Sample:

Sum += array[i] → working principle

STEP ①

②
Sum = Sum + ①

① ↶
Sum = ① + 2 = 3

③
Sum = ③ + 3 = 6

⑥
Sum = ⑥ + 4 = 10

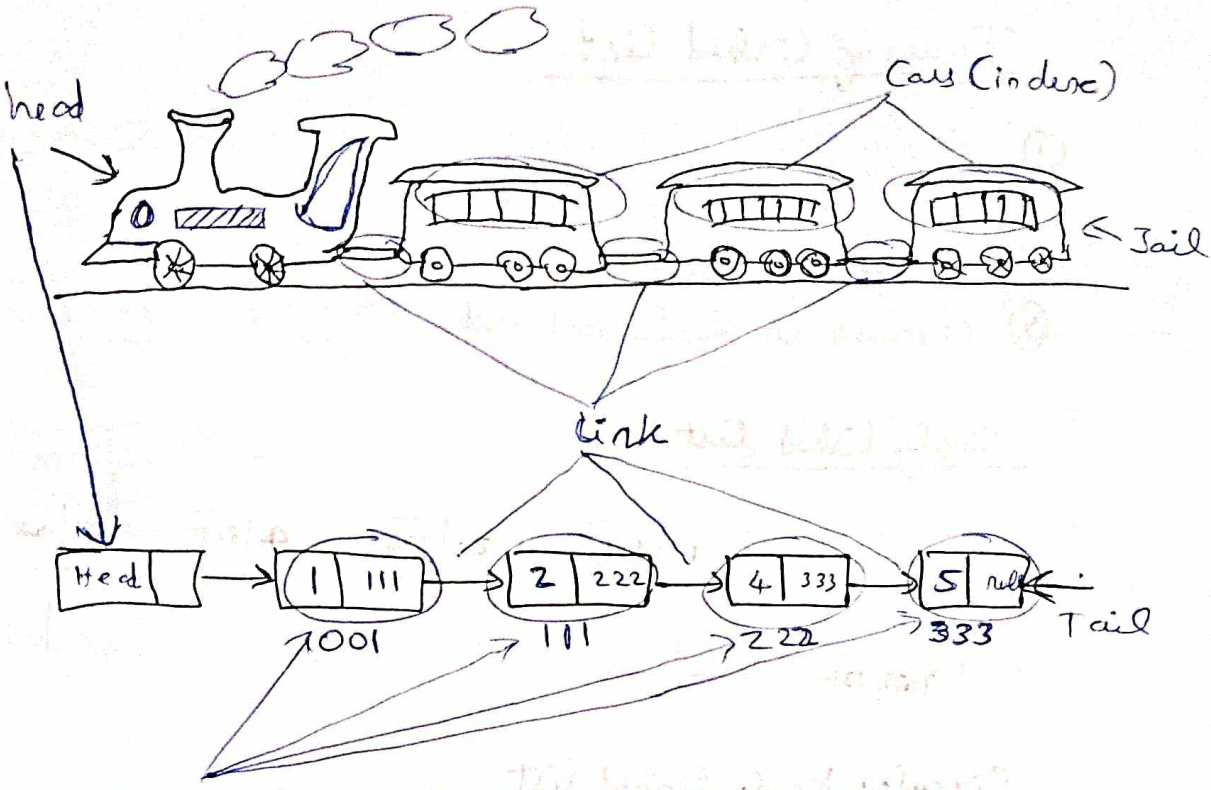
⑩
Sum = ⑩ + 5 = 15

Linked List

what is Linked List.

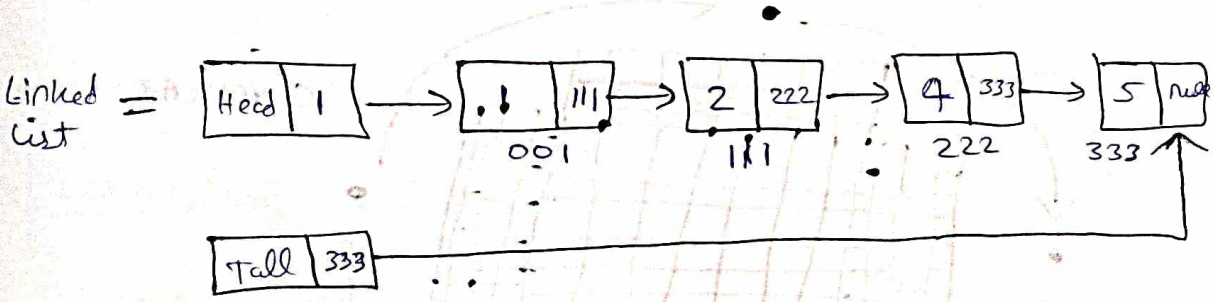
Linked list is a form of sequential collection and it does not have to be in order.

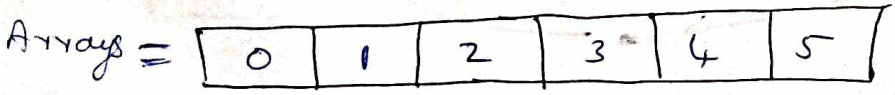
Linked list → independent node.
Each node → reference to next node.



Nodes: data and references.

Linked List Vs Array.



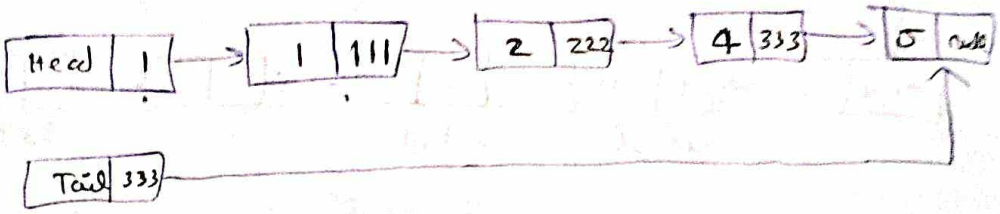


Types of Linked lisk in Next Page.

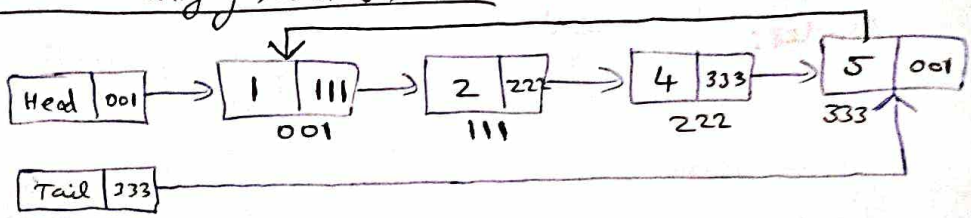
Types of Linked List.

1. Singly linked list
2. Circular singly linked list
3. Doubly linked list
4. Circular doubly linked list

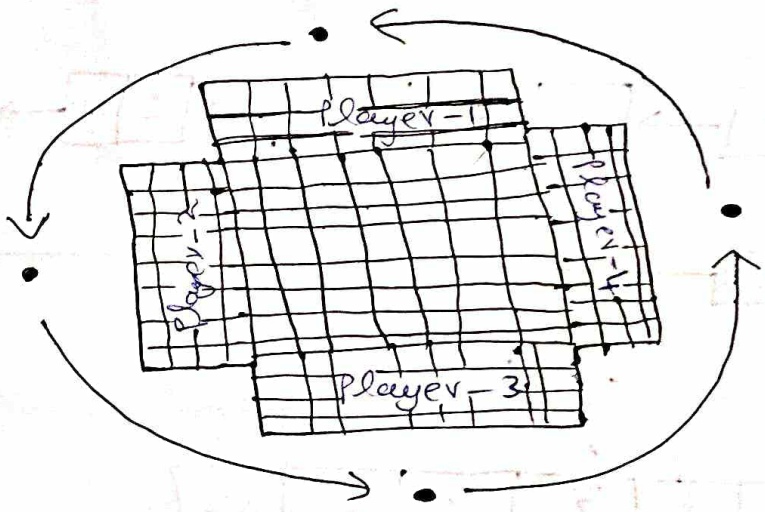
Singly linked list.



Circular singly linked list.

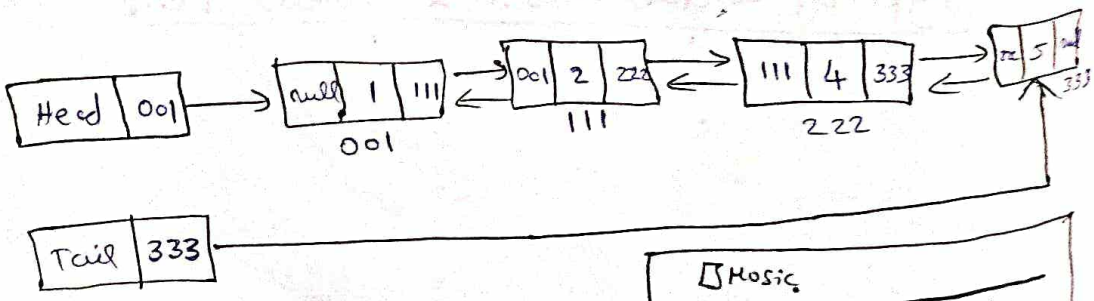


Example (4 Player Chess Board)

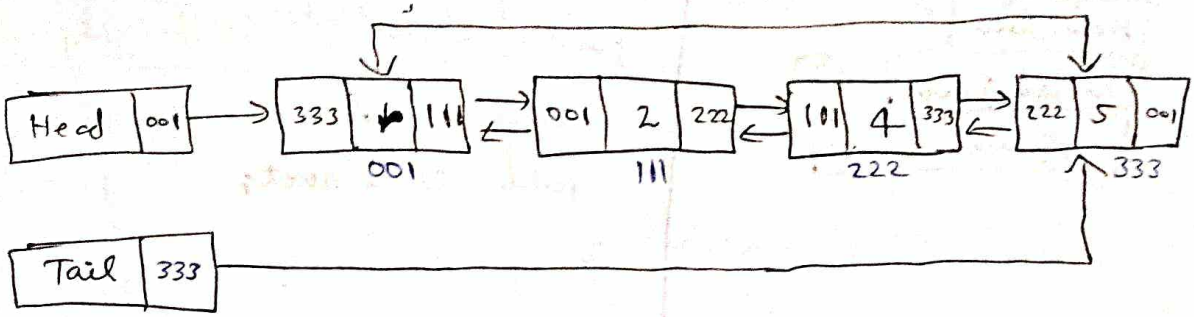


CIRCULAR

Doubly linked list.



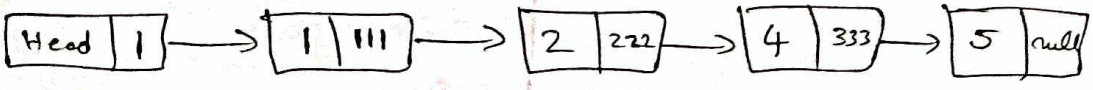
Circular doubly linked list.



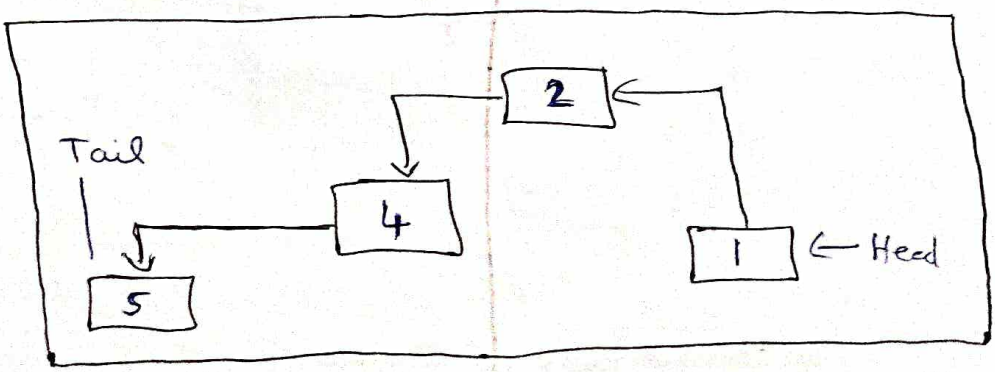
Example: ctrl + shift + Tab (or) windows + tab (Apple)

Linked list in Memory:-

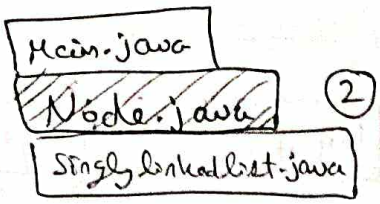
Linked list



Memory.

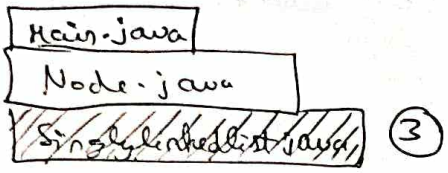


Program



Node.java

```
public class Node {
    public int value;
    public Node next;
}
```



Singly linked list.java

```
public class SinglyLinkedList {
    public Node head;
    public Node tail;
    public int size;

    public Node createSinglyLinkedList
    (int nodeValue) {
        head = new Node();
        ~~node.next = null;~~
        Node node = new Node();
        node.next = null;
        node.value = nodeValue;
        head = node;
        tail = node;
        size = 1;
        return head;
    }
}
```

Main.java ①

Node.java

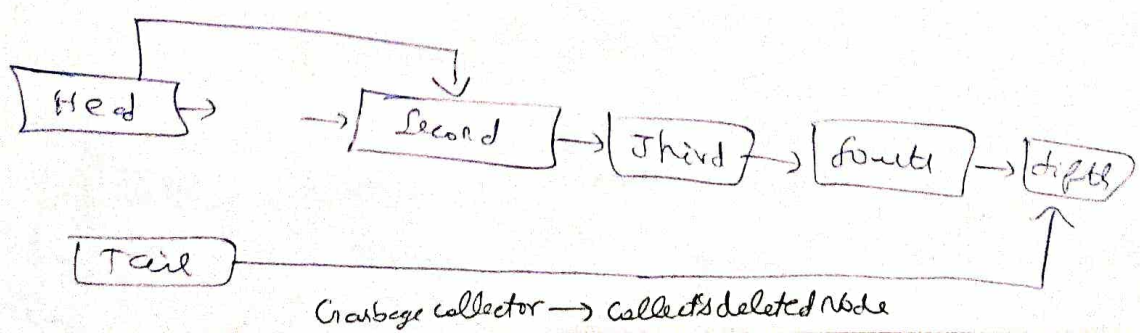
SinglyLinkedList.java

```
class Main {
public static void main
(String [] args)
SinglyLinkedList SLL = new
SinglyLinkedList();
SLL.createSinglyLinkedList(5);
System.out.println(SLL.head.
                   value);
```

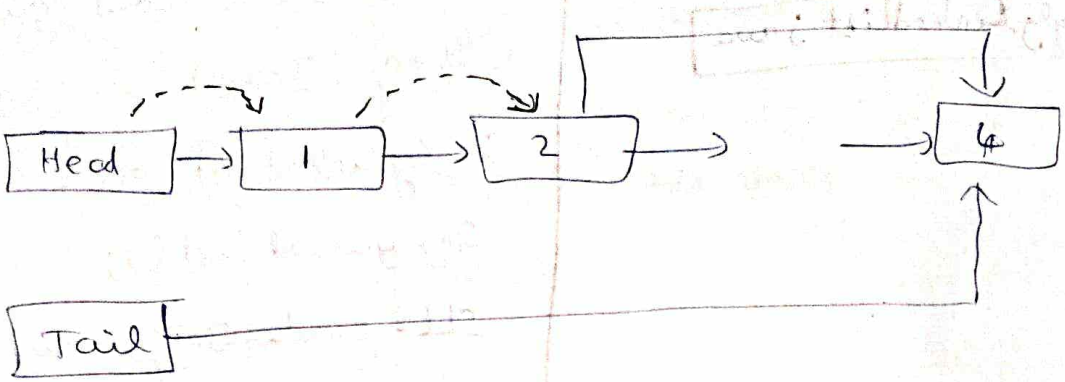
Singly linked list Deletion

① Deleting the first node
② Deleting the given node
③ Deleting the last node.

Deleting the first node.



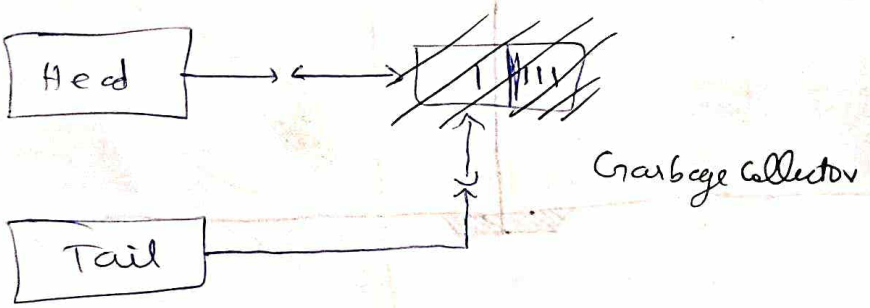
Deleting the given node



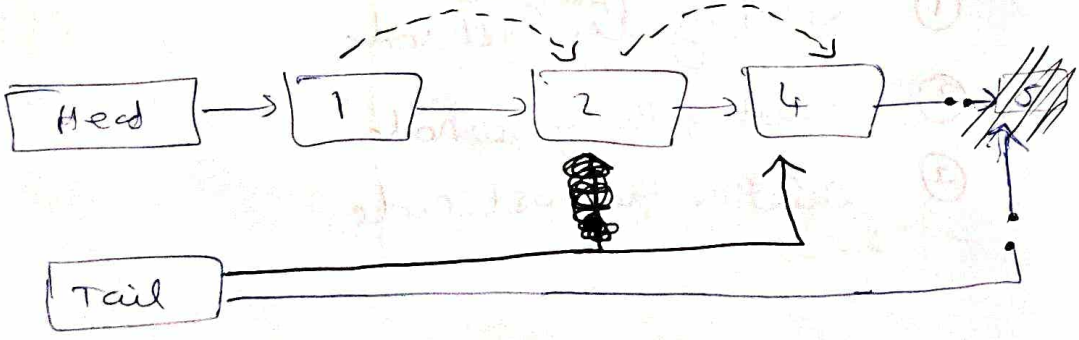
Garbage collector → collect deleted nodes

Deleting the last node

Case:1



Case:2



To create a new node.

```
Node node = new Node();
```

How To Take a Single Value in a String.

```
class SingleValue {

public static void main (String [] args)
{
// byte a = -128;
// System.out.println(a+1);
String c = " Java is one of the interesting language";
System.out.println(c);
String d = c.substring(5,7); // space is also included in
                                sub string
System.out.println(d);
}
}
```

(-128+1)

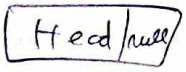
OUTPUT : -127

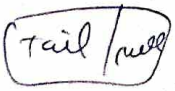
OUTPUT: is

we can represent in two ways

```
class Main {

public static void main ( String args[])
{

String a = "Hellow";
System.out.println ( a.charAt (2));
System.out.println ( a.substring (2,3));
}
}
// output:
// l
// l
```

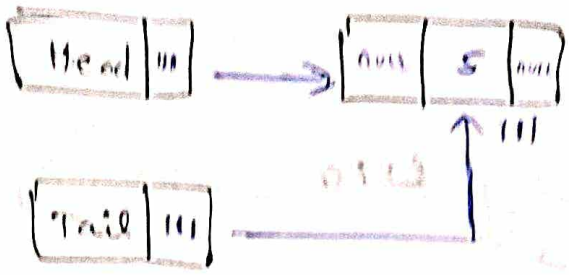
Deleting node





Remaining node → Garbage Collector.

Create Doubly linked list.



reverse traversal → To reverse the Nodes

1 → 2 → 3 → 4

4 ← 3 ← 2 ← 1

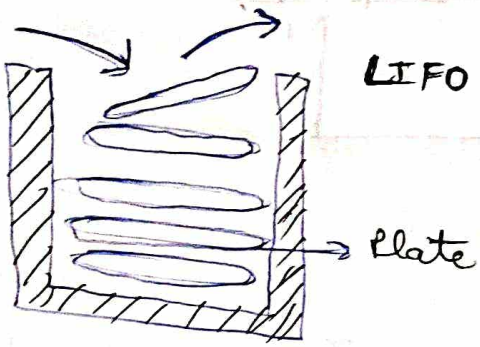
traversal 1 → 2 → 3 → 4

reversetraversal 4 ← 3 ← 2 ← 1

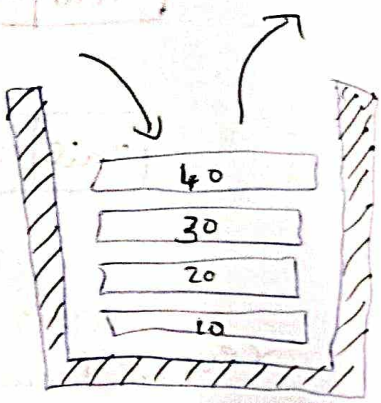
STACK AND QUEUE.

Stack:-

→ Stack is a data structure that stores item in Last in / First out manner.

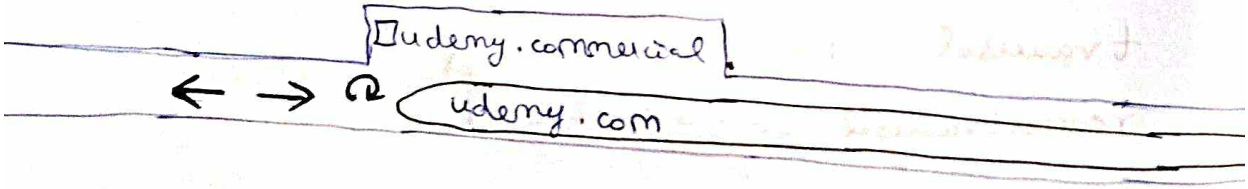


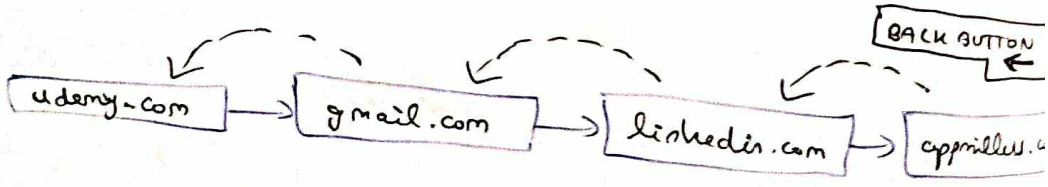
STACK EXAMPLE.

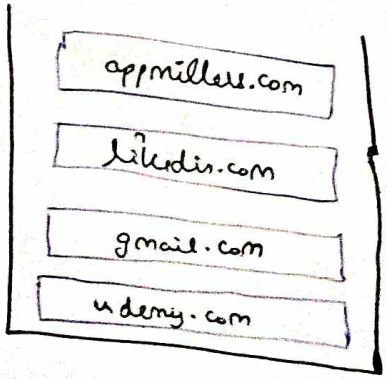


STACK

STACK BEST EXAMPLE :- WEB BROWSER.



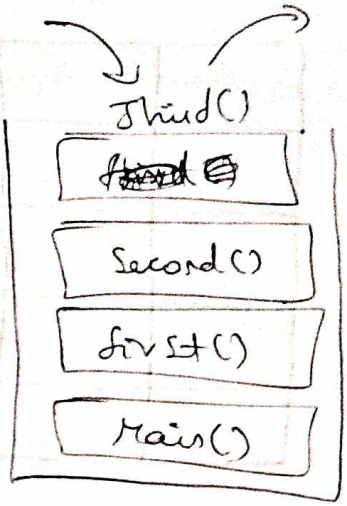




STACK

STACK OPERATION.

- Create stack
- Push
- Pop
- peek
- is Empty
- is Full
- delete stack.



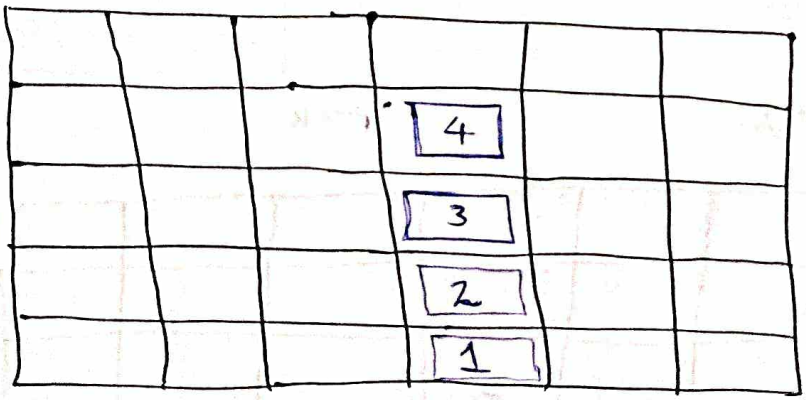
PUSH METHOD:-

// Adding particular element.

Custom Stack = [1, 2, 3, 4]

custom stack.push(4)

Memory.



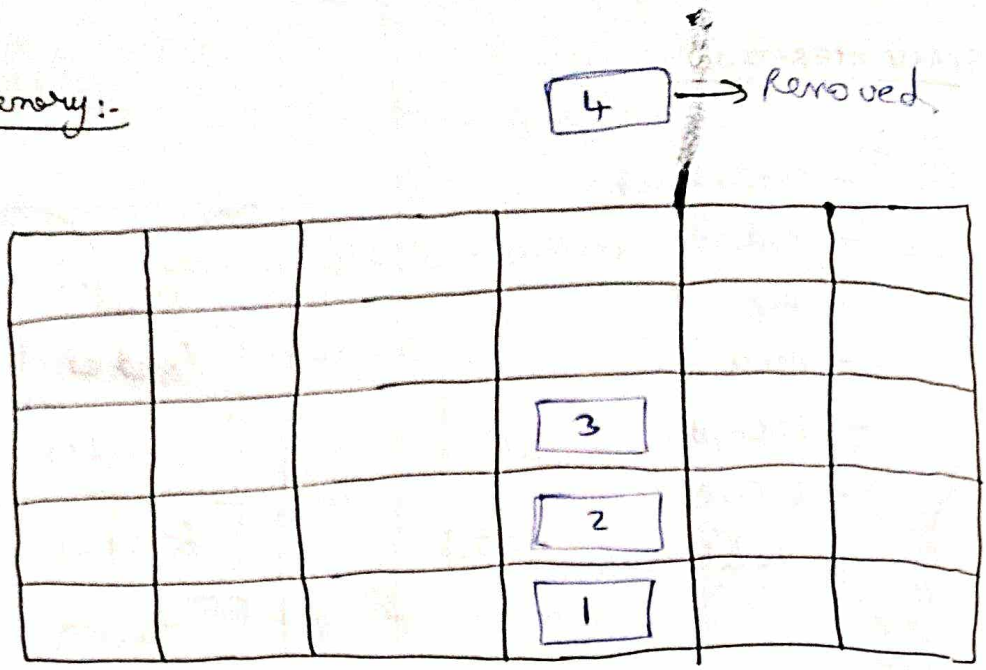
POP METHOD:-

// Removing particular element.

CustomStack = [1, 2, 3]

customStack.pop(4)

Memory :-



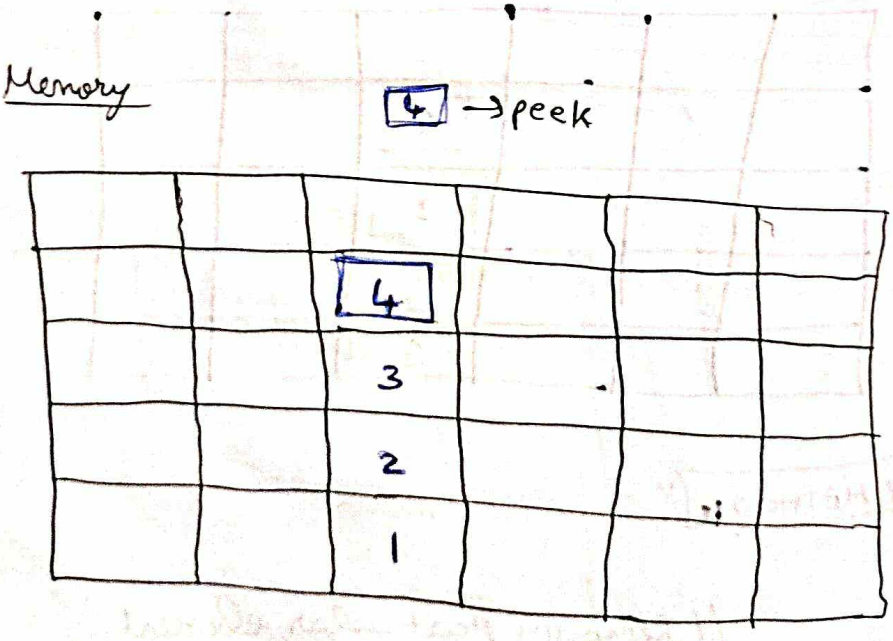
Peek Method.

// To check the Top of the element.

customStack = [1, 2, 3, 4]

customStack.peek(4)

Memory



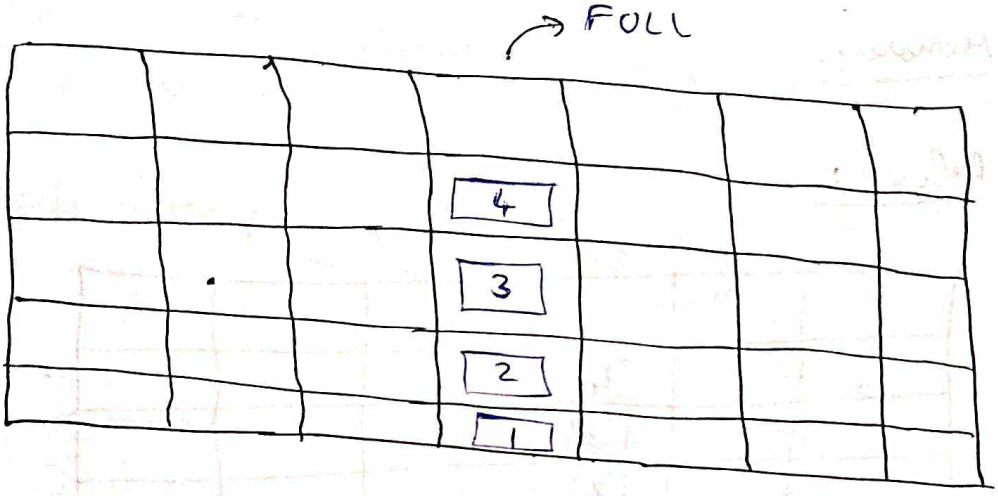
isEmpty Method:

// To check whether stack is empty or Not.

customStack = [1, 2, 3, 4]

customStack.isEmpty() ⟶ False

Memory:-

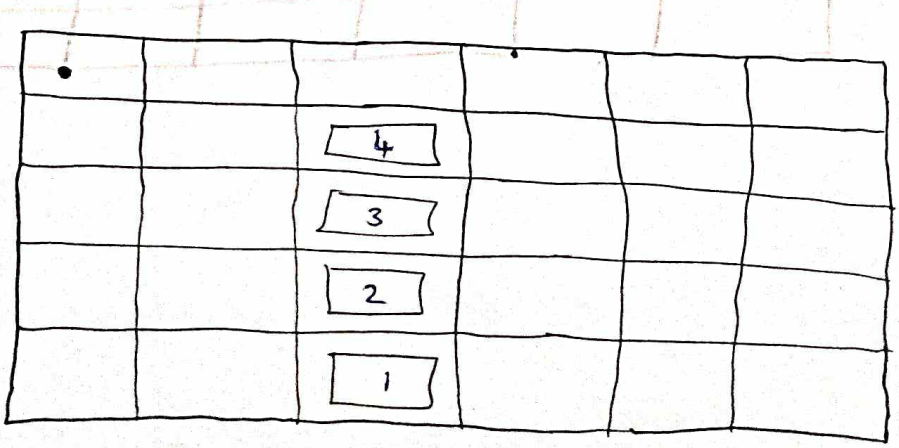


isFull Method:

customStack = [1, 2, 3, 4]

customStack.isFull() ⟶ False

Memory:-



Delete Method :-

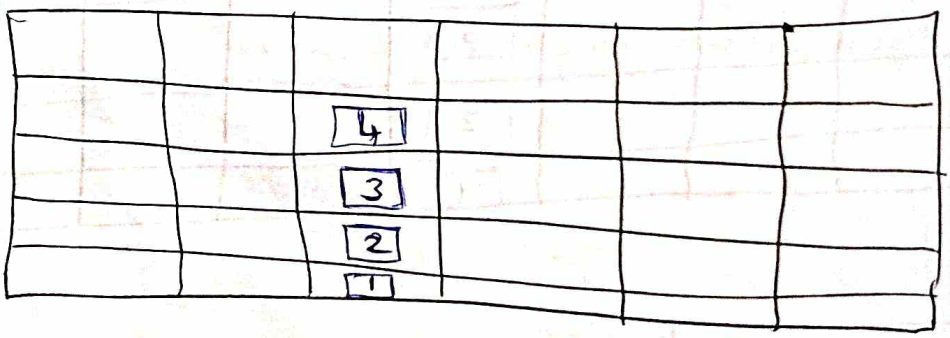
// All elements will get Deleted from the Memory.

customStack = [1, 2, 3, 4]

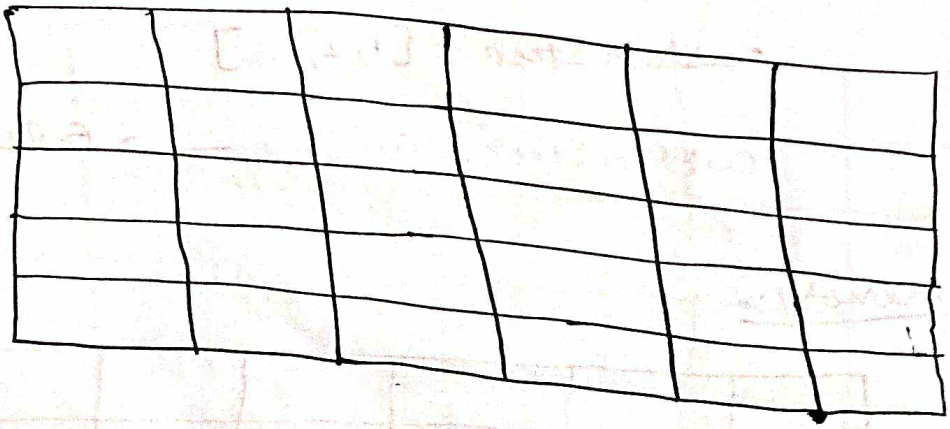
customStack.delete()

Memory :-

Before :



After :



Stack creation - Array Vs Linked list.

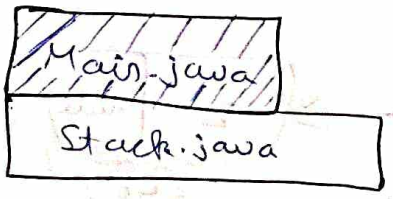
Stack using Array.

- Easy to implement
- Fixed size

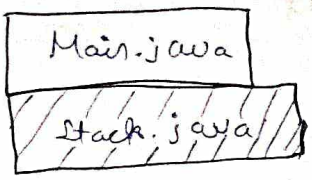
Stack using Linked list.

- Variable size
- implementation is not easy.

Stack Basic program: EASY

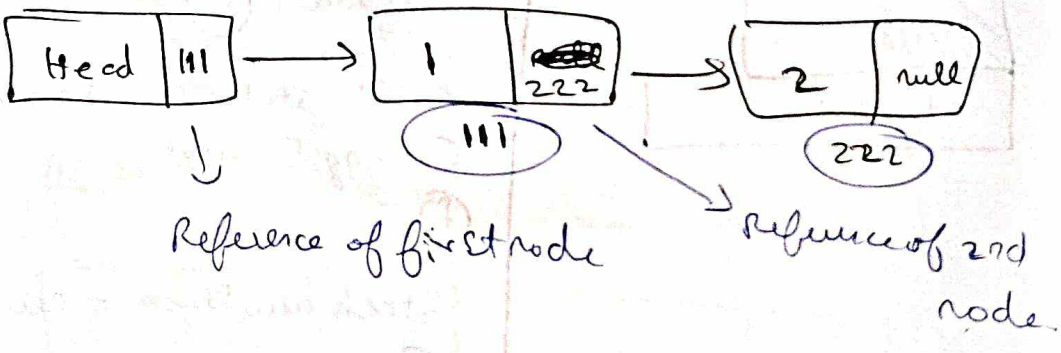



```
class Main {
public static void main
(String [] args)
{

Stack newStack = new Stack(4);

}
}
```




```
public class Stack {
    int[] arr;
    int TopofStack;

    public Stack(int size){
        this.arr = new int[size];
        this.topOfStack = -1;
        System.out.println("The Stack is
        created with size of: " + size);
    }
}
```

Easy Explanation



Overview of program:-

```
import java.util.*;
class Main {

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

// Push Method.

```
public int push() {

}
```

// isFull Method

```
public int isFull() {

}
```

// Pop Method.

```
public int Pop() {

}
```

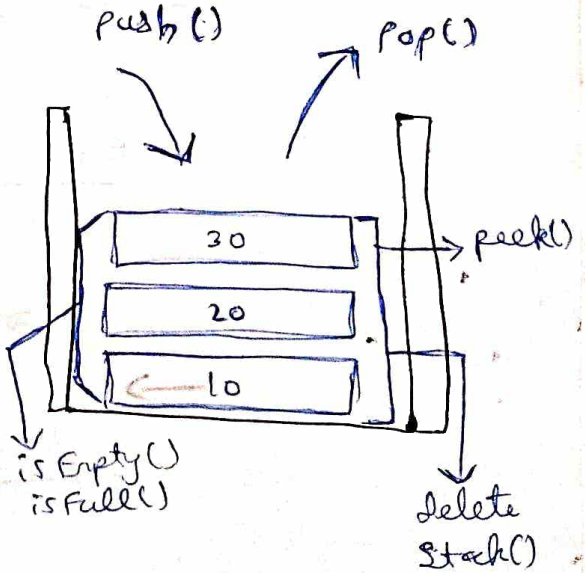
// delete Method.

```
Public void deleteStack()
{

}
```

// peek Method

```
public int peek() {

}
```

Simple diagram for Stack



// is Empty Method

```
Public int isEmpty() {

}
```

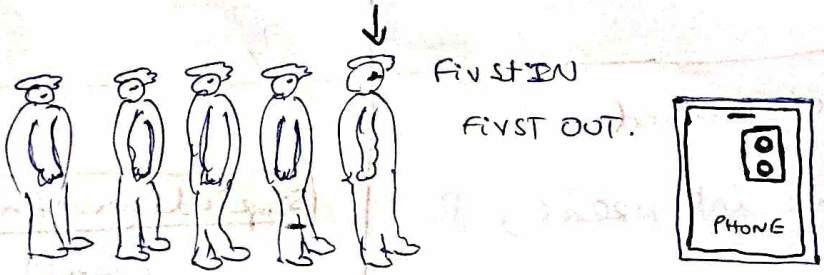
Stack → LIFO functionality

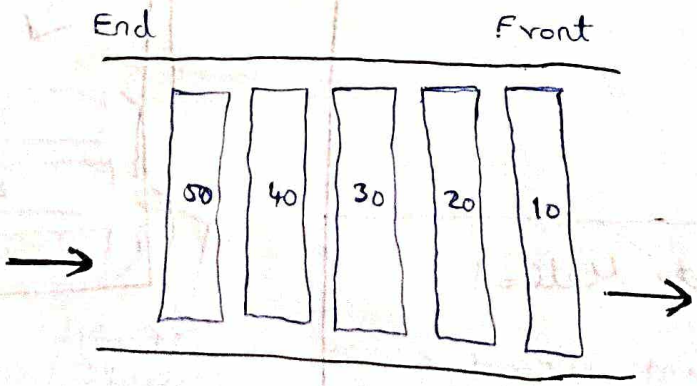
Queue

Queue

→ Queue is a data structure that stores items in a first in / first out manner.

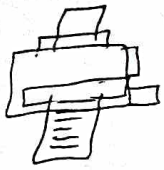
Queue → FIFO functionality





Utilize first coming data first, while others wait for their turn

Printer



→ FIFO System.

PHONE

→ FIFO System.

Incoming call

Queue operation

- Create Queue
- Enqueue
- Dequeue
- Peek
- is Empty
- is Full
- delete Queue

Implementation

1) Array
   - → Linear Queue
   - → Circular Queue
2) Linked List.

Create a Queue

enQueue method

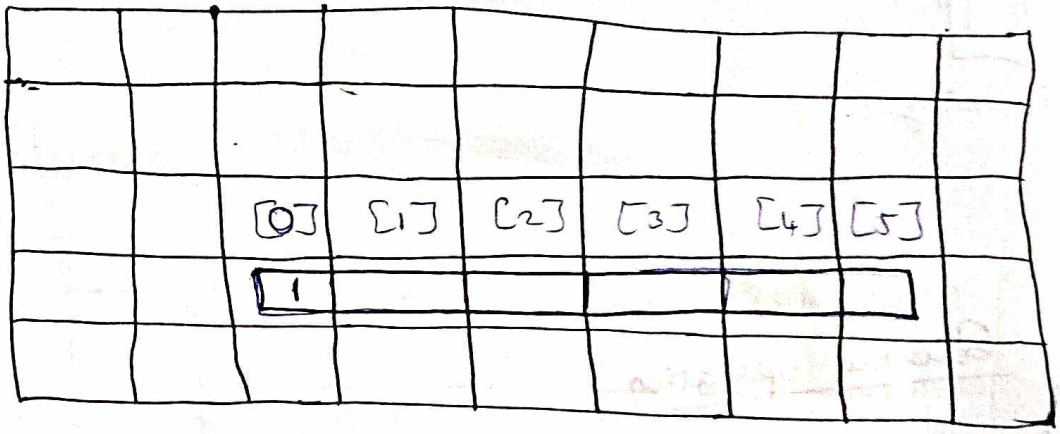
newQueue = Queue(6)

newQueue.enqueue(1)

beginning of Queue = 0

top of Queue = 0

Memory.



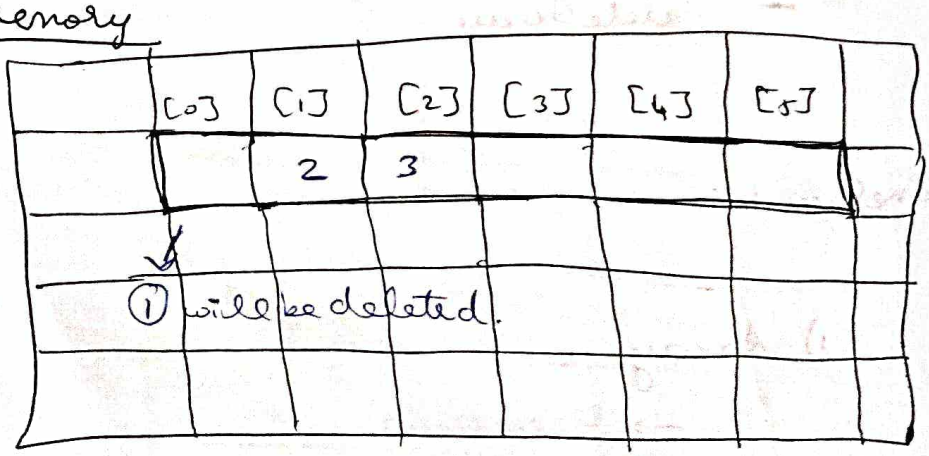
dequeue method.

newQueue.dequeue(1)

beginning of Queue = 1

top of Queue = 2

Memory



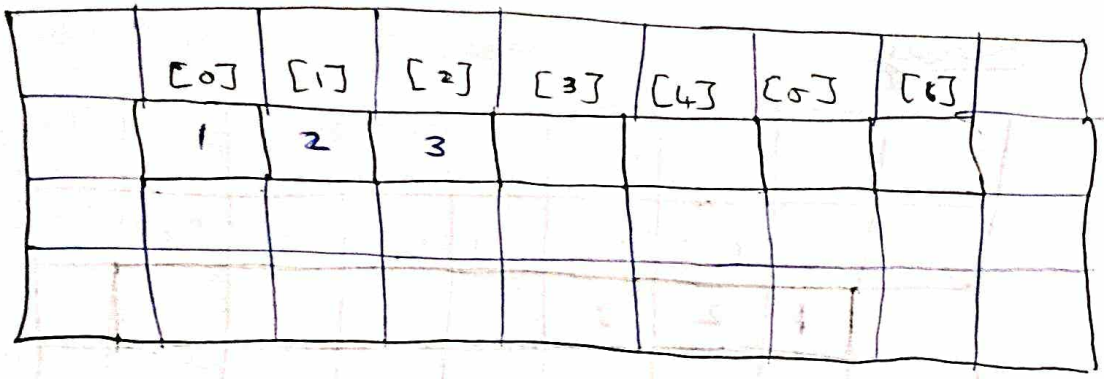
peek Method.

newQueue.peek() ⟶ 1

beginingOfQueue = 0

topOfQueue = 2

Memory



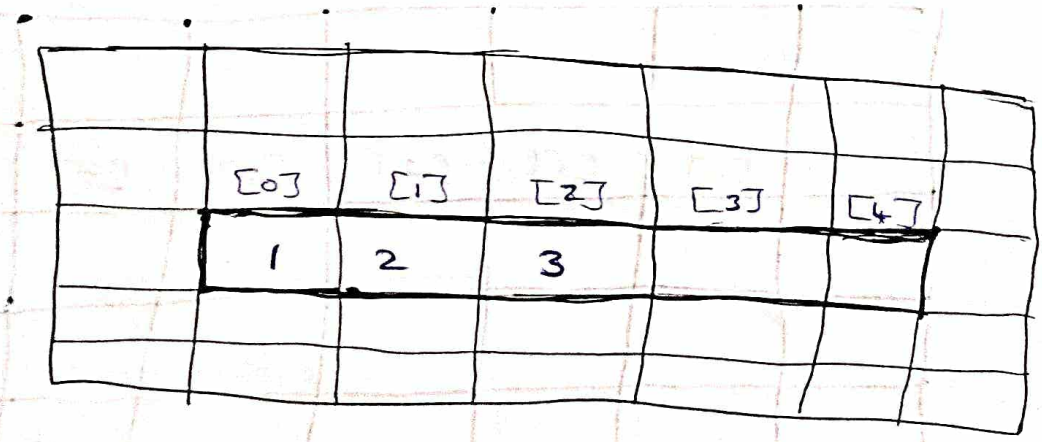
isEmpty Method.

newQueue.isEmpty() ⟶ False

beginningOfQueue = 0

topOfQueue = 2

Memory



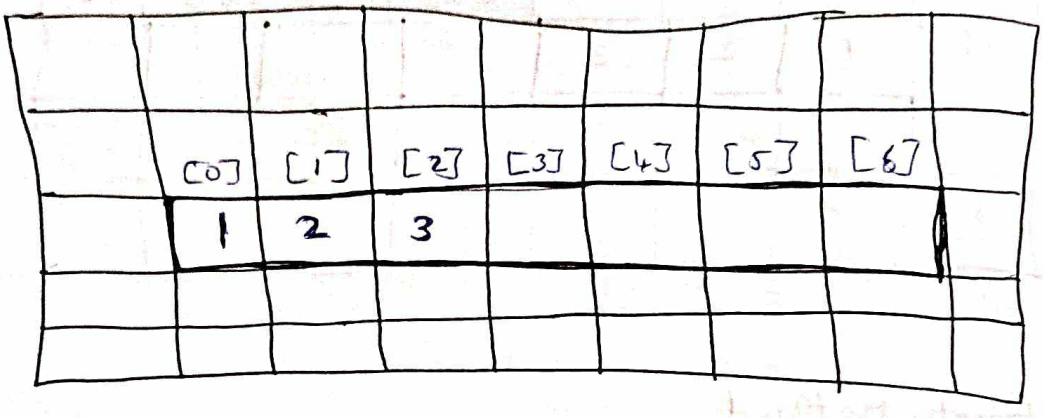
isFull Method

newQueue.isFull(6) ⟶ False

beginning of Queue = 0

top of Queue = 2

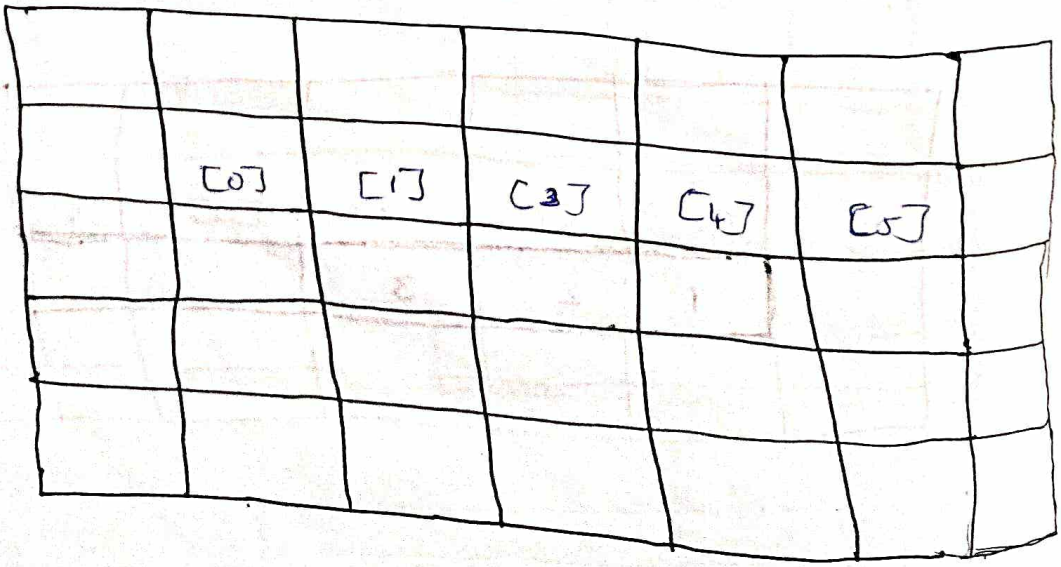
Memory



delete Method.

newQueue.delete()

Memory.



Overview of Program.

```
import java.util.*;
class Main {

public static void main (String [] args) {

// Push Method.
public int enqueue () {

}

// isFull Method.
Public int isFull () {

}

// Pop Method
public int dequeue (){

}

// delete Method.
public void deleteQueue() {

}

// Peek Method
public int peek() {

}

// isEmpty Method
Public int isEmpty () {

}
```

another method:-

beginning Of Queue = 0+1 = 1

top of Queue = 3+1 = 4

(or)

beginning of Queue = 1

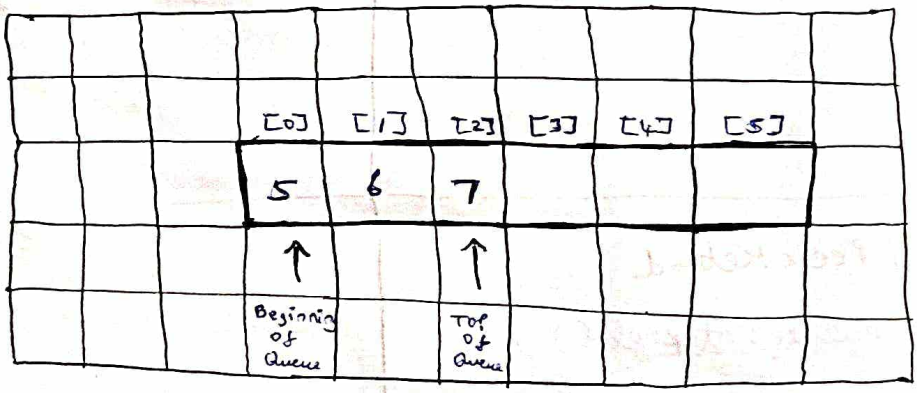
top of Queue = 4

enQueue Method

newQueue.enQueue(7)

beginningOfQueue = 0

top Of Queue = 2

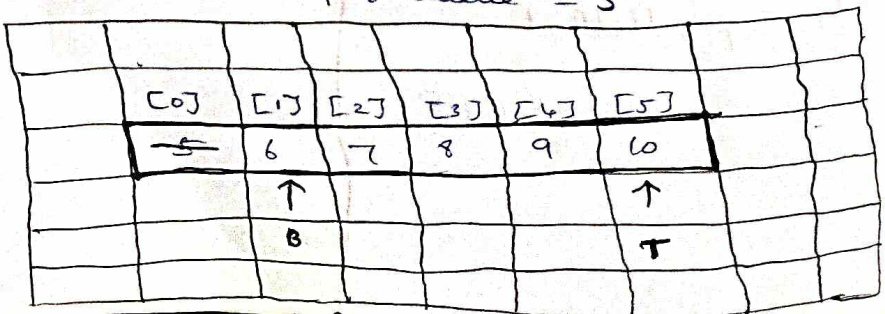


deQueue Method.

newQueue.deQueue() ⟶ 5

beginning of Queue = ~~0~~ 1

top of Queue = 5



② Method

New Queue. deQueue () ⟶ 6

beginning Of Queue = ~~1~~ 2

top Of Queue = 5

	[0]	[1]	[2]	[3]	[4]	[5]	
	~~5~~	~~6~~	7.	8	9	10	
			↑			↑	
			B			T	

③ Reverse Method.

New Queue . reQueue ()

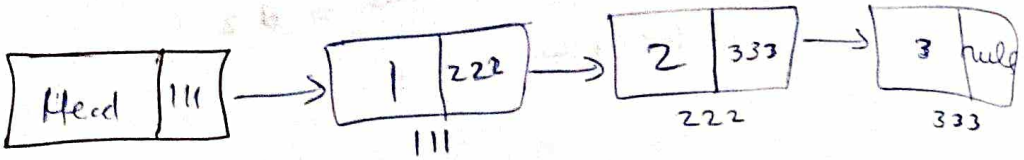
beginning Of Queue = 2

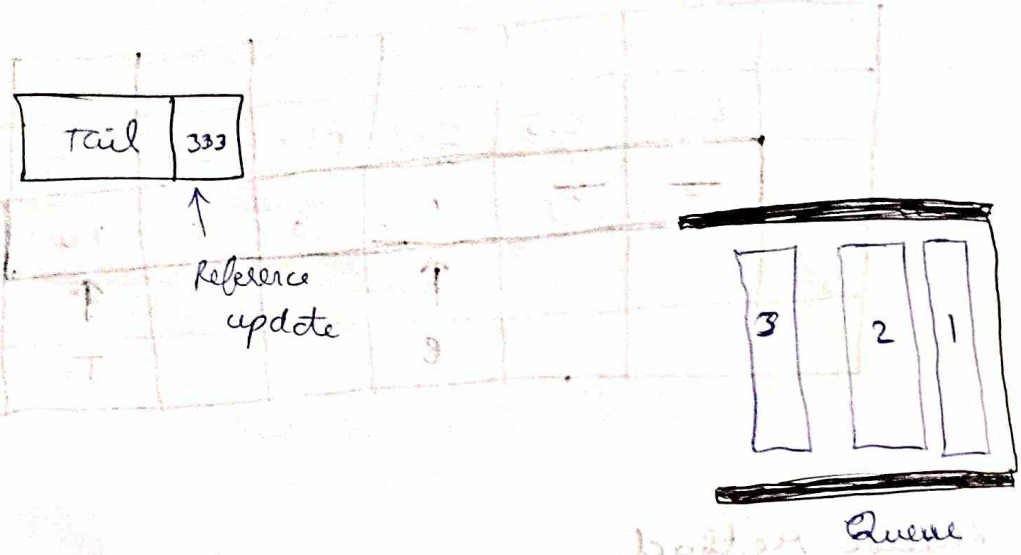
top Of Queue = 0

[0]	[1]	[2]	[3]	[4]	[5]
1	~~6~~	7	8	9	10
↑		↑			
T		B			

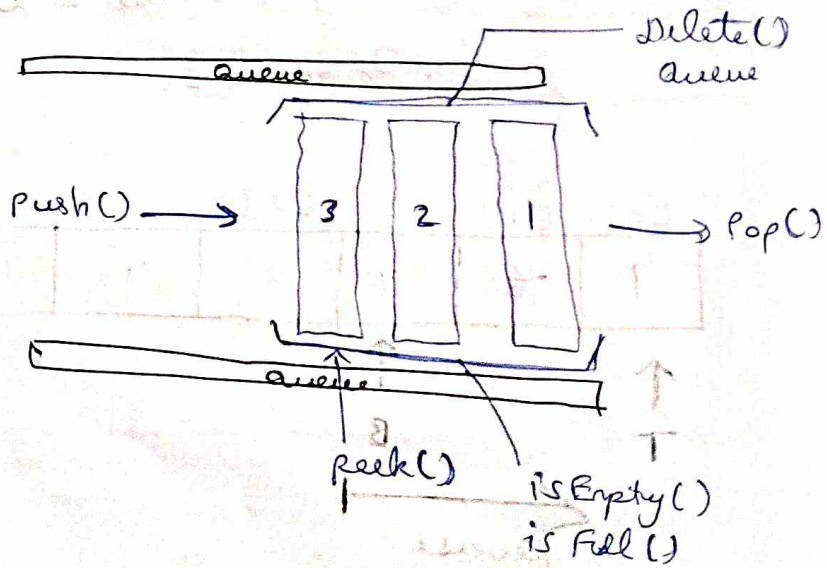
Reverse

Creation of Queue





Simple digram of Queue

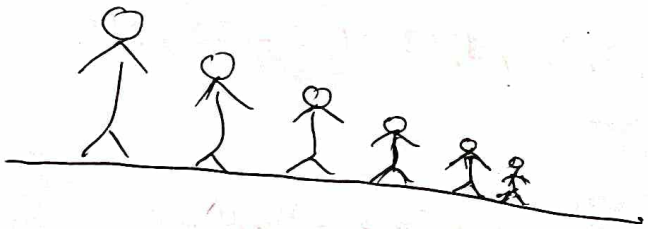


Example for Stack → Web Browser

Example for Queue → Printer

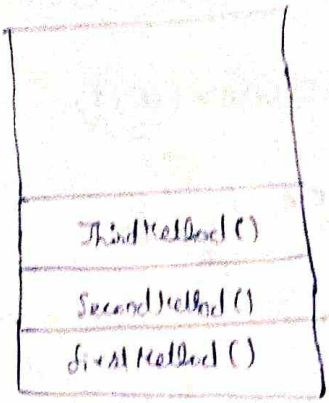
Recursion

Recursion :- a way of solving a problem by having a function calling itself



Problem gets smaller and smaller.

```
Static void firstMethod () {
   SecondMethod ();
   System.out.println("I am in the firstMethod");
  }
Static void SecondMethod () {
   ThirdMethod ();
   System.out.println ("I am in the SecondMethod");
  }
Static void ThirdMethod() {
   System.out.println("I am in the ThirdMethod");
  }
```



STACK MEMORY

Fibonacci Numbers.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

Step 1: Recursive Case - the flow

5 = 3+2

Step 2: Base Case

Step 3: unintentional case

Recursion Case — The flow.

10 = 10/10 = 1 and reminder = 0

54 = 54/10 = 5 and reminder = 4

$112 = 112/10 = 11$ and Reminder $= 2$

$11/10 = 1$ and Reminder $= 1$

Recursion Questions

① How to find GCD (Greatest Common Divisor) of Two numbers using recursion.

Sol: $gcd(8, 12) = 4$

$8 = 2 * 2 * 2$

$12 = 2 * 2 * 3$

Euclidean algorithm

gcd(48,18)

Step 1: 48/18 = 2 remainder 12

Step 2: 18/12 = 1 remainder 6

Step 3: 12/6 = 2 remainder 0

Decimal to Binary. (Important - ✗)

How to convert a number from Decimal to Binary using recursion.

Step 1: Divide a number by 2

Step 2: Get the integer quotient for the next iteration

Step 3: Get the remainder for the binary digit

Step 4: Repeat the steps until the quotient is equal to 0.

13 to binary.

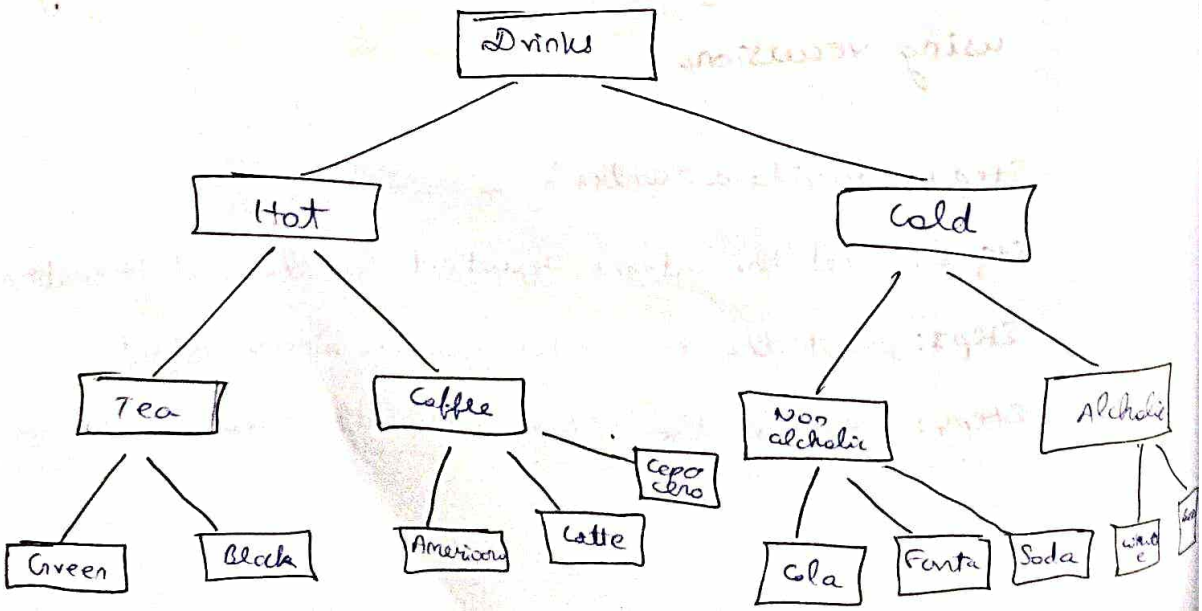
Divide by 2	Quotient	Remainder	
13/2	6	1	①
6/2	3	0	②
3/2	1	1	④
1/2	0	1	⑧

8 4 2 1

13 → 1 1 0 1

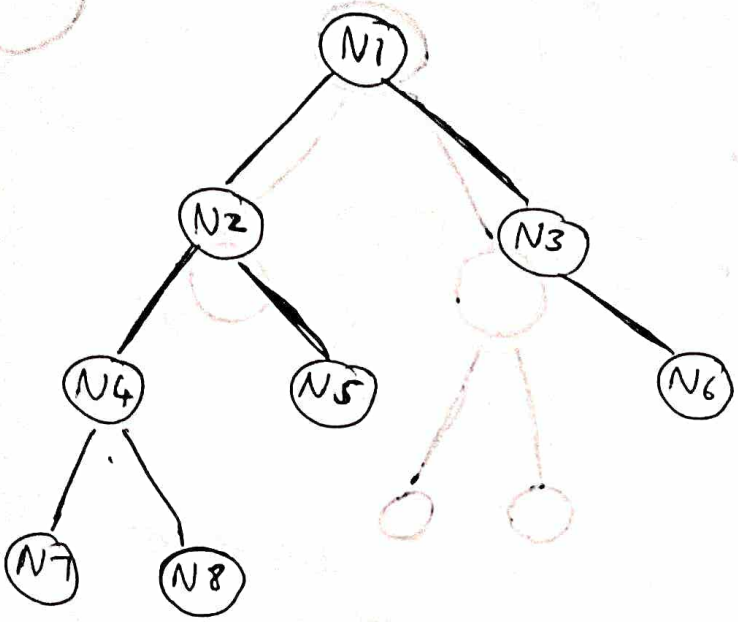
Tree / Binary Tree.

— A Tree is a non linear data structure with hierarchial relationships between its elements without having any cycle, it is beasically from a real life tree.



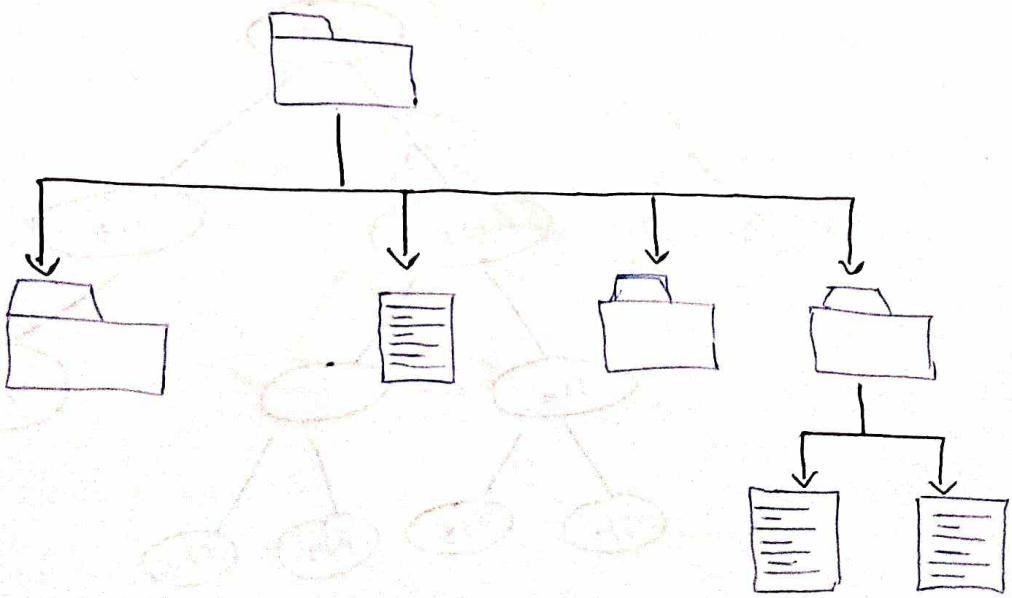
Every Node ⟶ Two components

data and link



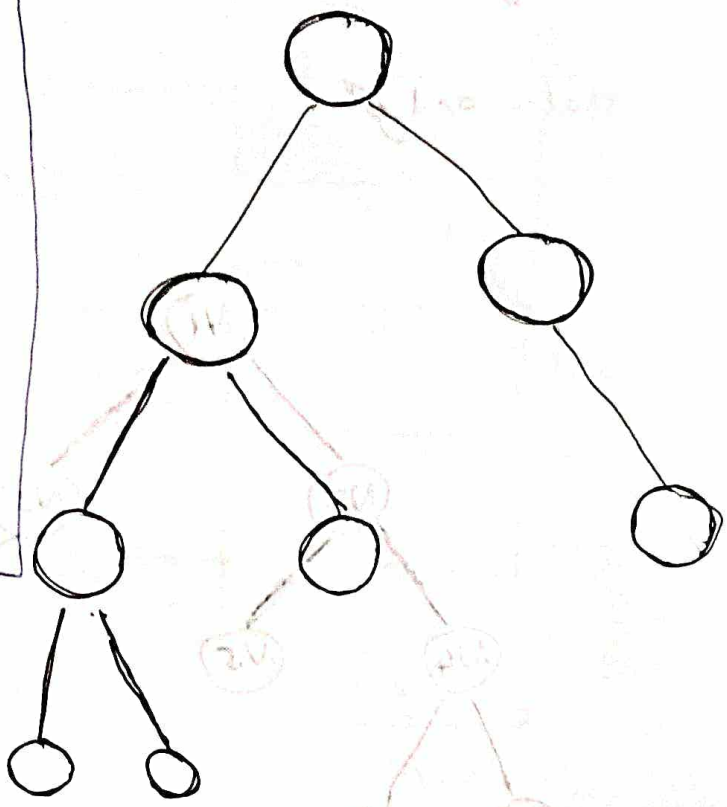
Tree:

- Quick and easier access to data
- folder, HTML / XML data



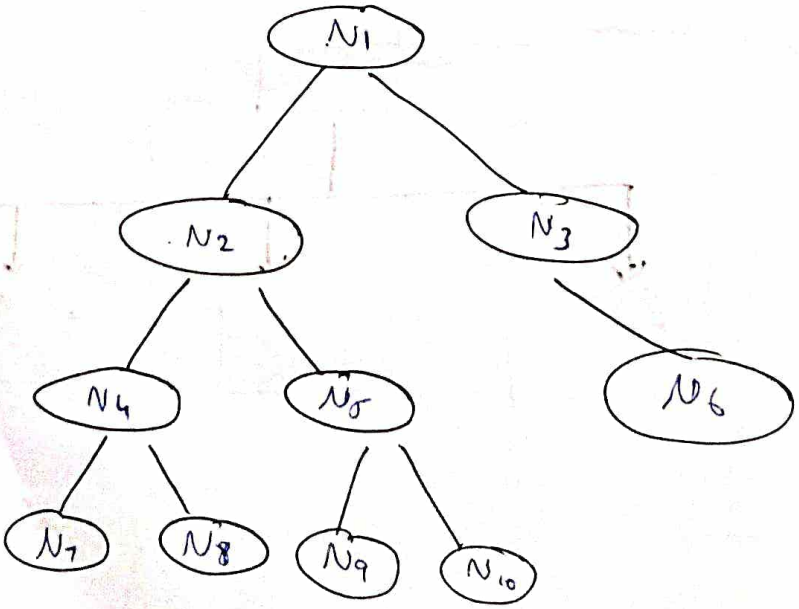
Tree Terminology

- Root
- Edge
- Leaf
- Sibling
- Ancestor
- Depth of node
- Height of node
- Depth of tree
- Height of tree



Binary Tree:

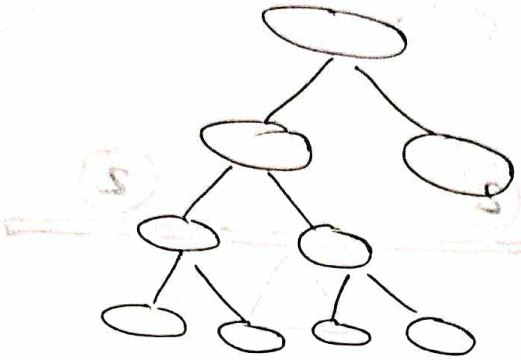
— It should not have more than two children.



Types of Binary Tree.

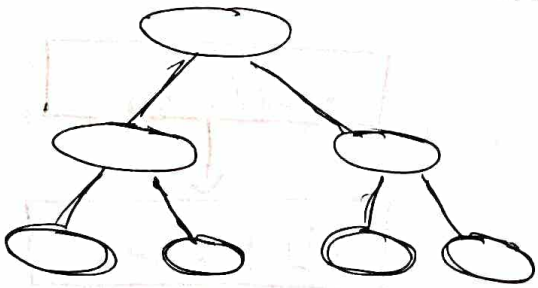
① Full Binary Tree.

↳ All Nodes have Two childes or No Empty Node.

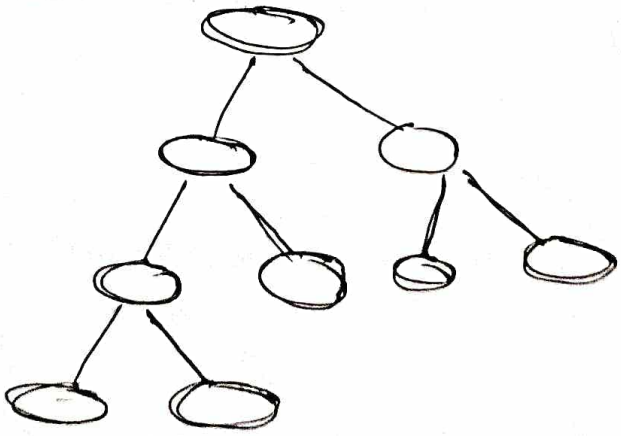


② Perfect Binary Tree.

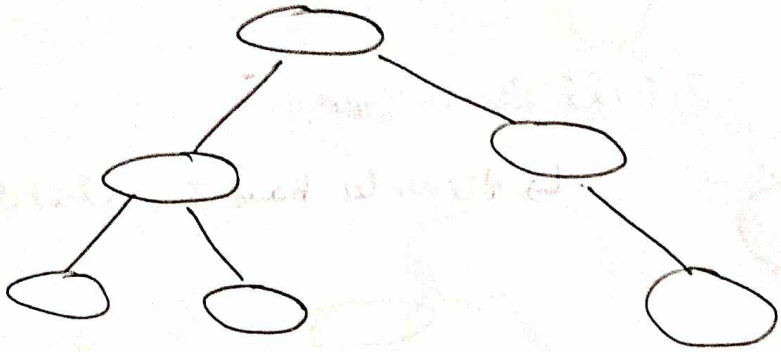
↳ All the leaf Nodes have Two children.

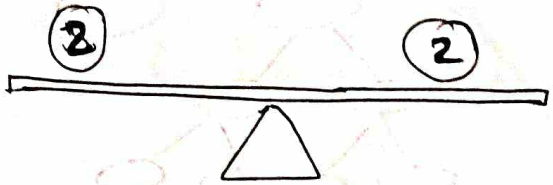


③ Complete Binary Tree.

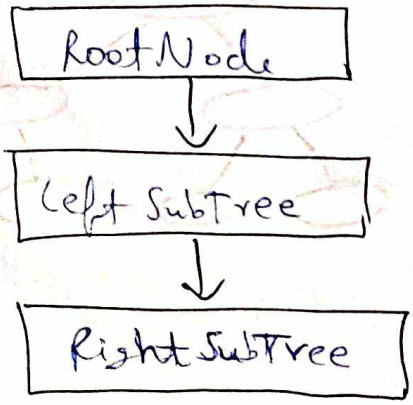


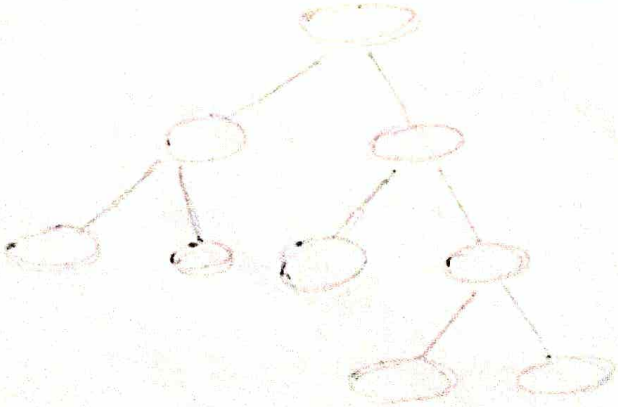
④ Balanced Binary Tree.



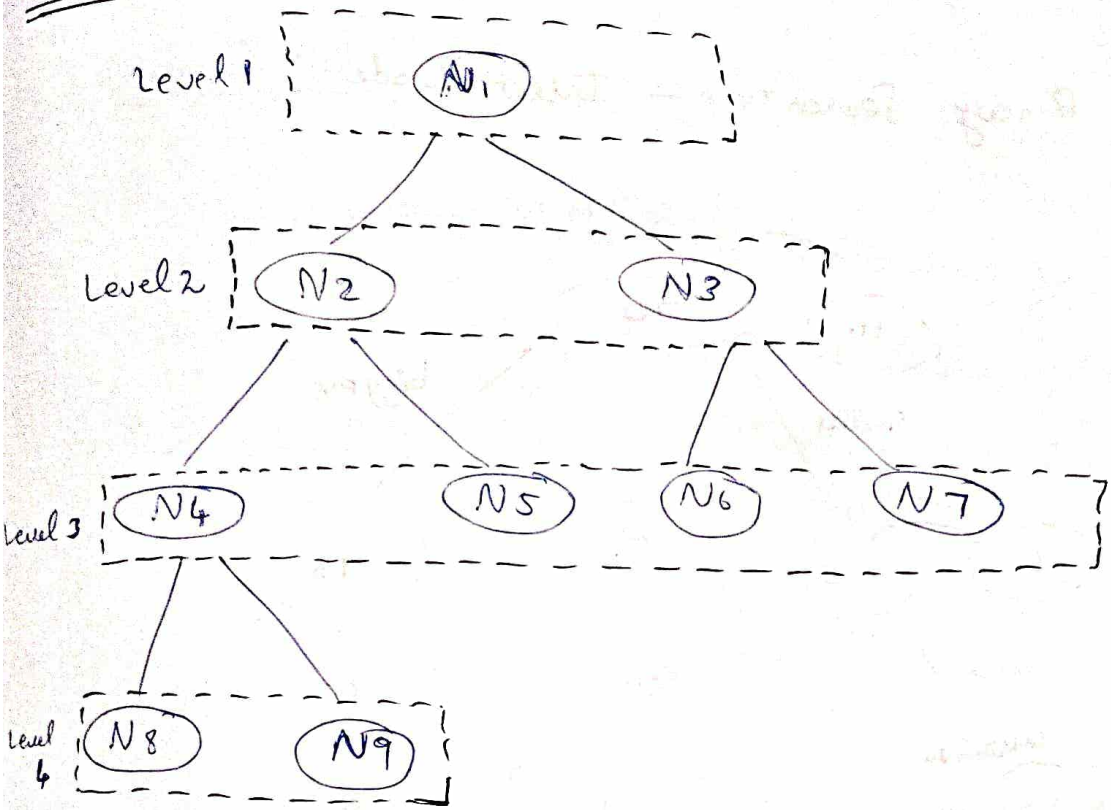


Binary Tree





Level order :-



N1 → N2 → N3 → N4 → N5 → N6 → N7 → N8 → N9

To insert a node ----→ A Node should be null.

What is Binary Search Tree :-

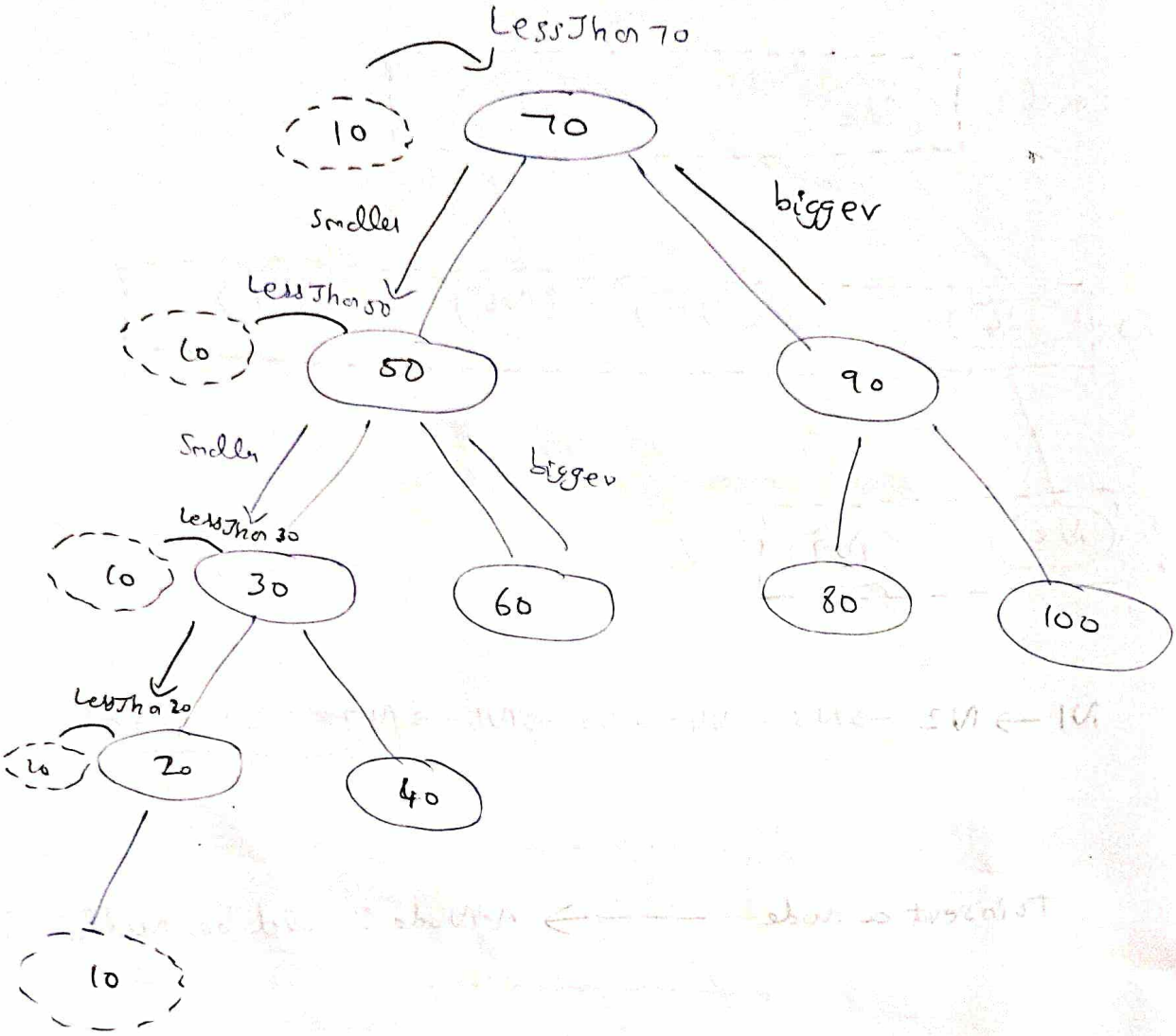
→ The value of Node is greater than its parent node's value.

→ It performs faster than Binary Tree

Operations on BST.

→ Creation of BST
→ Insertion of a Node
→ Deletion of a Node
→ Search for a value
→ Traverse all Nodes
→ Deletion of BST.

Binary Search Tree – Insert a Node

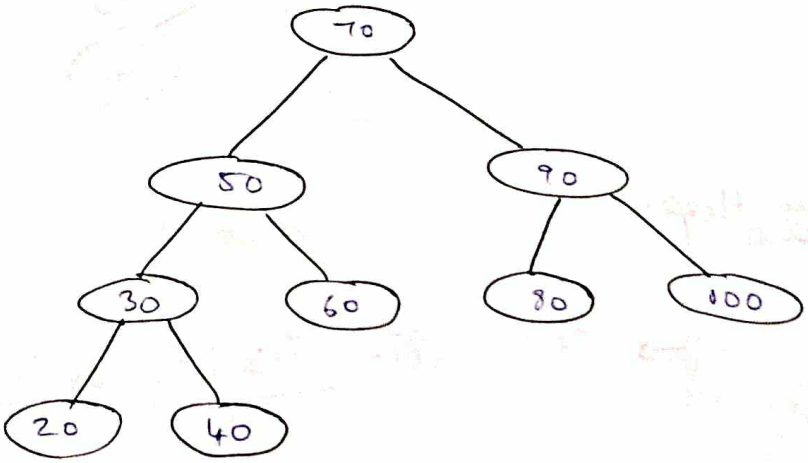


Extra one node added.

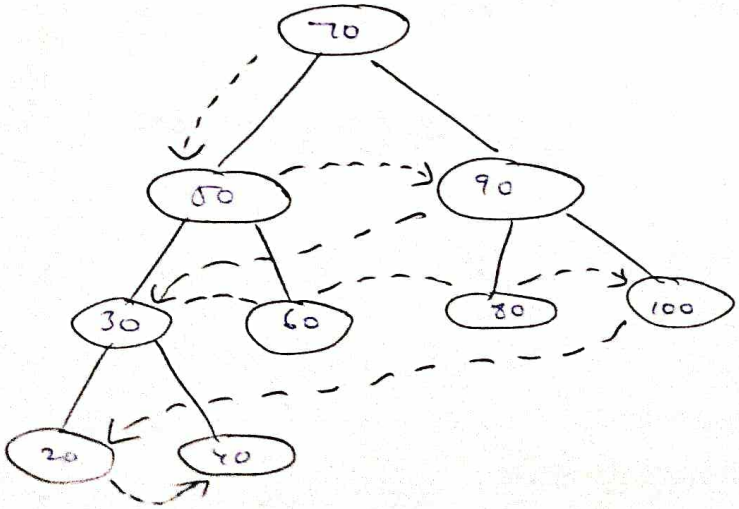
Example: 95, 60, 75, 100.

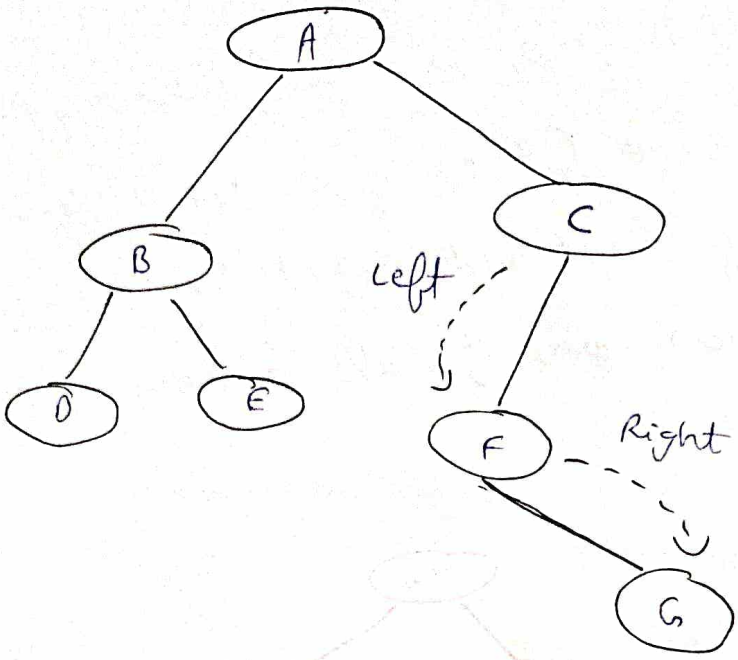
AVL Tree

→ AVL Tree is a self Balancing Binary Search Tree (BST) where the difference between height of left and Right Subtree cannot be more than one for all nodes.



SELF BALANCING

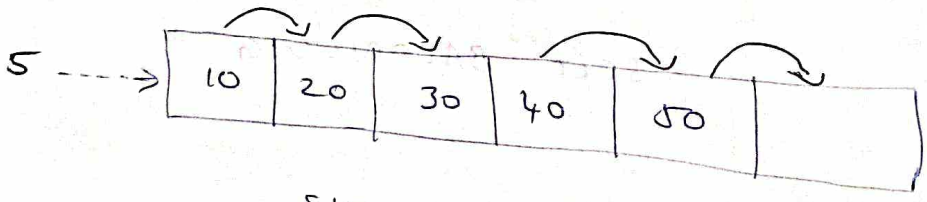




Binary Heap:

→ It is either Min Heap or Max Heap

Store the number in a Sorted Array



SHIFTING METHOD.

Types of Binary Heap

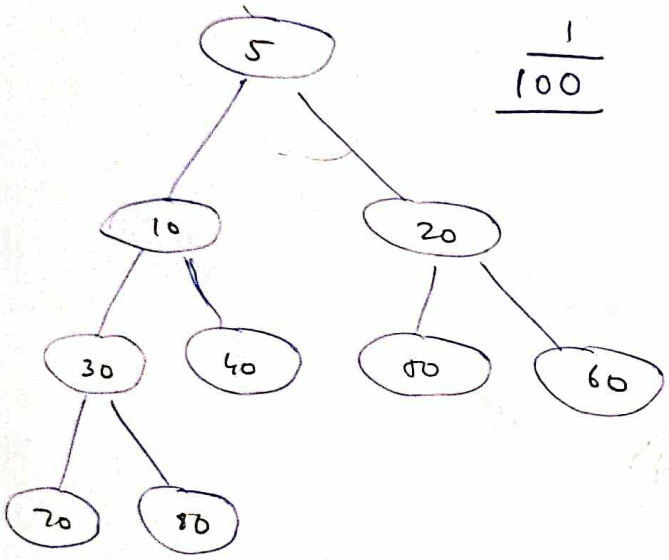
1) Minimum Heap
2) Maximum Heap

```
 6          5
0110       0101
1001       1010
----       ----
1111       1111
   1          1
-----      -----
10000      10000
```

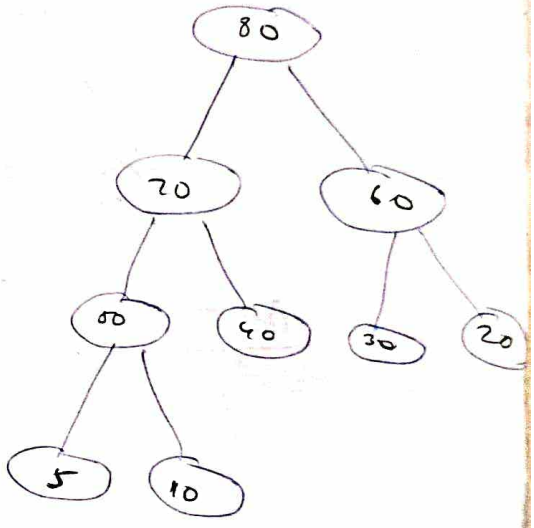
```
 1
0101
 1
----
0110
1001
----
1111
```

Minimum Heap

```
 10
 01
 --
 11
  1
---
100
```



Maximum Heap



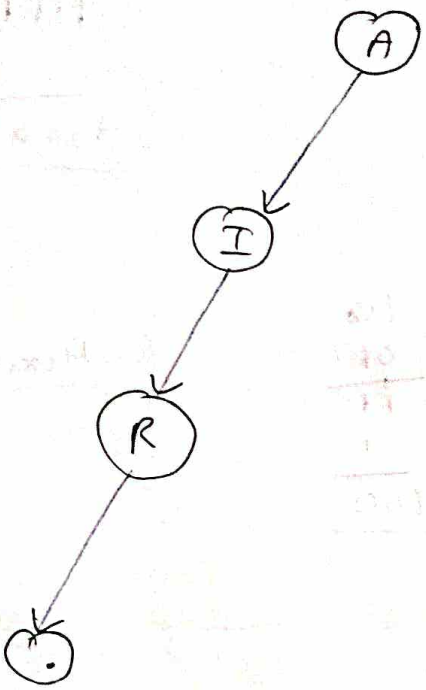
Trie

→ A Trie is a Tree based data structure that organize information in a hierarchy.

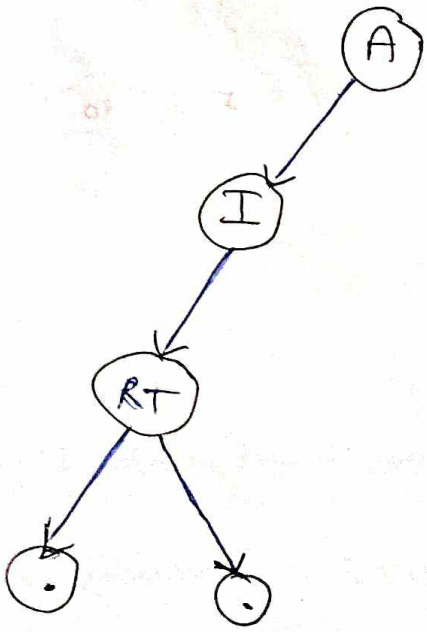
→ Every node stores a link of next child.

Example

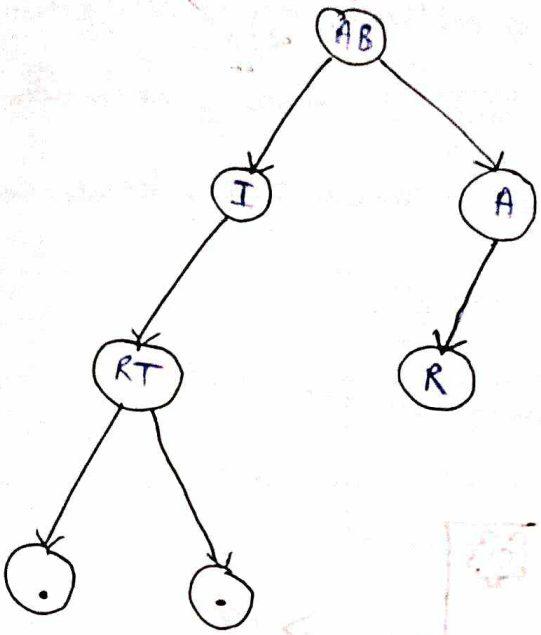
AIR



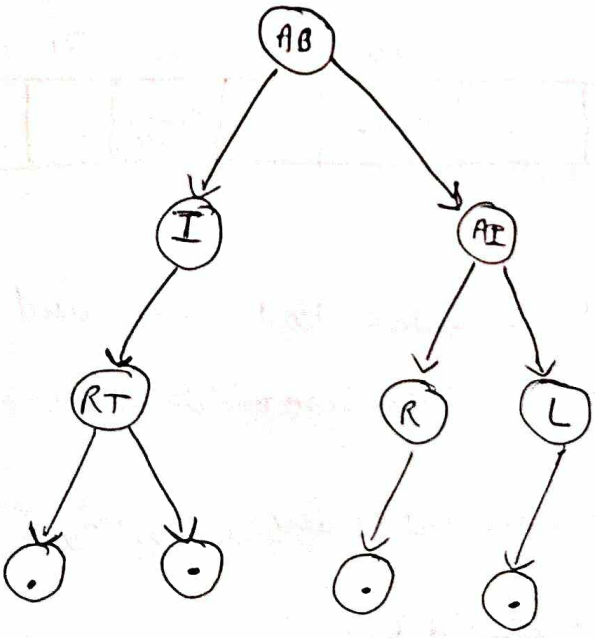
AIT



BAR



BIL



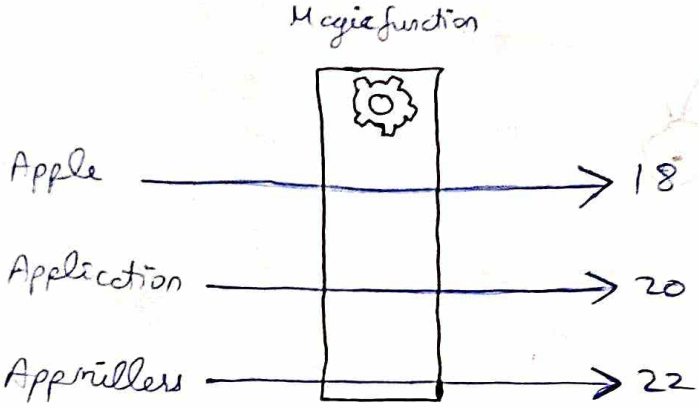
Application of Trie

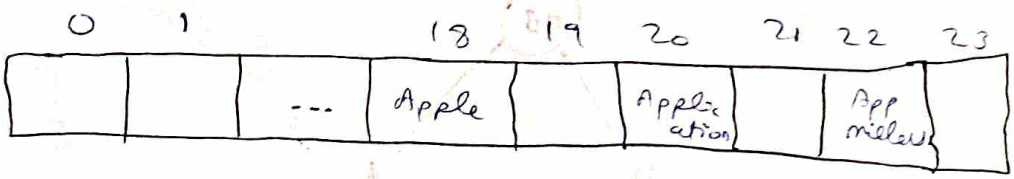
→ Spelling check

→ Auto completion

Hashing

→ Hashing is a method of sorting and indexing data. The idea behind hashing is to allow large amounts of data to be indexed using keys commonly created by formulas.



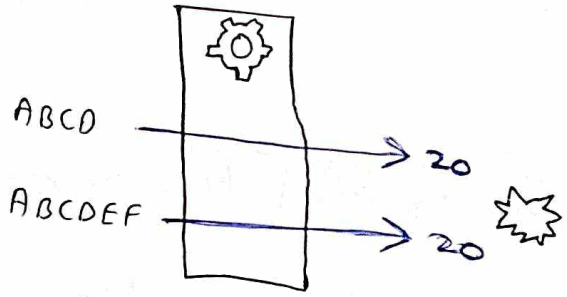


Hash function: It is a function that can be used to map of arbitary size to data of fixed size.

Hash Value: A value that is returned by Hash function

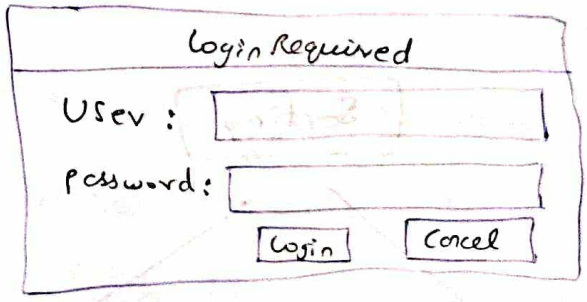
Hash Table: It is a data structure which implements an associative array abstract data type, a structure that can map keys to values.

Collision : A collision occurs when two different keys to a hash function produce the same output.



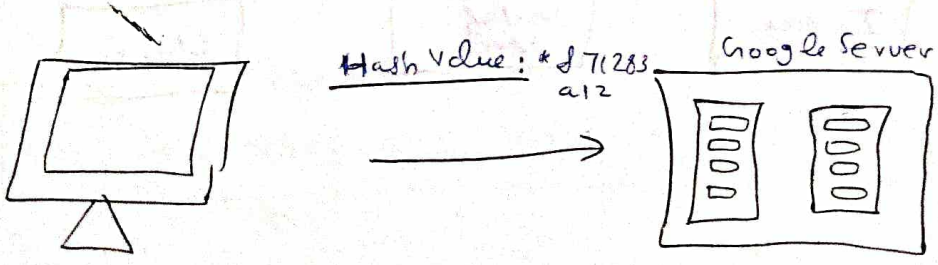
Practical use of Hashing

Password Verification



Real life example.

Personal Computer



Login : abc@gmail.com
Password : 123456

Password stored here

→ Hackers couldn't able to hack password by using hash value.

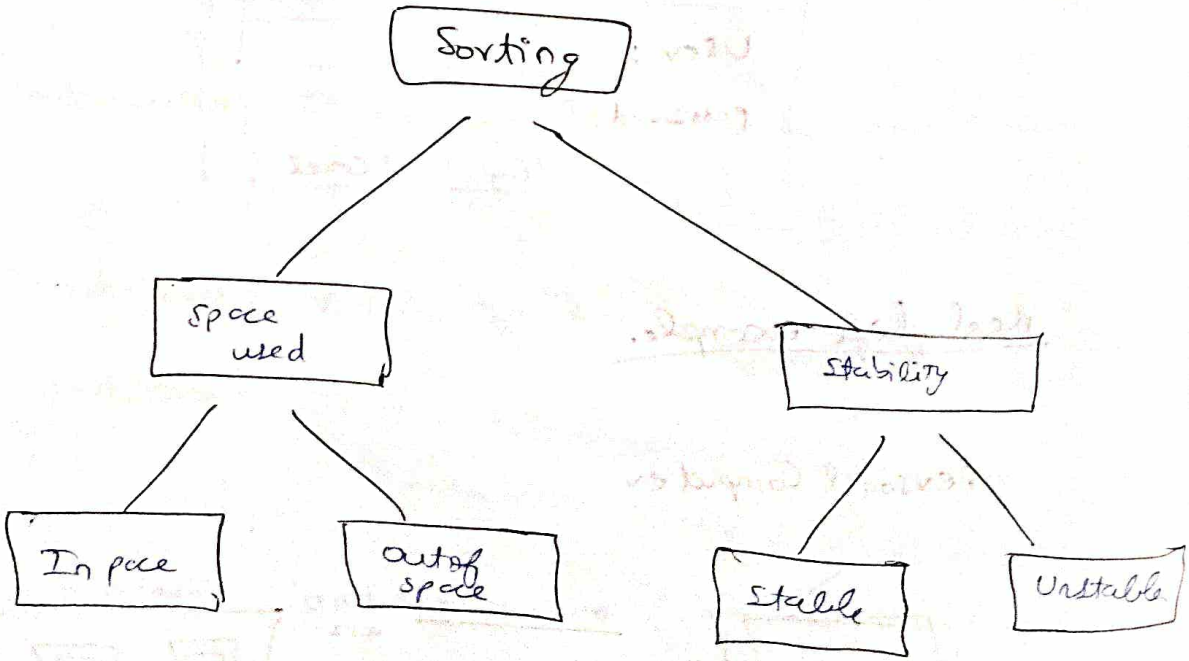
Password : 123456 ⟶ Hashvalue : *8712B3*A12

Sorting :-

→ Sorting refers to arranging data in a particular form : either assending or descending

Example : Microsoft Excel., Amazon (Price Sorting)

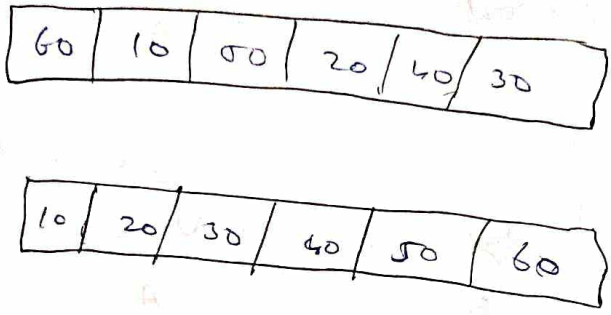
Types of Sorting :-



space used

In place Sorting :- Sorting algorithm which does not require any extra space for sorting

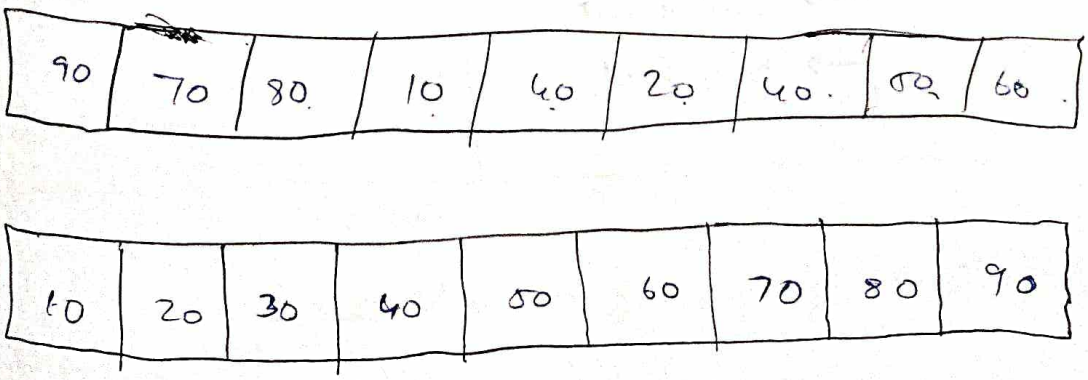
Example : Bubble Sort.



Not place Sorting :- Sorting algorithm which requires an extra space for sorting

Example : Merge Sort.

Stable Sorting :- If a Sorting after sorting the contents does not change the sequence of similar content in which they appear, then this sorting is called stable sorting.



Example : Insertion Sorting.

Unstable Sorting :-

If a Sorting algorithm after Sorting the content changes the sequence of similar content in which they appear, then it is called unstable Sort.

Example : Quick Sort.

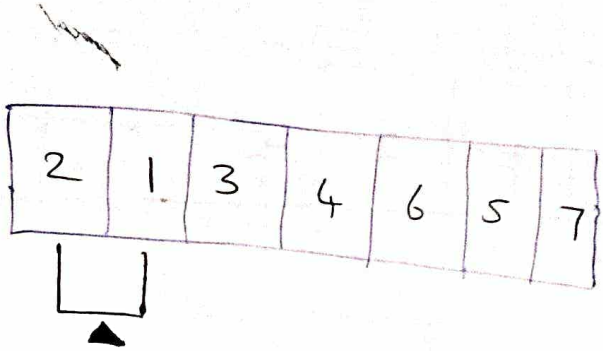
	Name	Age
→	A	6
	C	7
	D	7
→	B	6
	E	7

Name	Age
A	6
B	6
C	7
D	7
E	7

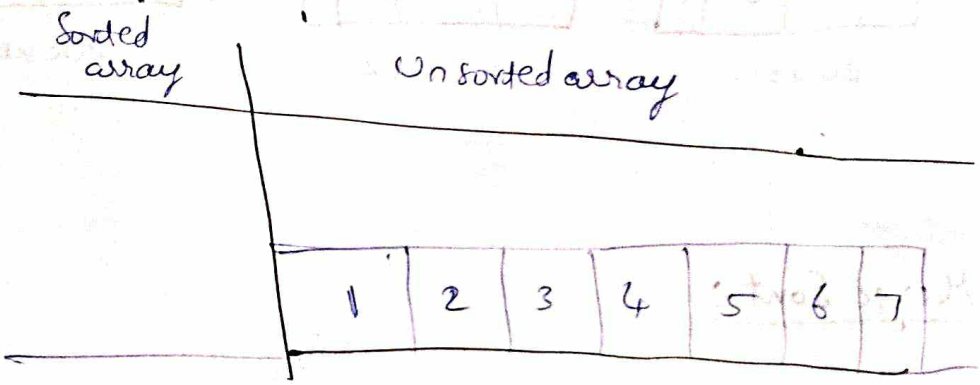
Sorting Algorithm.

→ Bubble Sort
→ Selection Sort
→ Insertion Sort
→ Bucket Sort
→ Merge Sort
→ Quick Sort
→ heap Sort.

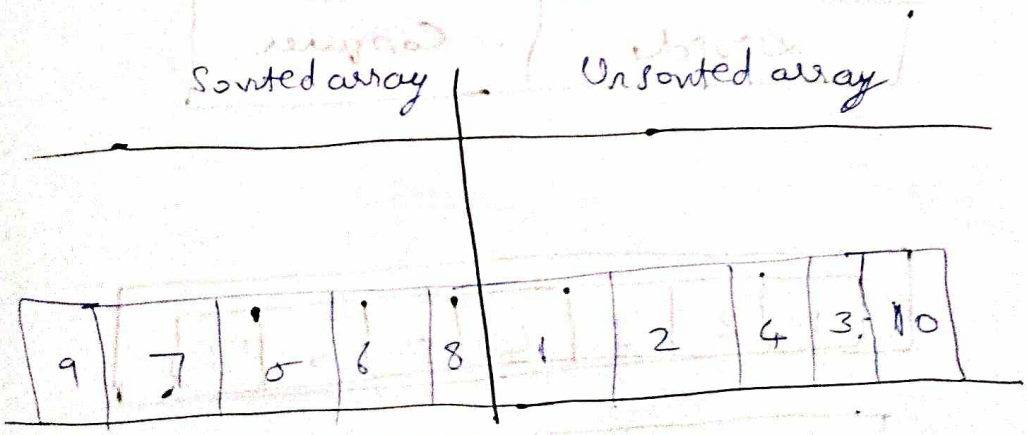
Bubble Sort :-



Selection Sort :-

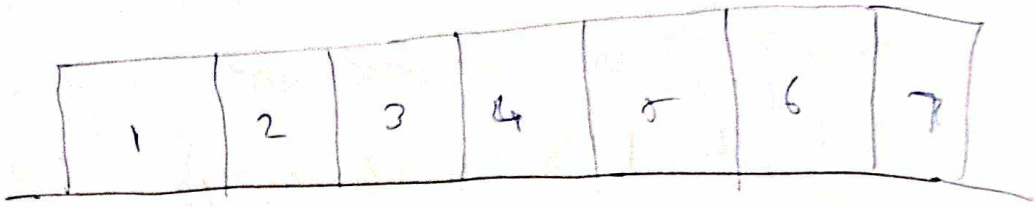


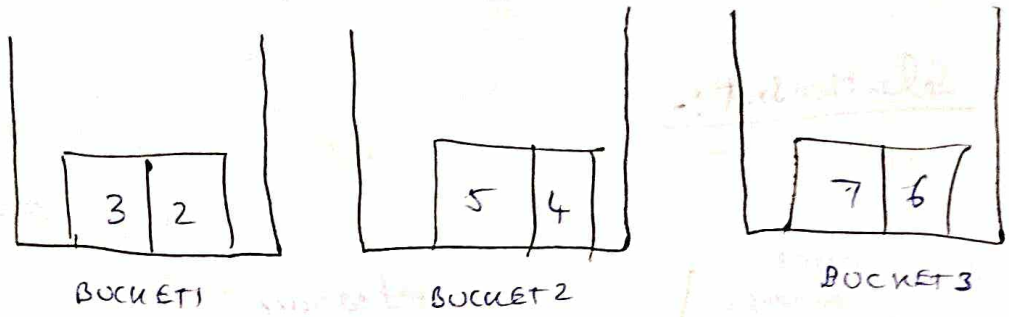
Insertion Sort.



→ Divide the given array into two parts

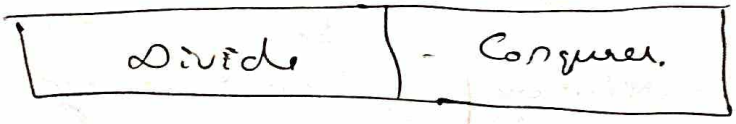
Bucket Sort:

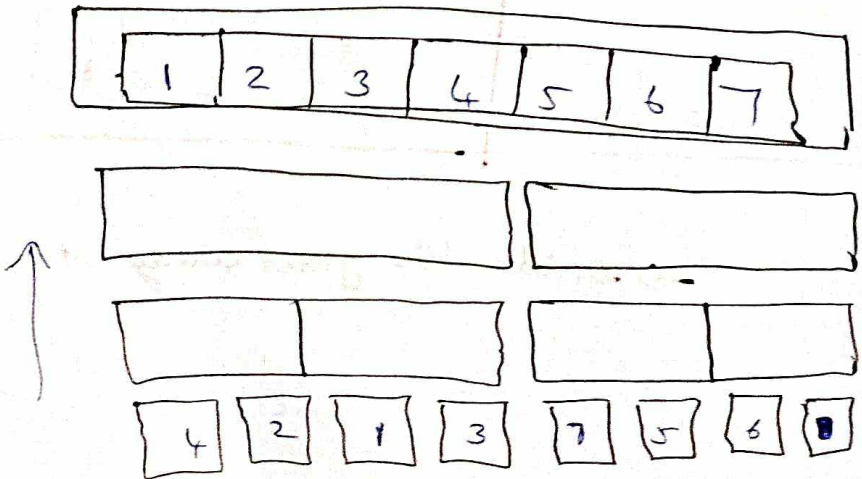




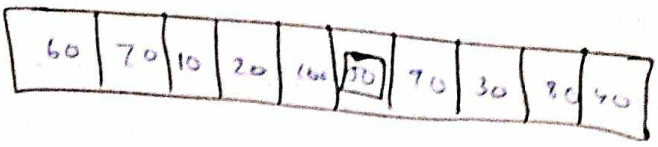
Merge Sort:

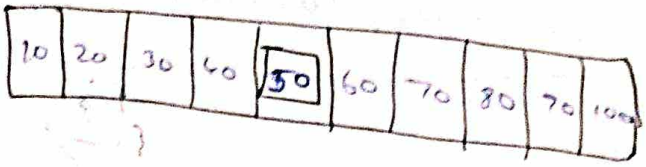
→ Divide and conquer algorithm.



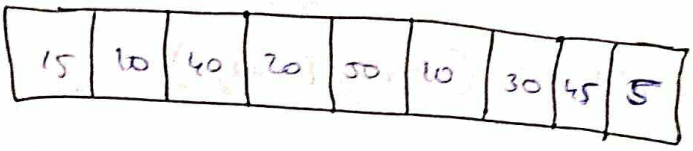


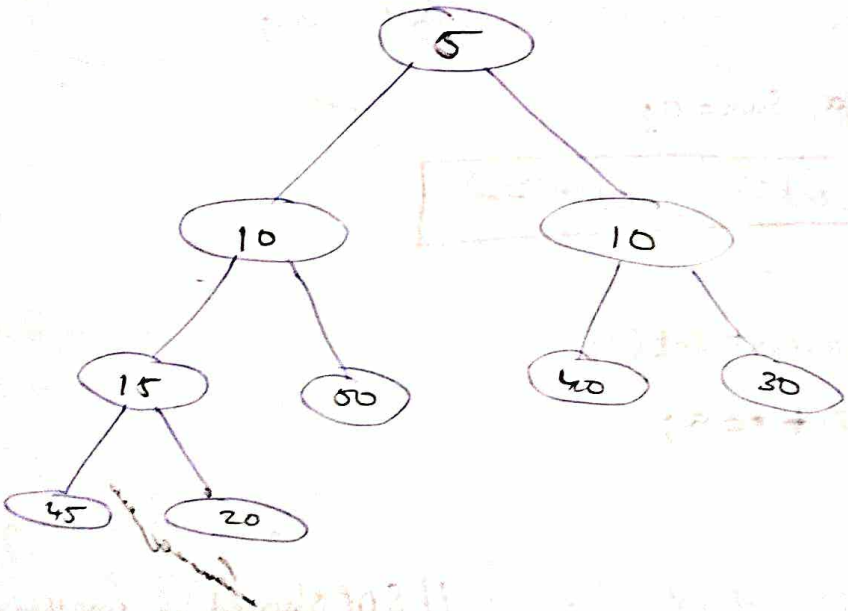
Quick Sort.





Heap Sort :-





Note:

Always System.out.println should be written outside of the (for, while, do-while) loop.

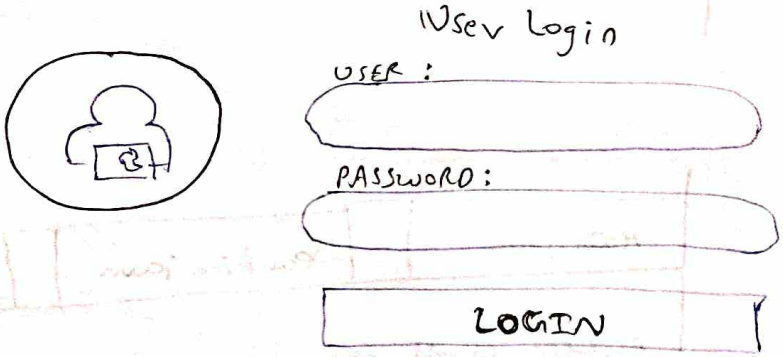
Example:-

```
import java.util.*;
class Main
{
public static void main (String [] args)
{
Scanner in = new Scanner (System.in);
int a, Sum = 0;
for (int i=0 ; i<3 ; i++)
{
a = in.nextInt();
Sum += a;
}
System.out.print (Sum); // SOP should be written outside of forloop.
}
}
```

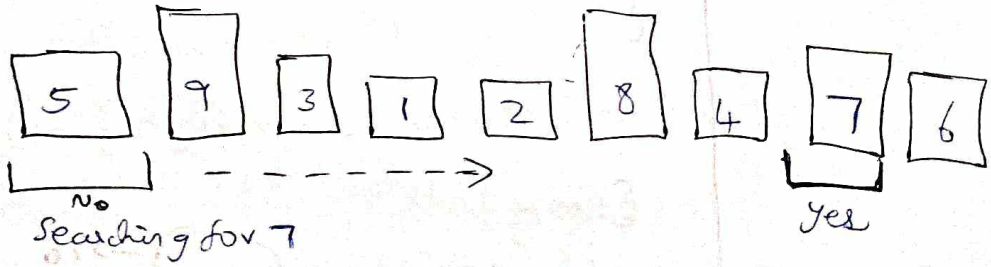
27 Searching Algorithm

1. Linear Search
2. Binary Search

Searching Algorithm



Search for 7



repl.it

v Files

Main.java

Searching.java

Main.java | X | Searching.java X

```
class Main
{
public static void main (String [] args)
{
int [] arr = {1, 3, 2, 10, 23, 13};
Searching.linearSearch (arr, 10);
}
}
```

keyword ↖ value ↖

v Files

Main.java

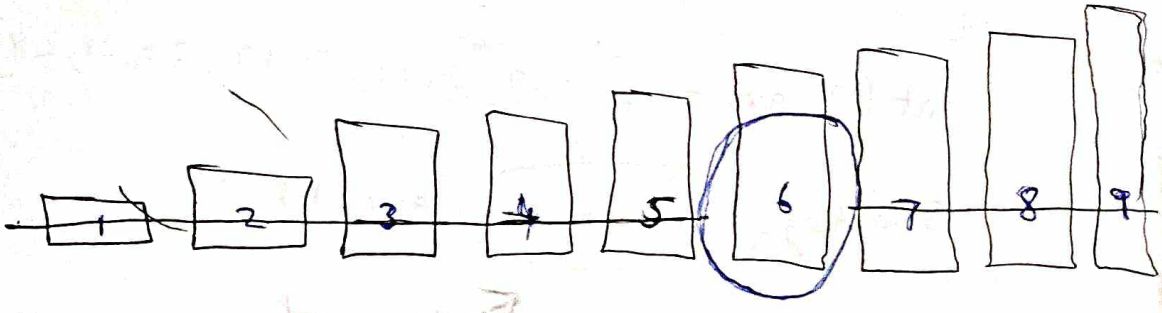
Searching.java

Main.java | Searching.java

```
public class Searching
{
public static int linearSearch (int arr[],
                                int value)
{
for (int i=0; i<arr.length; i++)
{
  if (arr[i] == value) → 10
  {
  System.out.println ("the element is found at the
                       index : "+i);
  return 1;
  }
  S.O.P ("the element "+value+" not found.");
} } return -1;
```

Binary Search

Searching for 6



Program:

Same as ~~linear search~~.

Searching.java

```
public static int binarySearch (int arr [], int value)
{
int start = 0;
int end = arr.length - 1;
int middle = (start + end) / 2;
System.out.println (start + " " + middle + " " + end);
return -1;
}
```

Main.java

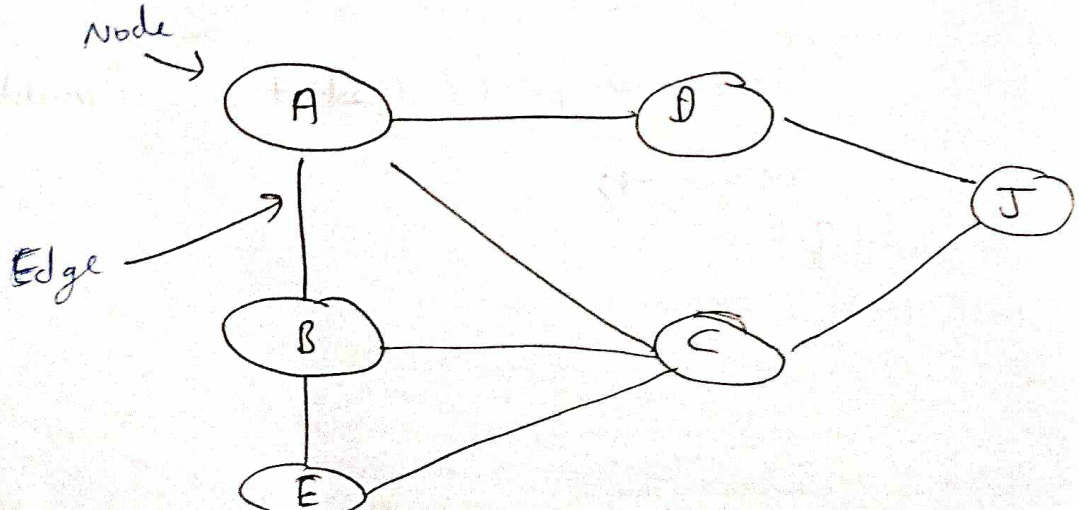
```
class Main {
public static void main (String[] args)
{
int [] arr = { 8, 9, 12, 15, 17, 19, 20, 21, 28};
Searching.(binary Search)(arr, 15);
}
}
```

← keyword

Graph

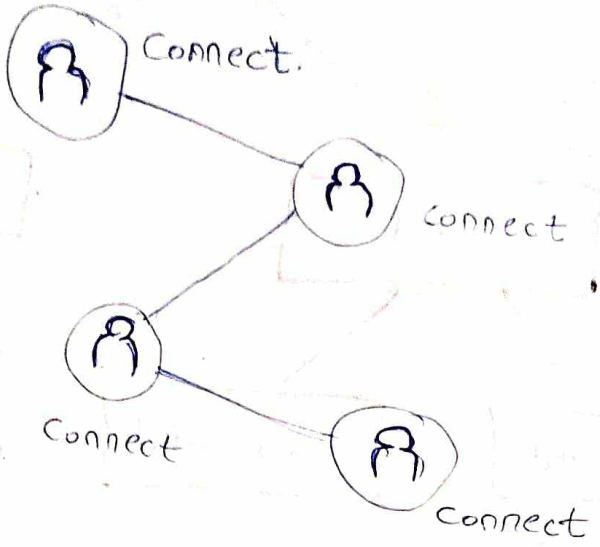
Graph → Non linear data Structure

Graph consist of finite set of vertices and a set of Edges which connect a pair of Nodes.



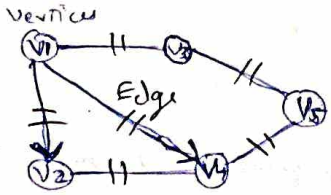
Graph :

facebook, LinkedIn



Graph Terminology.

Vertices → Vertices are nodes of graph.



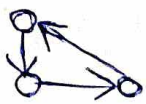
Edge

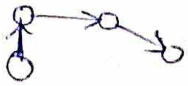
unweighted graph

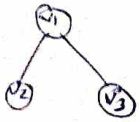
weighted graph

Undirected graph

Directed graph.

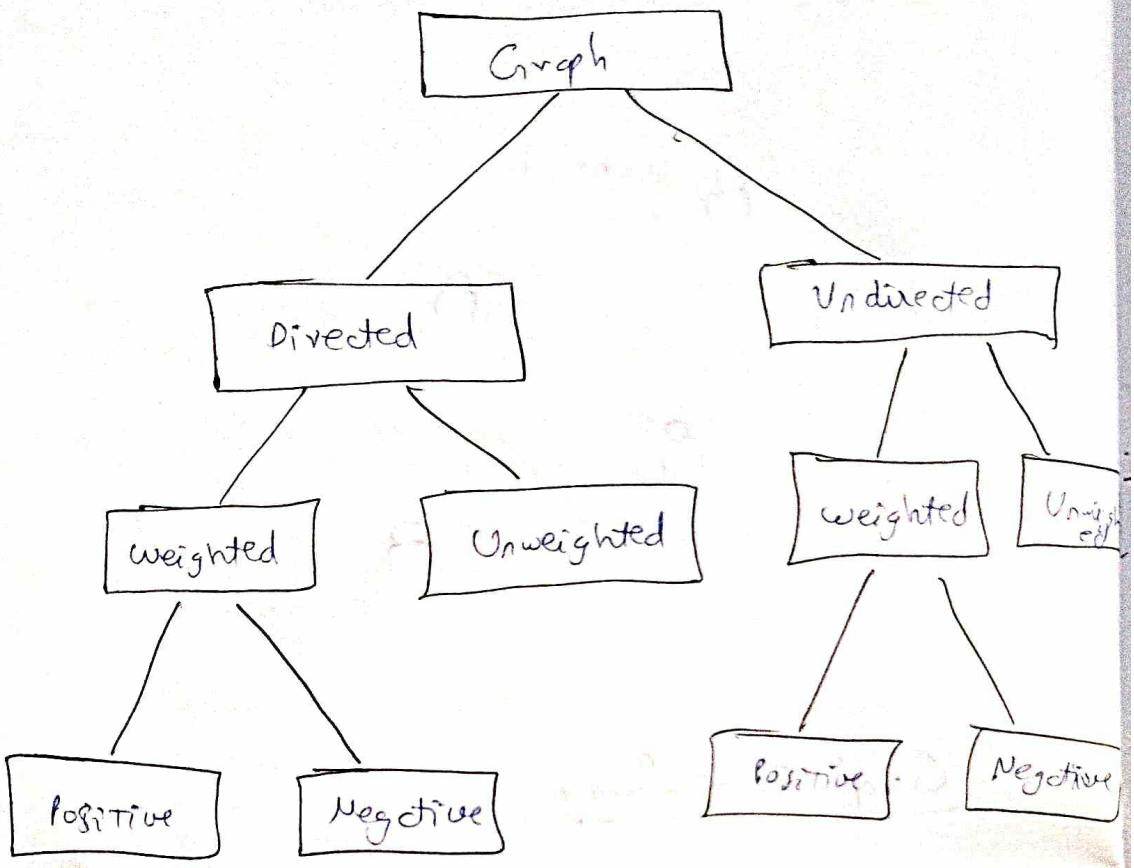
Cyclic graph 

Acyclic graph. 

Tree. 

No cycle in Tree

Graph Types :-



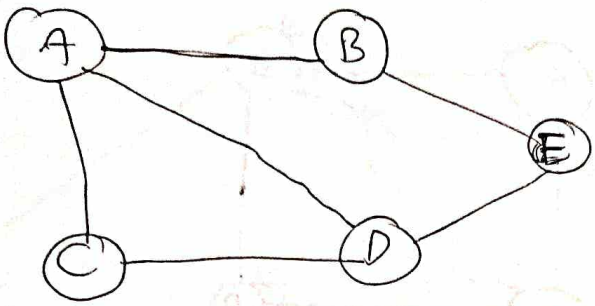
① Positive – weighted – Directed – graph

② Positive – weighted – undirected – graph

③ Negative — " — " — "

④ Negative — " — " — "

Matrix Representation



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	0	1
C	1	0	0	1	0
D	1	0	1	0	1
E	0	1	0	1	0

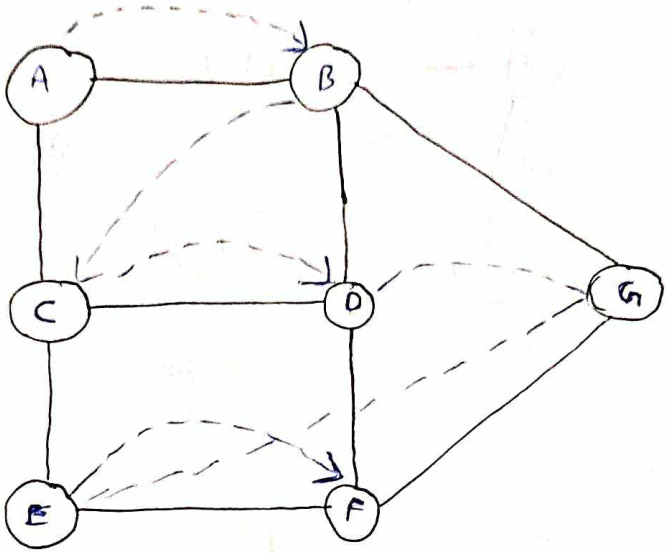
Program :-

GRAPH OUTPUT

OUTPUT: (MATRIX)

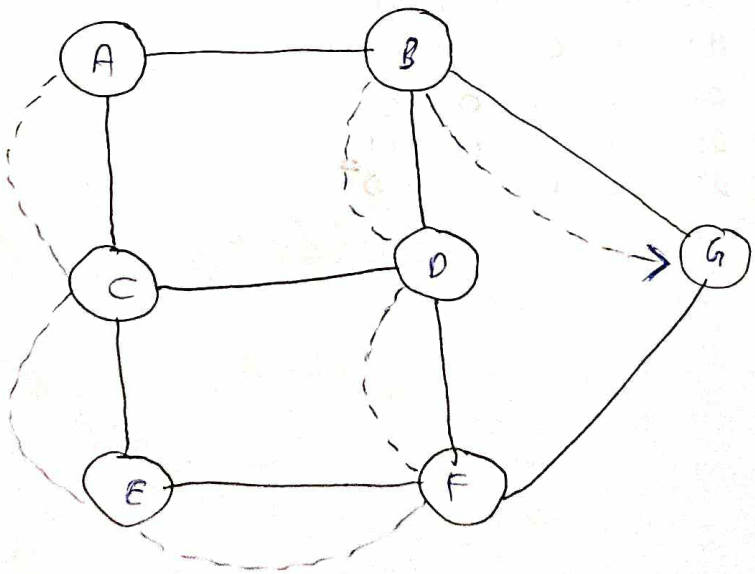
```
   A  B  C  D  E
A: 0  1  1  1  0
B: 1  0  0  0  1
C: 1  0  0  1  0
D: 1  0  1  0  1
D: 0  1  0  1  0
```

Breadth First Search Algorithm

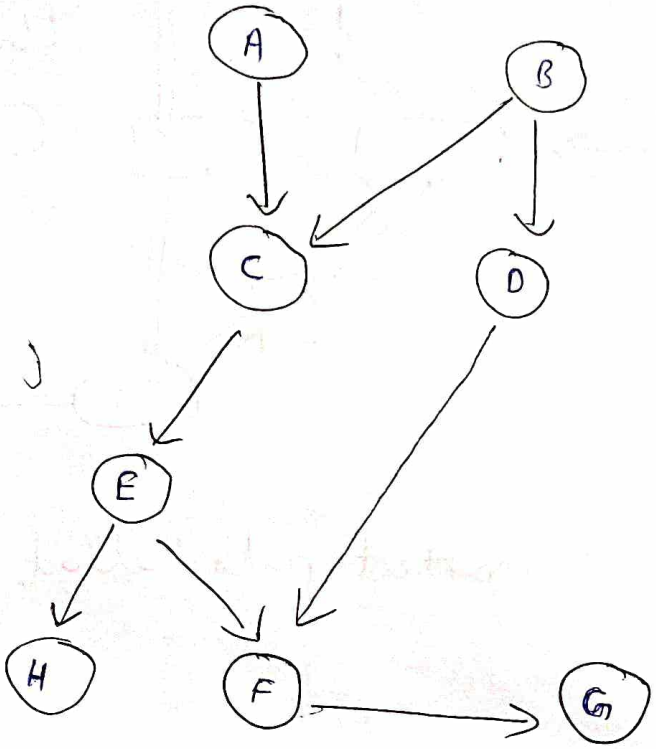


Queue

Depth first Search (DFS)

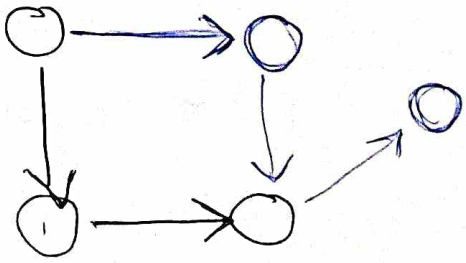


Topological Sort Algorithm



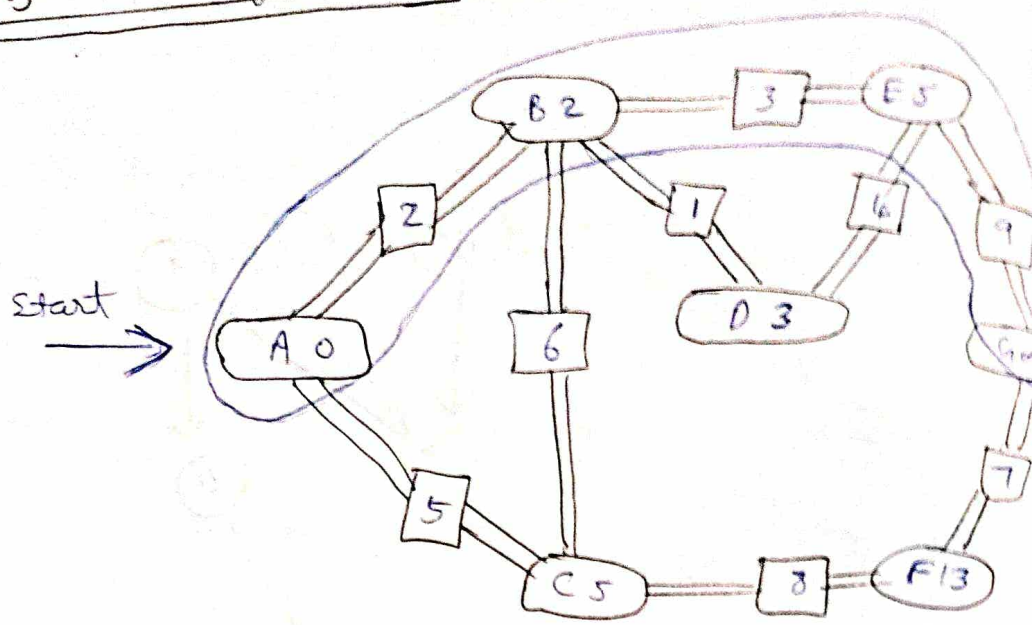
STACK → A B C D E H F G

Single Source Shortest path Problem (SSSPP)

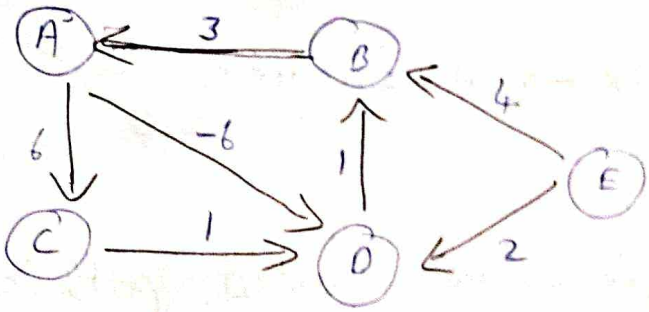


SSSPP → Finding the shortest path in over v
To minimize ₹cost and 9distance

Dijkstra's Algorithm



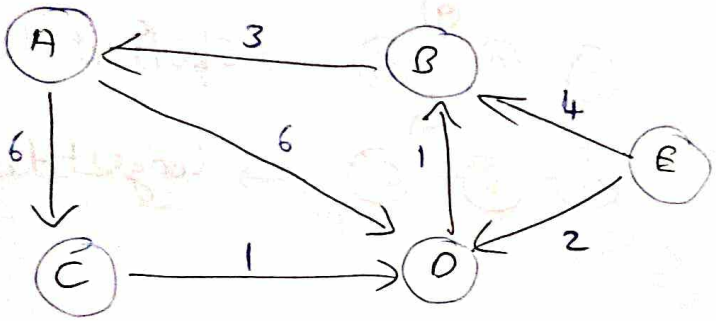
Shortest path Method



- We cannot find minimum distance in a negative cycle.
- We can only find minimum distance in a positive cycle.

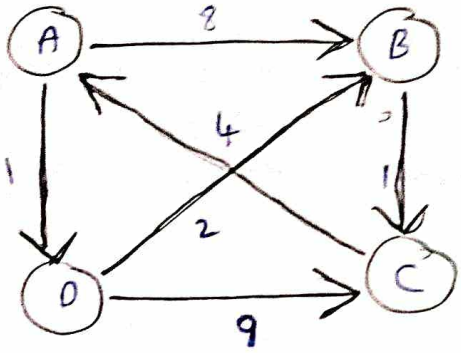
Bellman Ford Algorithm

Bellman Ford algorithm is used to find Single Source Shortest path Problem. If there is a negative cycle it catches it and report its existance.



Edge	weight
A → C	6
A → D	6
B → A	3
C → D	1
D → C	2
D → B	1
E → B	4
E → D	2

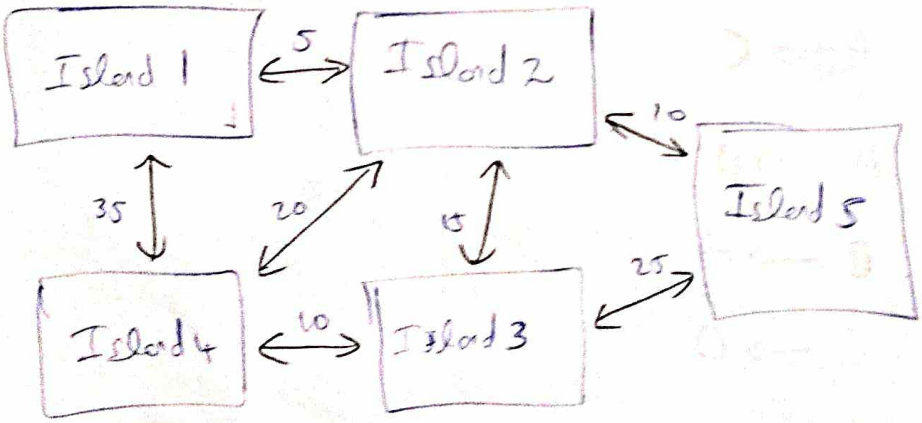
Floyd Warshall Algorithm



(9)
A – B – C → Shortest distance.

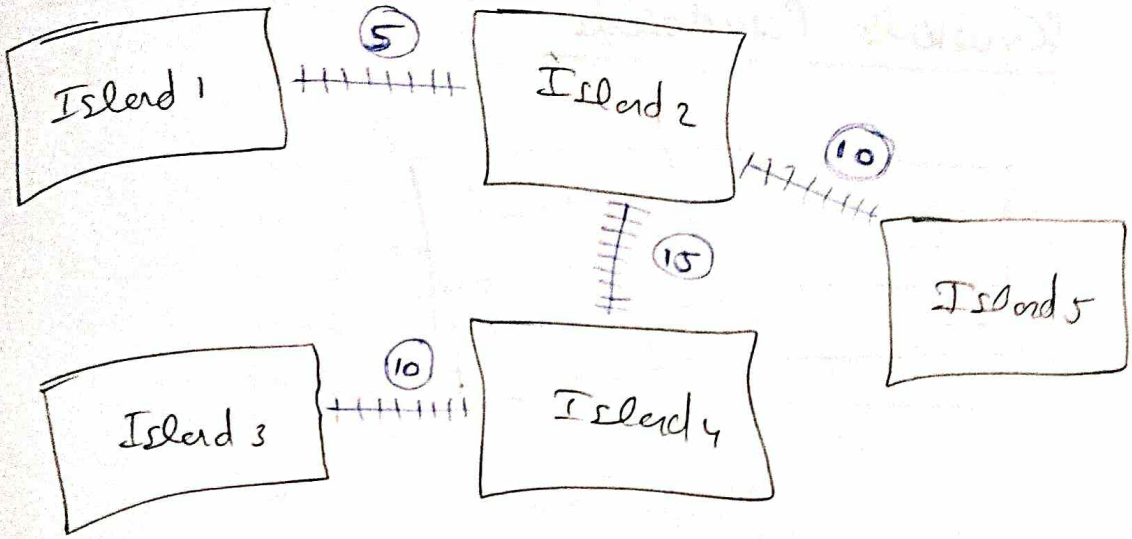
(10)
A – D – C → Longest distance.

Minimum Sparring Tree.



Problem: Correct Thake all Island is the Shortest Path So that every Island should be accesible.

Solution:

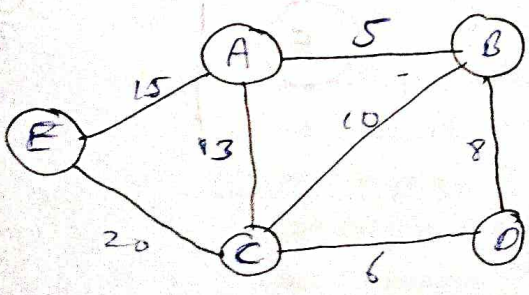


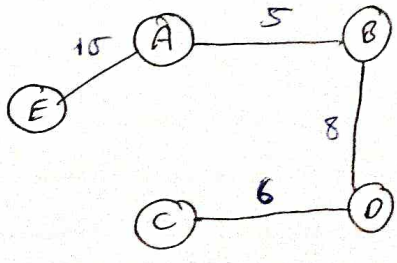
Every Island is accessible

Kruskal Algovithm

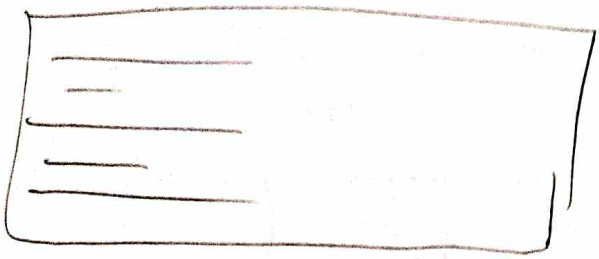
```
ArrayList <WeightedNode> nodelist = new ArrayList
                                    <WeightedNode>();

ArrayList <UndirectedEdge> edgelist = new ArrayList
                                      <UndirectedEdge>();
```

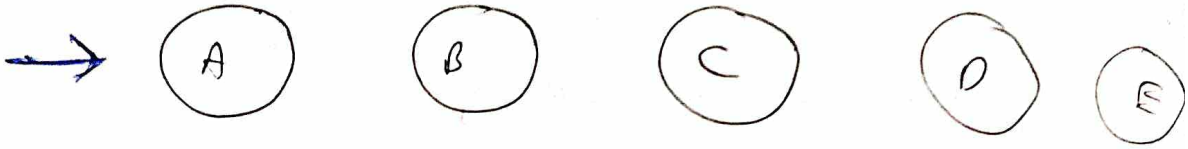
- It is a greedy Algovithm
- Assending order. Minimum cost





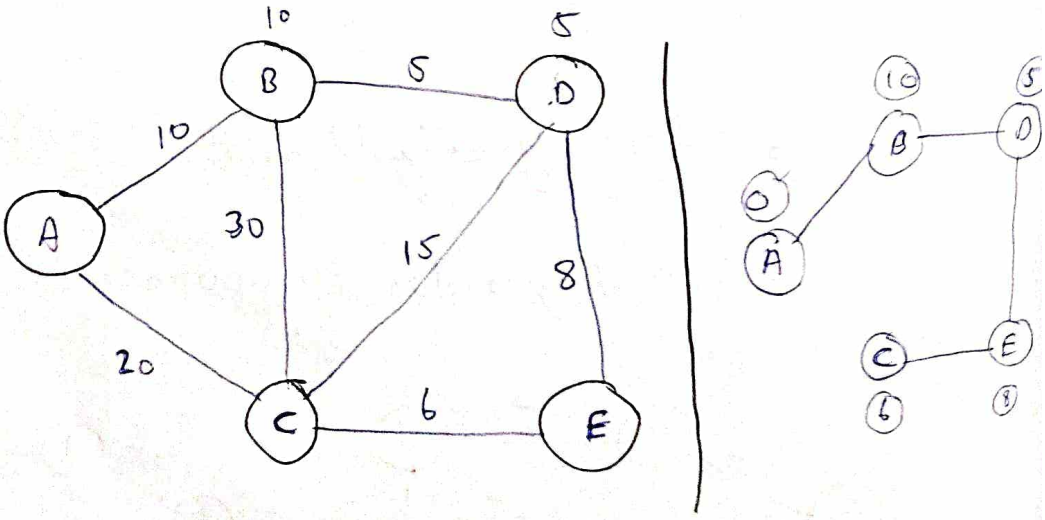
Kruskal's Pesudocode.



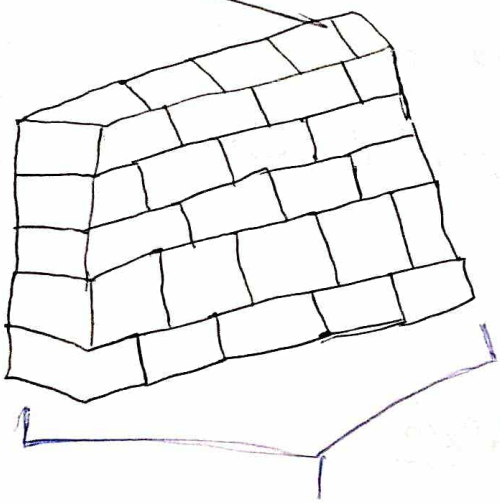


Prim's Algorithm

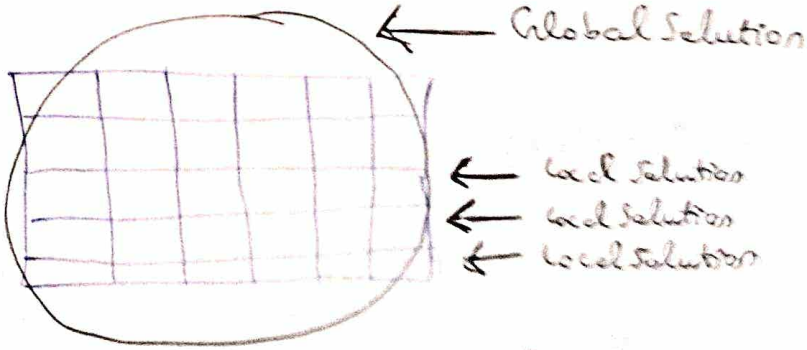
- It is a greedy Algorithm



Greedy Algorithm



- We always put the best bricks in the first layer to make it strong.
- Selecting → best bricks → To complete the wall
- It is an algorithm paradigm that builds the solution piece by piece.



- Greedy Algorithm
  - → Insertion Sort
  - → Selection Sort
  - → Topological Sort
  - → Prim's Algorithm
  - → Kruskal Algorithm

- Coin change Problem
- Fractional knapsack Problem

Coin change Problem

- You are given coins of different denomination and total amount of money. Find the minimum number of coins that you need to make up the given amount.

Infinite supply of denominations: { 1, 2, 5, 10, 20, 50, 100, 1000 }

Example 1:

Total amount: 70

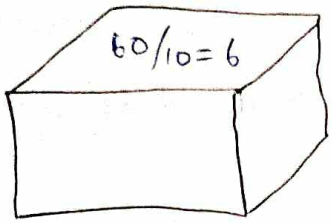
Answer: 2 → 50+20 = 70

Example 2:

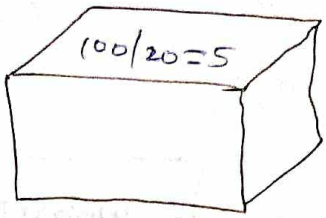
Total amount: 122

Answer: 3 → 100+20+2 = 122

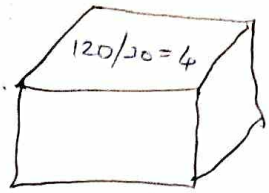
Fractional Knapsack Problem



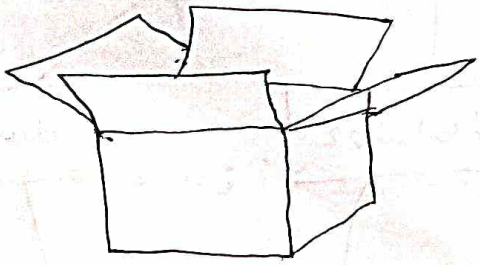
10kg Value: 60



20kg, Value: 100



30kg, Value: 120

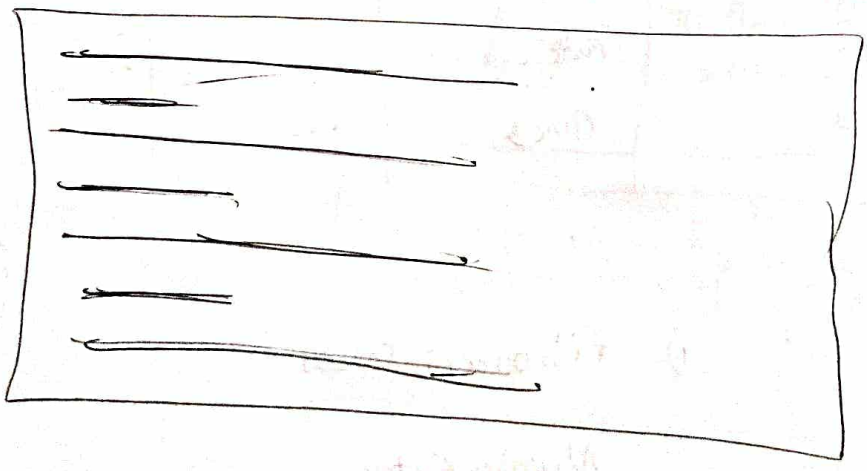


50kg

100 + 120 = 220

60 + 100 + 120 * 2/3 = 240

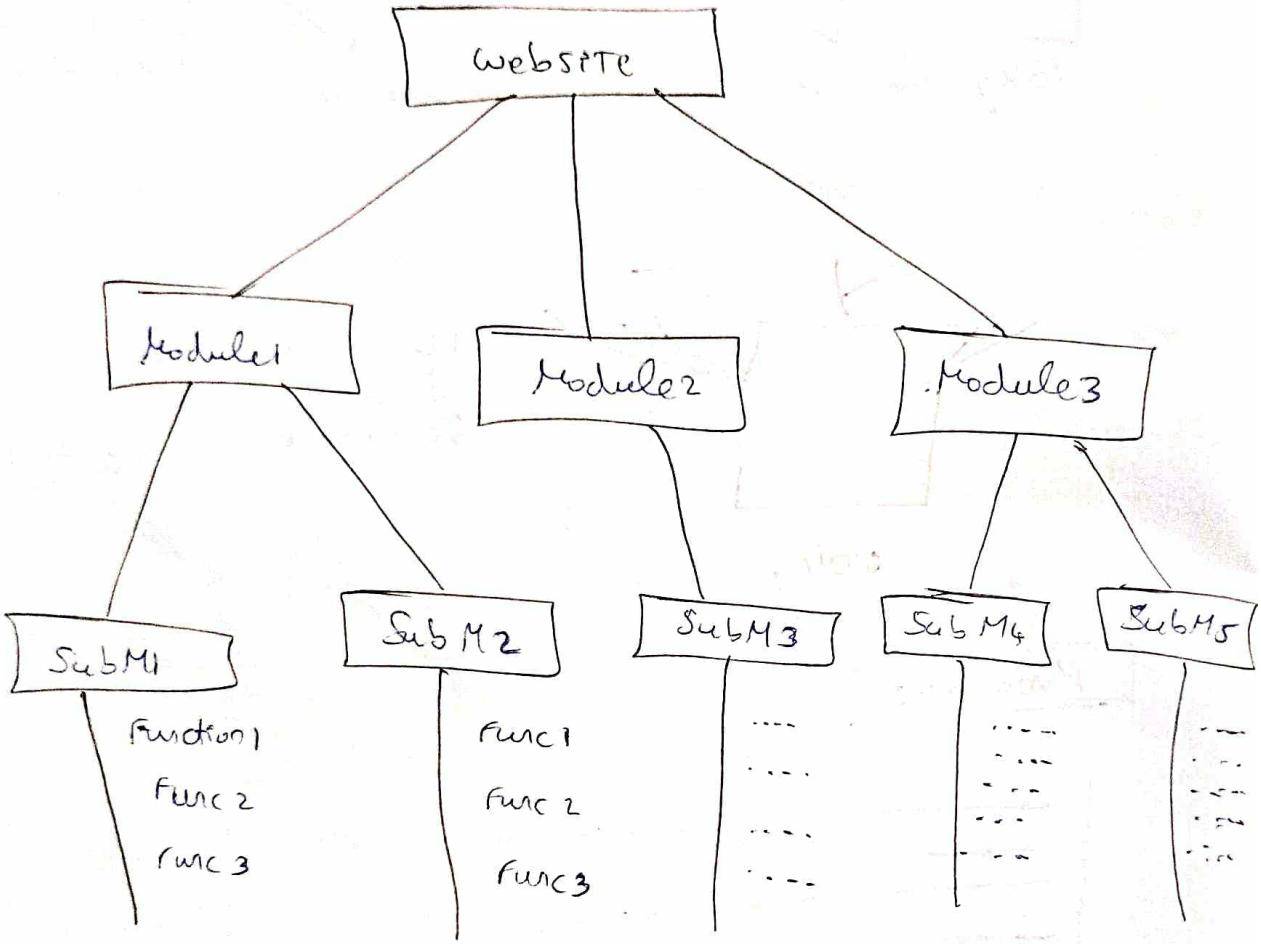
Program:



Potter Program → Interconnected Program.

Divide and Conquer Algorithm:

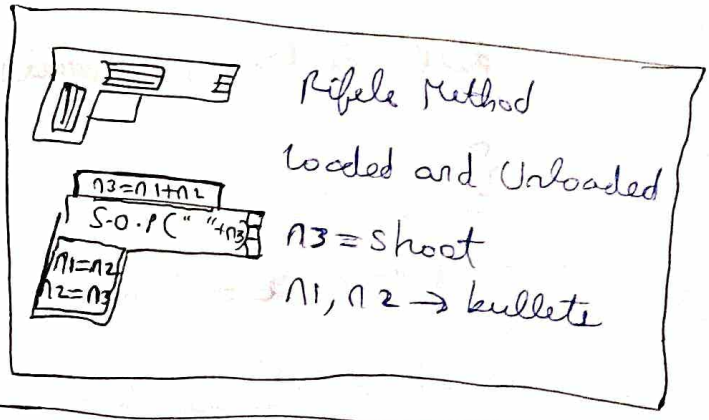
- Dividing the Problem into Sub Problems.

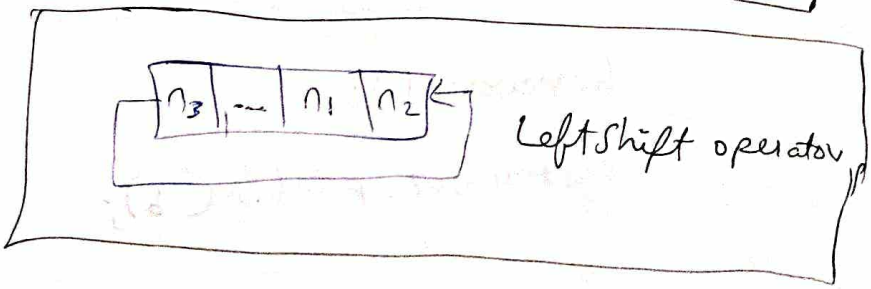


1) fibonacci Series
2) Number Factor
3) One String To another
4) Knapsack Problem
5) Minimum cost

Fibonacci series

```
class Fibonacci
{
public static void main (String [] args)
{
int n1=0, n2=1, n3, i, count=10;
System.out.println(n1+" "+n2);
{
for (i=0; i<count; i++)
{
n3=n1+n2;
S.o.p(" "+n3);
n1=n2;
n2=n3;
}
}
}
```





- each number is the sum of two preceding numbers
- Starts with 0 and 1.

String:

Reverse a String.

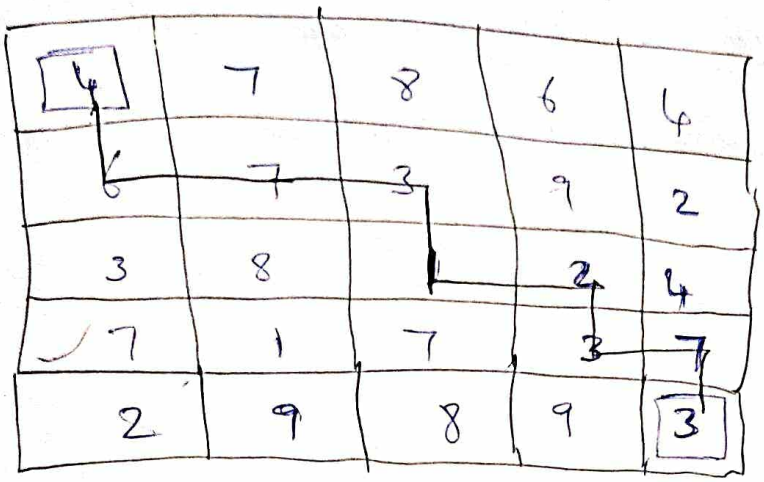
① StringBuffer b = new StringBuffer(a);

② StringBuilder b = new StringBuilder(a);

Program:

```
class test
{
public static void main (String[] args)
{
 String a = "Hellow";
  StringBuffer b = new StringBuffer(a);
  b.reverse();
  System.out.println(b);
  }
}
```

Minimum cost to reach the last cell.



2D MATRIX

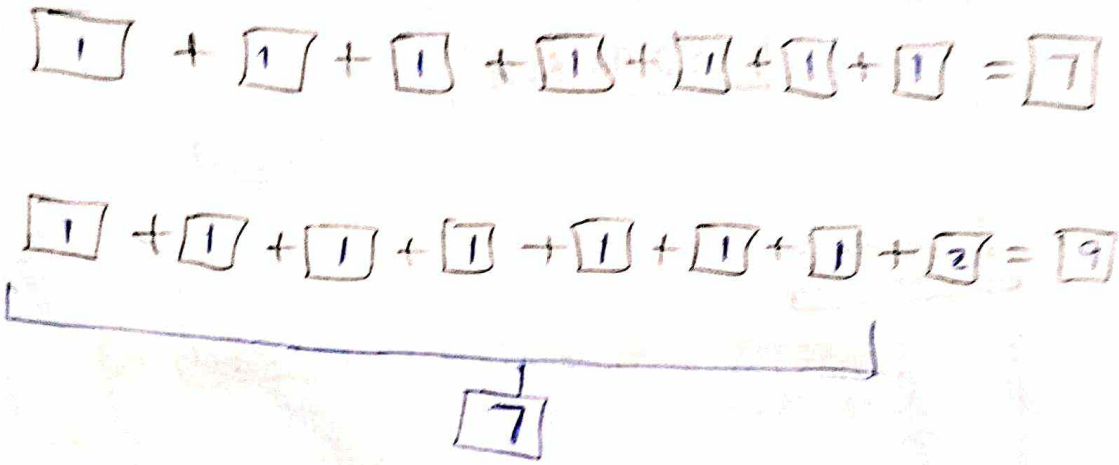
Program :

```
class Main
{
PSVM (String[] args)
{ int[][] array =
{ 1,2,3,4,5 };
{ 6,7,8,9,10 };
{ 11,12,13,14,15 };
{ 16,17,18,19,20 };
}}
```

Dynamic programming

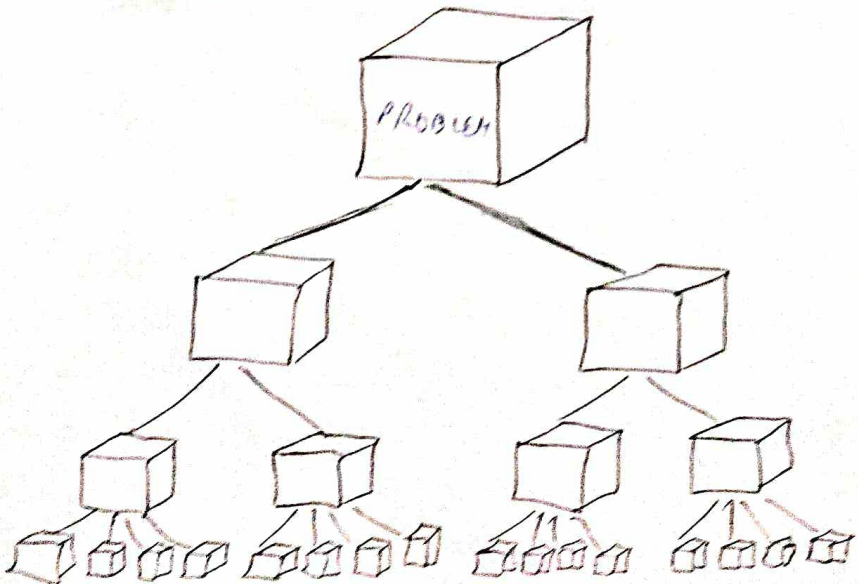
Dynamic programming is an algorithm technique by solving an optimization problem by breaking it down into simpler subproblems.

Example:1

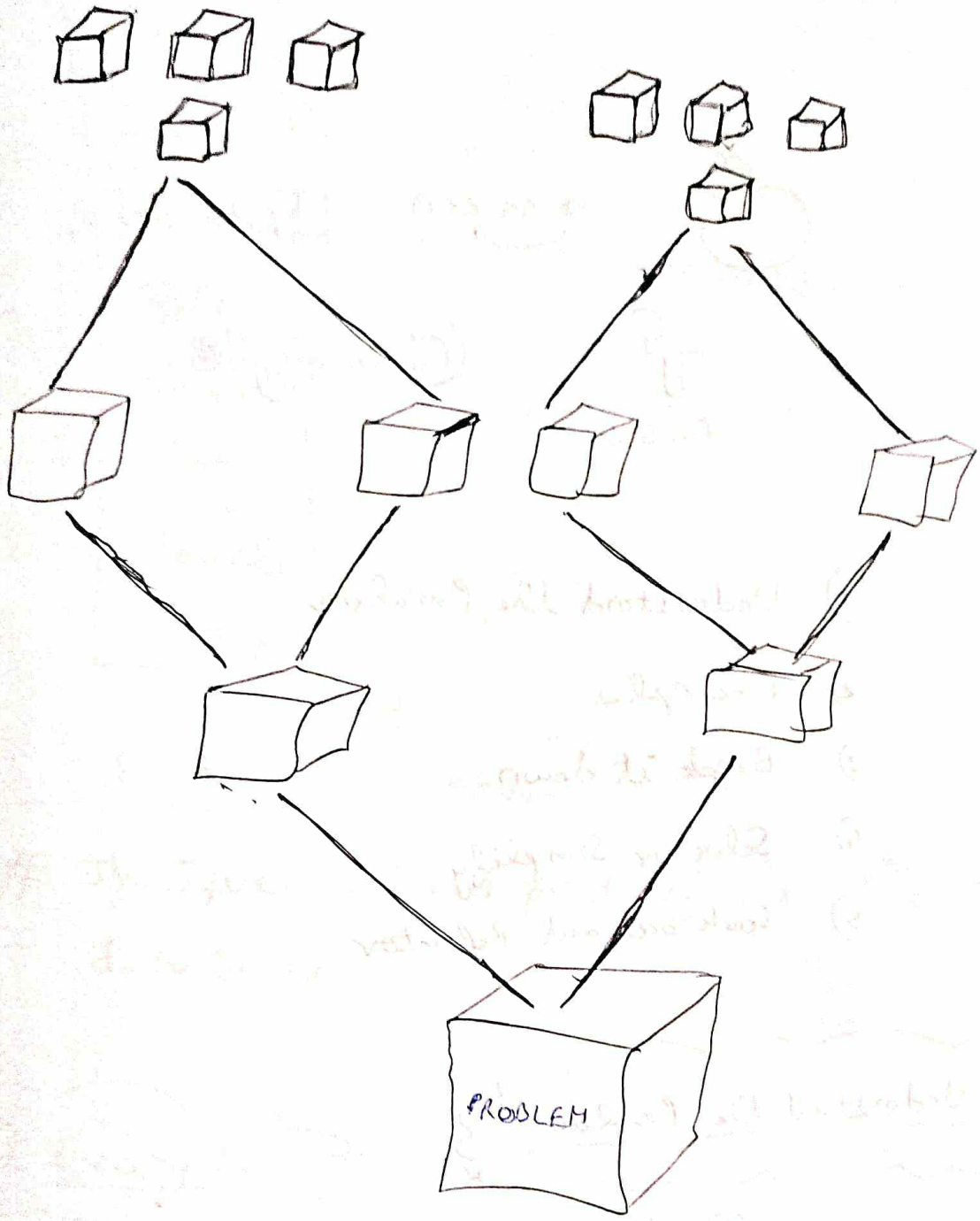


Example: Fibonacci series

TOP-DOWN ALGORITHM



BOTTOM UP APPROACH



TOP - DOWN APPROACH : BIG - SMALL

BOTTOM-UP APPROACH : SMALL - BIG

Recipe for Problem Solving

amazon

Microsoft

facebook

Google

1) Understand the Problem
2) Examples
3) Break it down
4) Solve or Simplify
5) Look back and Refractor.

1) Understand the Problem:

* Many People Learn Coding but they don't know how to Solve a problem.

* Analyze How many Inputs.

* Analyze How many Outputs.

Data Types

int
float
char
byte
short
long
double
String

```
class Main
{
 public static void main (String[]args)
{
 int a;
 S.O.P (a);
  }
}
```

OUTPUT:
#error

The System does not print default value for data Types

② Example

write a function which takes in a string and returns count of each character in the string.

Own program:

```
class Program
{
public static void main(String[] args)
{
int count=0;
String s="Hellow";
for(int i=0; i< s.length(); i++)
{
System.out.println(s.charAt(i));
count++;
}
S.O.P("The number of characters are:"+count);
}
}
```

Confusion Program

```
class test
{
PSVM(String[]args)
{
S.O.P(a); <-- (*) line
String a="Hellow";
}
}
```

OUTPUT:
#error

(3) BREAK IT DOWN.

(*) write out the step that you need to take

④ SOLVE / SIMPLIFY

Simplify the Problem

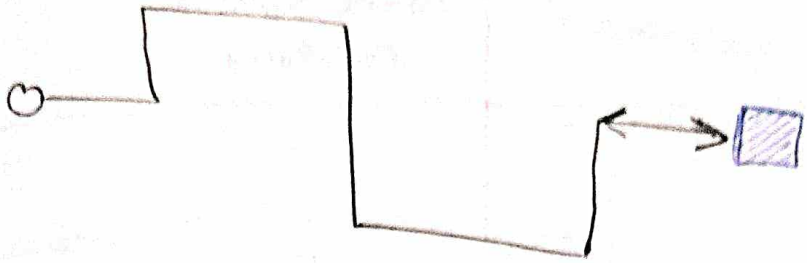
1) Find the core difficulty
2) Temporarily ignore the difficulty
3) Write a simplified solution
4) Then incorporate that difficulty.

⑤ LOOK BACK REFRACTOR

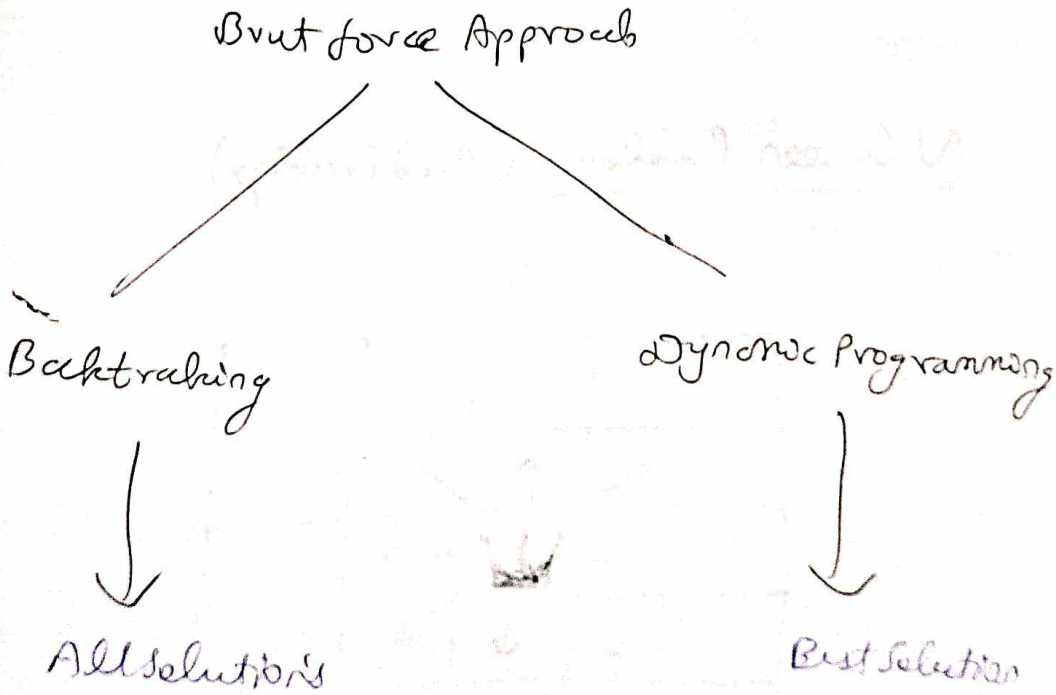
(*) Improving performance of Solution

PROGRAM:

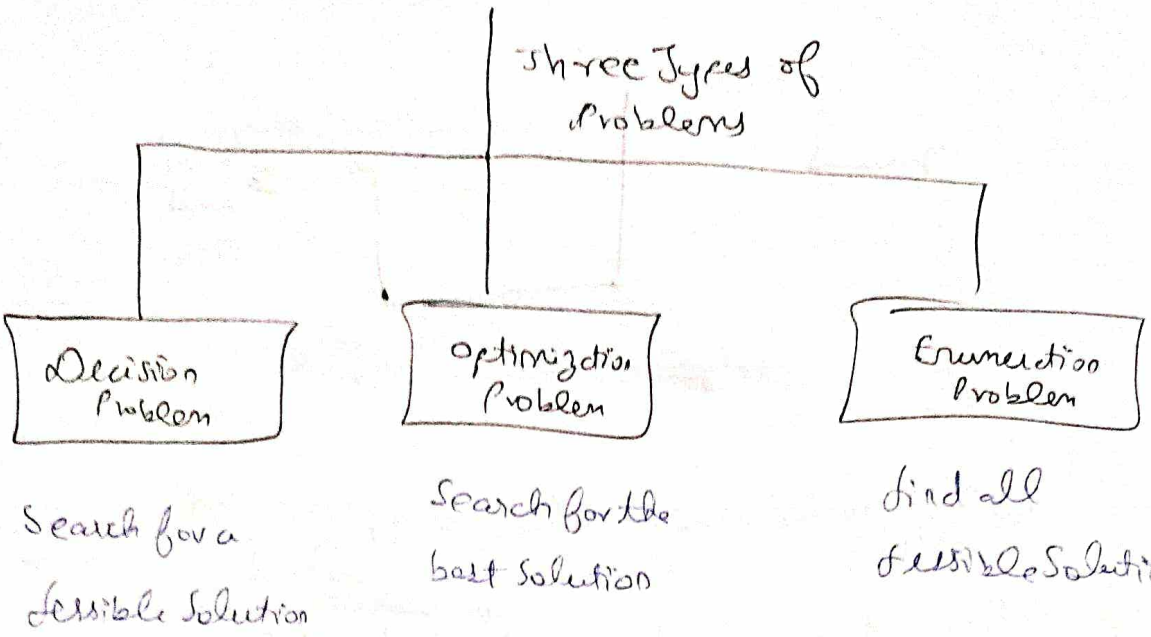
BACKTRACKING



- A backtracking algorithm is a problem solving algorithm that uses a brut force approach for finding the desired output.

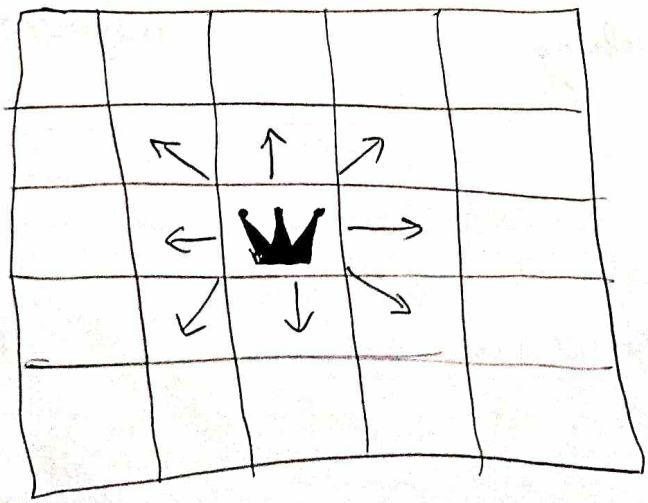


BACKTRACKING



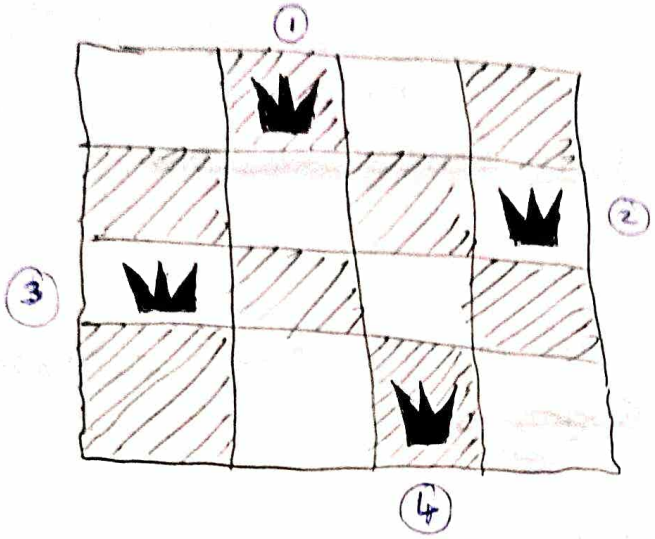
BackTracking → checks → Node → valid or Not
✓ ✗

N Queen Problem (BackTracking)



Queen

Place 4 queens on 4×4 chess board, in such a manner that no two queens can attack each other.



Cereal Programing

① Cereal getter and Setter program.

```
1) public class Bank
{
        int minimumamount = 2500;
private [redacted];

public static int getMinimumamount ()
{
return minimumamount;
}

public void setMinimumamount (int value)
{
  this.minimumamount = value;
}

public static void main (String [] args)
{ Bank Lakshman = new Bank();
  int min = Lakshman.getMinimumamount();
  System.out.println(min);
  min = Lakshman.setMinimumamount (3000);
  System.out.println (min);
}
```

② Compare Two arrays (easy)

> **Keyword:**
> Arrays.equals(a,b);
> compare two arrays

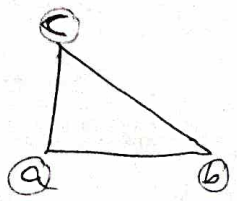
```
import java.util.*;
class Game
{
public static void main(String[] args)
{
int[] a={1,3,3,4,5};
int[] b={1,2,3,4,5};
System.out.println(Arrays.equals(a,b));
}
}
```

---

③ Compare Three arrays

```
import java.util.*;
class Compare {
Public static void main(String[] args){
int[]a={1,3,3,4,5};
int[]b={1,2,3,4,5};
int[]c={1,2,3,4,5};
boolean compare1 = Arrays.equals(a,b); //a=b
boolean compare2 = Arrays.equals(b,c); //b=c
boolean compare3 = Arrays.equals(c,a); //c=a
if (compare1 && compare2 && compare3)
S.O.P ("All arrays are equal");
else
S.O.P ("All arrays are not equal");
}
}
```

Compare Three Arrays = Pythogorous Theorem.



a=b
b=c
c=a

Find LowerCase letters, UpperCase letters and digits in a String

```
class Find
{
public static void main (String [] args)
{
String s = "HelloW..";
int count = 0;
int Ucount = 0;
int Dcount = 0;
char [] c = s.toCharArray ();
for (int i = 0; i < s.length (); i++)
{
System.out.print (s.charAt (i));
if (Character.isLowerCase (c[i]));
{
Count++;
}
else if (Character.isUpperCase (c[i]));
{
Ucount++;
}
else if (Character.isDigit (c[i])
{
Dcount++;
}
else
{
System.out.println ("There is a wrong Input");
}
} Main Close
S.o.P ("The LowerCase is:" + count);
S.o.P ("The UpperCase is:" + Ucount);
S.o.P ("The Digit is:" + Dcount);
}
}
```

Keyword

toCharArray ();
Convert String to character
Character.isLowerCase (c[i]);
Character.isUpperCase (c[i]);
To check the character

else if
;
Space

Difference

① Character.isLetter(c[i]);

finding the number of letters in a String.

Eg: Hellow / The number of letters is: 6

② Character.isLowerCase(c[i])

finding the number of lowercase letters in a String

Eg: Hellow / The number of LC letters are: 5

③ Character.isUpperCase(c[i])

finding the number of Upper Case letters in a String

Eg: Hellow / The number of Upper Case letters are: 1

④ Character.isDigit(c[i])

finding the number of DIGITS in a String.

Eg: Hellow 12345 6 / The number of digits are: 6

Find the number of even digits in a number, if it is greater the 2 then print true else false

```
import java.util.*;
class EvenDigit
{
public static void main (String[]args)
{
 Scanner in = new Scanner (System.in);
 int number = in.nextInt();
 int evencount=0;
 while (number>0)
 {
 int rem = number%10;
 if (rem%2==0)
 {
  evencount++;
 }
 number = number/10;
}
if (evencount>2)
{
  S.o.p ("true");
}
else
{ S.o.p("false");
```

Input: 14236 OUTPUT: TRUE INPUT: 1459 OUTPUT: False	%10 to separate group of integer value to single value

Find the number of even digits in a number, if it is greater the 2 then print true else false

```
import java.util.*;
class EvenDigit
{
 public static void main (String[] args)
 {
  Scanner in = new Scanner (System.in);
  int number = in.nextInt();
  int evencount=0;
  while (number>0)
  {
   int ven = number%10;
   if (ven%2==0)
   {
    evencount++;
   }
   number = number/10;
  }
  if (evencount>2)
  {
   S.o.p (+true);
  }
  else
  {
   S.o.p (+false);
```

Input:
14236

OUTPUT:
TRUE

INPUT:
1459

OUTPUT:
False

%=10 to separate digits of the value to single value

Find the length of the String with Space.

```
class Test
{
public static void main (String[] args)
{
String s = "Hellow world";
System.out.println(" The number of words in a given
    String with Space : " + s.length());
}
}
```

Keyword

s.length();

Finding words with Space

Reverse the words in a String.

Input:

Welcom to my Java Progvraning.

Output:-

Progvraning Java my to Welcome.

Program Turn over →

Program

```
import java.util.*;
class Word {
public static void main (String [] args)
{
Scanner sc = new Scanner (System.in);
String str = sc.nextLine ();
String [] arr = str.split (regex: " ");
for (int i = arr.length -1; i >=0; i--)
{
 if (arr[i] != " ")
 {
  System.out.println (arr [i] + " ");
 }
}
}
}
```

Input:

Welcome to my Program.

Output:

Program to my Welcome

Remove Space in a String

```
class RemoveSpace
{
public static void main (String [] args)
{
 String str = " Hellow World";
 System.out.println (str.replace (" ", ""));
 }
}
```

remove

add

Input:
Hellow World
Output:
HellowWorld

```
import java.util.*;
class ReplaceString
{
 public static void main (String args)
{ Scanner in = new Scanner (System.in);
 String str = " Hellow world";
 String replace = in.nextLine();
 System.out.println (replace, " ");
 }
}
```

Reverse a string using for loop.

```
import java.util.*;

public class test {

public static void main (String[] args)
{

Scanner sc = new Scanner (System.in);
String str = sc.nextLine();

for (int i = str.length(); i >= 0; i--)
{
    System.out.println (str.charAt(i) + "");
} // Printing from Back Side
}
}
```

Reverse a string using StringBuffer and StringBuilder

```
import java.util.*;
class Test {
public static void main (String[] args)
{
 String a = "Hellow";
 StringBuffer b = new StringBuffer (a);
```

```
b.reverse();
System.out.println(b);
}
```

③

Input:
Hellow
Output:
wolleH

Input:
welcome to my Program
Output:
margorP ym ot emoclew

Reverse the Integer Number

```
class IntegerReverseNumber
{
public static void main(String[] args)
{
int number = 123456;
int reverse = 0;
int remainder;
while (number > 0)
{
remainder = number % 10;
reverse = reverse * 10 + remainder;
number = number / 10;
}
}
System.out.print(reverse);
```

How it works →

working:

int number = 123456;

Same program

① Step

while (123456 > 0) ✓

{

⑥ ← 123456

remainder = number % 10;

0 ← ⑥

reverse = reverse * 10 + remainder;

6 ← 0 + 6

12345 ← 123456

number = number / 10;

}

Same program

② Step :

while (12345 > 0)

{

⑤ ← 12345

remainder = number % 10;

⑥ ← ⑤

reverse = reverse * 10 + remainder;

65 ← 60 + 5

1234 ← 12345

number = number / 10;

}

Same program

③ like wise it repeats 654321.

xxxxxxx

Palindrome Program (easy)

```
class Palindrome {

public static void main (String [] args) {

int number = 12345;
int reverse = 0;
int reminder;
int temp;

temp = number;

while (number > 0)
{
reminder = number % 10;
reverse = reverse * 10 + reminder;
number = number / 10;
}
if (temp == reverse)
{
System.out.println (" Palindrome number");
}
else
{
System.out.println ("Not Palindrome");
}
}
```

why we use temp

Number value → changes

temp value → Not changes

Printing patterns in java.

① class RightTrianglePattern

```
{
public static void main (String [] args)
{
int i,j;
for (int i=0; i<6; i++)
{
for (j=0; j<i; j++)
{
System.out.print("*");
}
System.out.println();
}
}
}
```

Output:

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

Printing patterns in java.

(1) class RightTrianglePattern

```
{
public static void main (String[] args)
{
int i,j;
for(int i=0; i<6; i++)
{
for(j=0;j<i;j++)
{
System.out.print("*");
}
System.out.println();
}
}
}
```

Output:

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

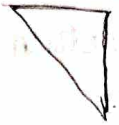
website: javapoint.com

② Left Triangle Pattern 

③ Pyramid Pattern 

④ Diamond Pattern 

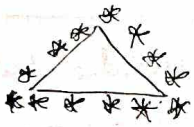
⑤ Downward Triangle Pattern 

⑥ Right Down Mirror Pattern 

⑦ Right Pascal Triangle Pattern 

⑧ Left Pascal Triangle Pattern 

⑨ Sand glass Pattern 

⑩ Triangle Pattern 

⑪ Number Pattern

Pattern depends only on For-loop

Best example for static method using return.

```
class Testing
{
 public static int function (int i, int j)
 {
  return i+j;   // Return does not print anything
 }
 public static void main( String [] args)
 {
   int a = function (10, 20);
   System.out.println(a);
  }
}
```

Static function can be called inside static method directly.

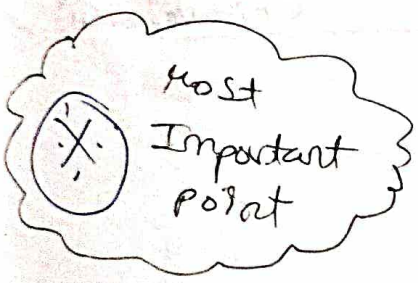
Best example for Non static method using return

```
class Testing
{
 public int function (int i, int j)
 {
  return i+j;   // Return does not print anything
 }
```

```
public static void main(String [] args)
{
    Testing Test = new Testing();
    int a = Test.function(10,20);
    System.out.println(a);
}
}
```

In order to call a Non static function we have to create an object for Class and Then we have to call inside main method.

```
public static int function(int i, int j)
```



If we give any arguments we should declare value only in arguments.

```
int a = function(10,20);
```

→ ✓

```
int
int i = 10;
int j = 20;
```

→ ✗

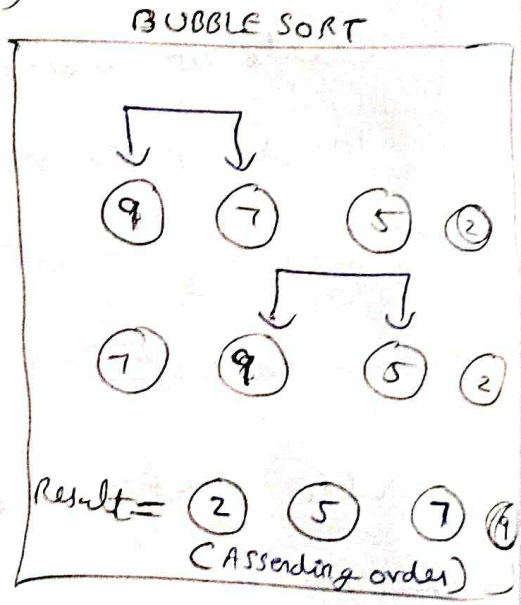
```
i = 10;
j = 20;
```

→ ✗

we should not declare outside.

Bubble Sort one simple Program.

```
int[] arr = {1, 3, 4, 5, 2};
for (int i=0 ; i < arr.length; i++)
{
  for( int j=i+1 ; j<arr.length;j++)
      {
       if (arr[i] > arr(j))
       {
         tem = arr[i];
         arr[i]= arr[j];
         arr[j] = temp;
       }
}
}
```

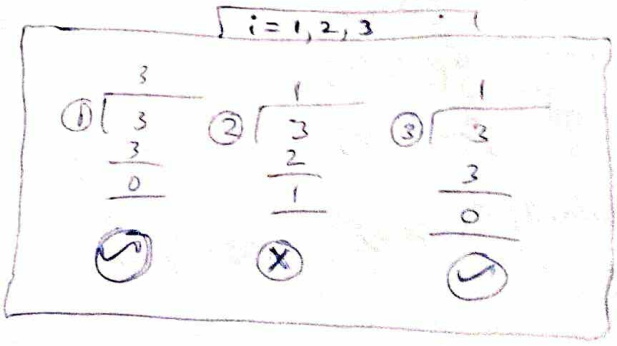


PRIME NUMBER

19 → 1 and 19 Itself (Should have only 2 factors)

28 → 1, 2, 4, 7, 14, 28 → Not a Prime Number.

```
import java.util.*;
public class PrimeNumber {

public static void main (String[] args)

{
Scanner in = new Scanner (System.in);
int num = in.nextInt();
int count = 0;

if (num > 1)    // If this condition is not true straight away goes to else part.
{

for( int i = 1; i <= num ; i++)
{
 if( num % i == 0)
  count++;
}

if (count == 2)
{
  System.out.println (" Prime Number");
}
else
{
System.out.println ("Not a Prime Number");
}
}
else
{
System.out.println ("Not a Prime number");
}
}
```



Armstrong Number

153 → 1 5 3

= 1*1*1 + 5*5*5 + 3*3*3

= 1 + 125 + 27

= 153

LHS = RHS

```
public class Armstrong
{
Public static void main (String [] args)
{
  int n = 153;
  int temp = n;
  int r, sum = 0;

  while (n > 0)
  {
   r = n % 10; // Seperated 3
   n = n / 10; // Seperated 15
   sum = sum + r * r * r;
  }

  if ( temp == sum)
  System.out.println(" It is a Armstrong Number");
  else

  System.out.println ("It is not a Armstrong Number");
} }
```

OOPS Concept

1) Class & object
2) Encapsulation
3) Abstract (No object created)
4) Polymorphism
5) Inheritance

Example

① Class and object.

```
class Pen {

    String color;
    String type;

②   public void write() {

    System.out.println(

public class OOPS {

public static void main (String args[])
{

Pen pen1 = new Pen();
pen1.color = "blue";
pen.color = "gel";
}
}
```

```
class Pen
{
    public void printColor {
        System.out.println(this.color);
    }
}

public class OOPS {
    public static void main(String []args)
    {
        Pen pen1 = new Pen();
        pen1.color = "blue";
        pen1.type = "gel";
        pen1.printColor();
    }
}
```

This keyword.

Non static method → object should be created.

Static Method → Do it want to create object.

Constructor

- Non parameterized constructor
- Parameterized constructor
- Copy Constructor

① Non Parameterized Constructor

```
Student() {
    System.out.
```

② Parameterized Constructor

```
Student (String name, int age) {
    this.name = name;
    this.age = age;
}
```

③ Copy constructor.

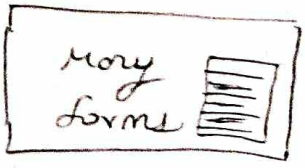
```
Student (Student s2) {
    this.name = s2.name;
    this.age = s2.age;
}
```

C++ → constructor & Destructor

Java → Constructor & Garbage Collection.

Polymorphism (Same form → Different Candidate)

```
Public void printInfo (String name) {
    System.out.println(name);
}

Public void printInfo (int age) {
    System.out.println(age);
}

Public void printInfo (String name, int age) {
    System.out.println(name + " " + age);
}
```



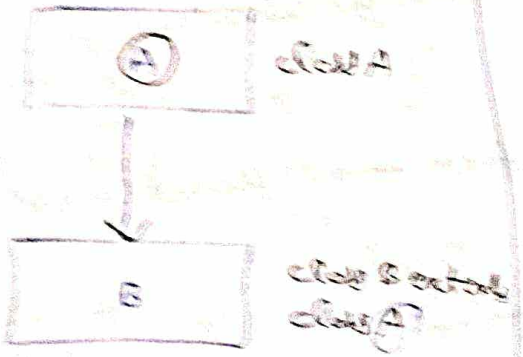
Polymorphism → Compile time polymorphism
→ Run time polymorphism

Inheritance

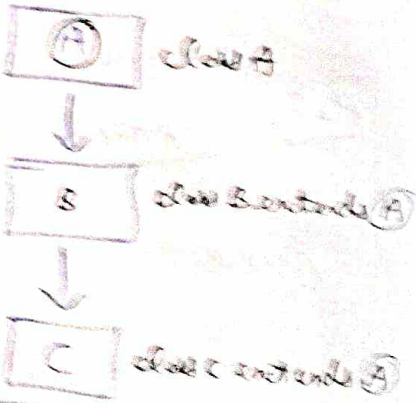
1. Single level Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance
5. Multiple Inheritance

Single Concept

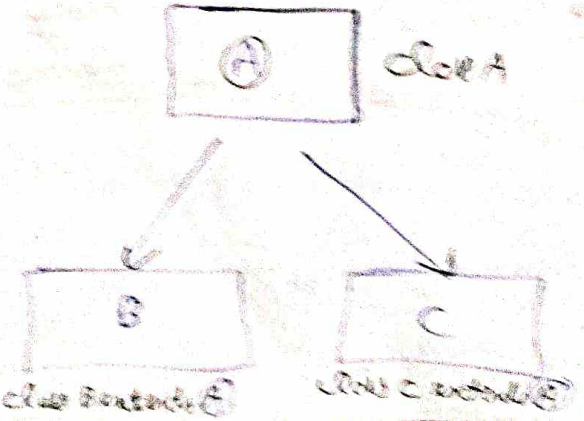
Single Inheritance



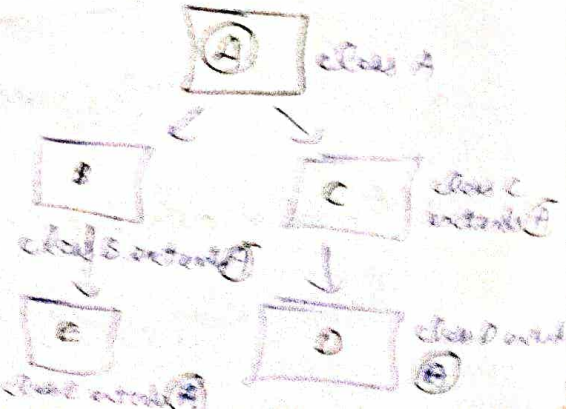
Multilevel Inheritance



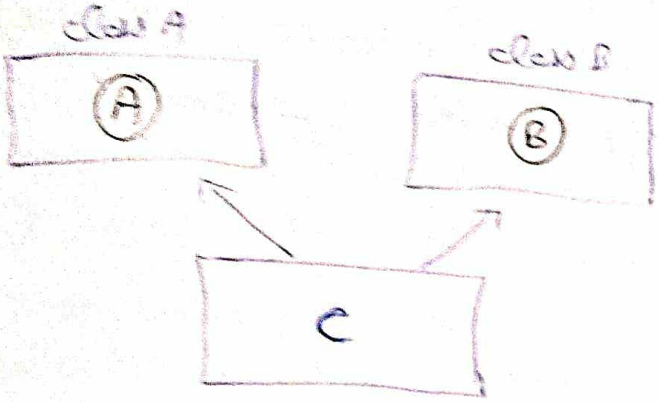
Hierarchical Inheritance



Hybrid Inheritance



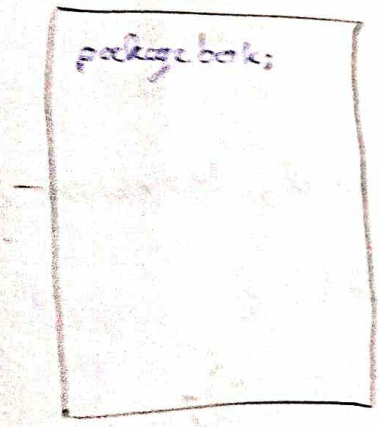
Multiple Inheritance



class C implements A, B

~~class C implements A~~

Packages




```
import java.util.*;
import book;
class ____ {
P.S.V.M{
 }
}
```

* In order to access private information we use getters and setters.

1. private
2. public
3. protected
4. Default.

Encapsulation

Encapsulation → hiding unwanted information

Abstraction

Abstraction → Don't want to Create object.

Abstract → abstract keyword.

Multiple Inheritance

```
Class Horse implements Animal, Herbivore {
    public void walk() {
    S.O.P ("Walks on 4 legs");
}
}
```

[FINISHED]

```
int arr[][] = new int[5][6];
S.o.p (arr.length); // Row count
S.o.p ( arr[0].length); // Column count
```

(*) Main method → Entry point of Program.
(public static void main (String [] args))

(*) Literals → Constant value, cannot be changed
(Pi = 3.14)
"Hellow world!"
Boolean true or false

(*) public static Vechicle (String car, String bus)
Parameters

Vechicle (Volvo, Benz)
arguments
(car) (bus)

(*) _____ is used to create an object.

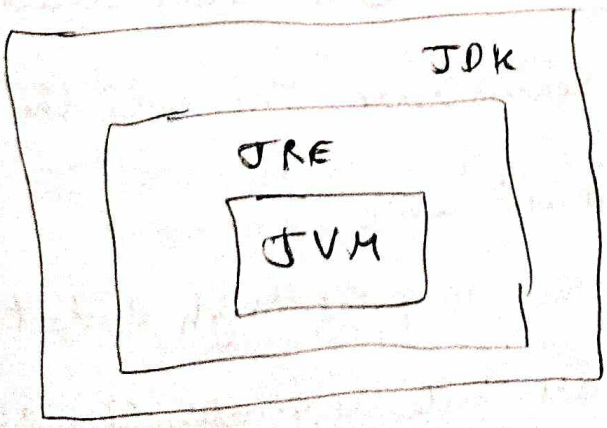
Constructor

Constructor automatically creates an object.

(*) long l = 3000000000L;
long l = 300_000_000L;
float f = 2.5F;

(*) Java is an interpreter language

(*) Java development kit → Consist of Compiler



Create class Movie with below attributes

movieName - String
company - String
genre - String
budget - int

Create class Solution and implement static method "getMovieByGenre" in the Solution class This method will

take array of Movie objects and a SearchGenre string as parameters. And will return another array of Movie objects where the SearchGenre string matches with the original array of Movie objects genre attribute.

Write necessary getters and setters.

Before Calling "getMovieByGenre" method in the main method read values for four Movie objects referring the attributes in above sequence along with a string SearchGenre. Then call the "getMovieByGenre" method and write

logic in Main Method to print "High Budget Movie", if the movie Budget attribute is greater than 8000000; else print "Low Budget Movie".

Input

aaa
Manual
Action
250000000
bbb
Manual
Comedy
350000000
ccc
Manual
Comedy
200000
ddd
Manual
Action
300000000
Action

Output

High Budget Movie
low Budget Movie

Syntax out println (" "); only works on void method

Other than void it won't work (we should use return)

Eg: public void function();
public static void main(String[]args);

public int function (int, int);

we should call both return and System.out.println("");

Question : Phone class

Constructor

* Same as class name
* whatever we update in setters just update same in getters — (getters) this.price = price — (setters)

Even If we write return a+b;
we should create an object int a = function();
and we should write S.O.P and then we should print the return value.
[Return won't print any value]

Program:

```
import java.util.*;
public class Solution
{
public static void main (String [] args)
{
Scanner in = new Scanner(System.in);
Phone [] ph = new Phone [4];
for (int i=0; i<ph.length; i++)
{
 int phoneId = in.nextInt(); in.nextLine();
 String os = in.nextLine();
 String brand = in.nextline();
 int price = in.nextInt(); in.nextline();

 ph[i] = new Phone(phoneId, os, brand, price);
}

String Brand = in.nextline();
String os = in.nextline();

int ans1 = findPriceForGivenBrand(ph, Brand);
if (ans1 != 0)
{

 System.out.println(ans1);
}
else
{
 System.out.println("The brand is not available");
}
int ans2 = getPhoneBasedOnOs(ph, os);
if (ans2 != 0)
{
 System.out.println(ans2);
```

```
else
{
  System.out.println("No phones are available with Specified OS and
                      price range");
    }
}
```

// Main Method

```
public static int findPriceForGivenBrand(Phone[]ph,
                                         String Brand)
{
  int sum = 0;
  for(int i=0; i<ph.length; i++)
  {
   if(ph[i].getBrand().equalsIgnoreCase(Brand))
   {
      sum += ph[i].getPrice();
   }
  }
  return sum;
} // System.out.println("The Total Sum Value is"+sum);
```

It works

```
public static int getPhoneBasedOnOS(Phone[]ph, String os)
{
  for(int i=0; i<ph.length; i++)
  {
```

```
if (ph[i].getOs().equalsIgnoreCase(os) && ph[i].getPrice() >= 50000)
{

  return ph[i].getId();
  // System.out.println("The Id value is"+ph[i].getId());
 }
}
return 0;
}
}
```

It works

```
public static class Phone
{
 int id, price;
 String os, brand;

 public Phone (int id, String os, String brand, int price)
 {  this.id = id;
    this.os = os;
    this.brand = brand;
    this.price = price;
 }

 public int getId()
 {
   return id;
 }
```

(There is no object So we use constructor)

```
public void setId (int id)
{
 this.id = id;
}
public int getPrice()
{
 return Price;
}
public void setPrice (int Price)
{
 this.price = price;
}
public String getOs()
{
 return os;
}
public void setOs (String os)
{
  this.os = os;
}
public String getBrand()
{  return brand;
}
public void setBrad (String brad)
{ this.brad = brad;
```

OUTPUT:

Input:

```
111
iOS
Apple
30000
222
android
Samsung
50000
333
Symbian
HTC
12000
444
Paranoid
HTC
89000
HTC
Android
```

— Setters (Input)

in.nextInt();

OUTPUT:

```
1,01,000
222
```

— Getter's (output)

return

0 1 1 2 3 5 8 13 21
0 1 2 3 4 5

Fibonacci series

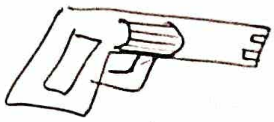
```
{
int n1= 0, n2=1, i, count=10;

S.o.p(n1 + " "+n2);

for(i=2;i<count;++i)
{
① 0 1
n3=n1+n2;
①
S.o.p(" "+n3);
① ①
n1=n2;
① ①
n2=n3;

}
}}
```

⑤ ② ③
n3 = n1+n2

② ③
n1 = n2

③ ⑤
n2 = n3

// only n1 and n2 values are updated and result = n3



Rifle Method
Loaded and Unloaded

```
      12345
   _________
10 | 123456
     10
     ---
      23
      20
     ---
       34
       30
     ---
        45
        40
     ---
         56
         50
     ---
          6
```

⑥ → rem ①

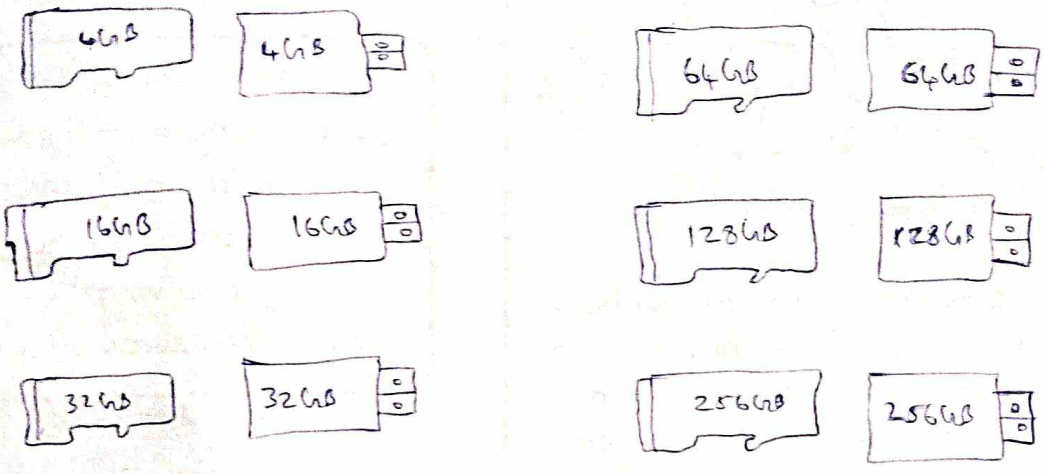
123456 > 0

⑤

Basics Binary digits

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Memory Card & Pendrive



45 → (32)(16)(8)(4)(2)(1) → 1 0 1 1 0 1

10 → (32)(16)(8)(4)(2)(1) → 0 0 1 0 1 0

ASCII → American Standard Code for Information Interchange

A = 65 a = 97

A → 65 → 1000001

2 → 10

B → 98 → 1000010

A 2 B

1 00 0001 10 1000100