30/8/24

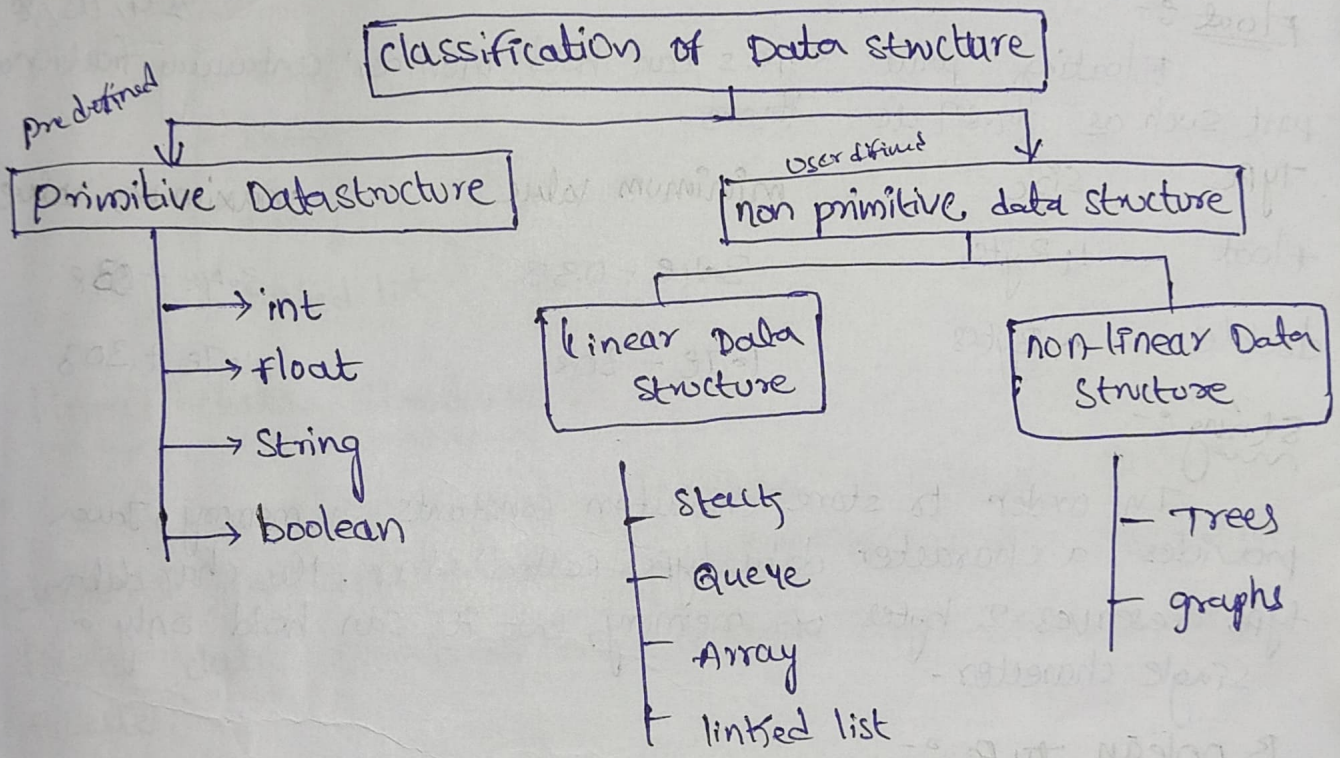
Unit-IV

Data structure

Introduction:-

Data structure is a scheme and it is particular way of storing and, retriving, viewing and organising the data in the system memory. Here data is nothing but collection of raw fact. such as audio, ~~video~~ Video, text and images. Data structures are used to create different formations in the memory.

Classification of Data Structure:-



Primitive Data Structure.

Primitive data structures are also known as scalar built in, Basics and fundamental data types. This predifined data type already exist in any programming languages. such as C, C++, Java and so on....

Integer Types :-

Integer types can hold whole numbers such as 123, -96 etc.. The size of the values that can be stored depends on the integer data type we choose. Java doesn't support concept of unsigned.

Type	size	minimum value	maximum value
Byte	1 Byte	-128	127
short	2 Bytes	-32,768	32,767
Int	4 Bytes	-2,147,483,648	2,147,483,647
long	8 Bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Float :-

Floating point types can hold number containing fractional part such as 27.59 etc.. ~~there~~

Type	size	minimum value	maximum value
float	4 Bytes	3.4e-038	3.4e+038
double	8 Bytes	1.7e-308	1.7e+308

String :-

In order to store character constants in memory, Java provides a character data type called 'char'. The char data type reserves 2 bytes of memory but it can hold only a single character.

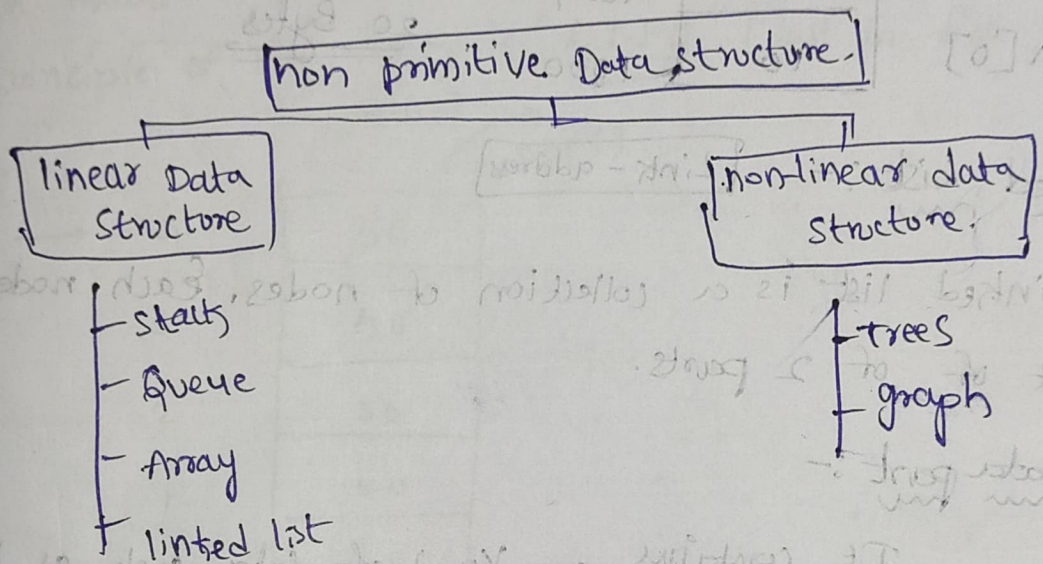
Boolean type :-

Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a Boolean type can hold; true or false. Boolean type is denoted by the keyword 'boolean' and uses only one bit of storage.

non primitive Data Structure:-

Non primitive data structures are also known as ADT's (Abstract data type). Non primitive data structure is classified into two types.

1. Linear
2. Non-linear.



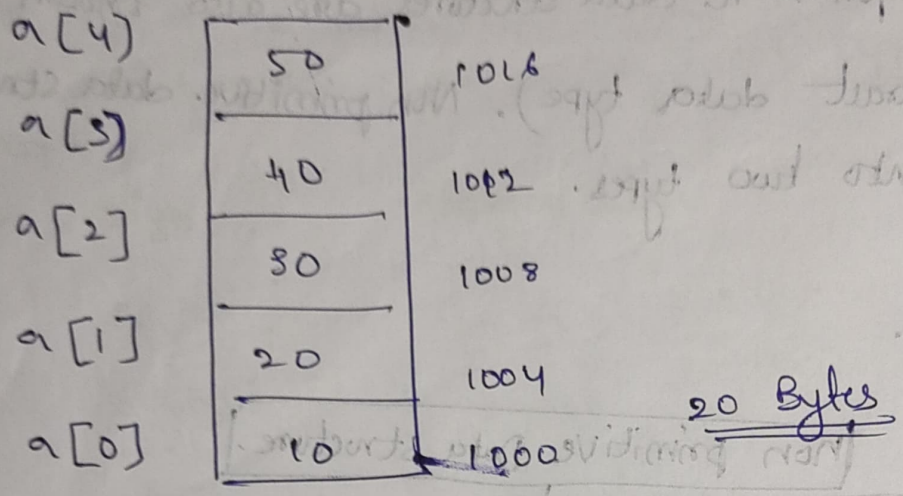
linear data structure :-

Linear data structure is used to perform elements are arranged in sequencial order. (or) linear order. linear data structures are arrays, stacks, linked list, queue.

1. Arrays :-

An array is a linear order collection of elements of same type. that share a common name. An ele array elements are stored in contiguous memory allocation. It is fixed size in the memory. Thats why it is static data structure.

a[5] = {10, 20, 30, 40, 50};



2. Linked List :- Link - address

linked list is a collection of nodes. Each node consist of of 2 parts.

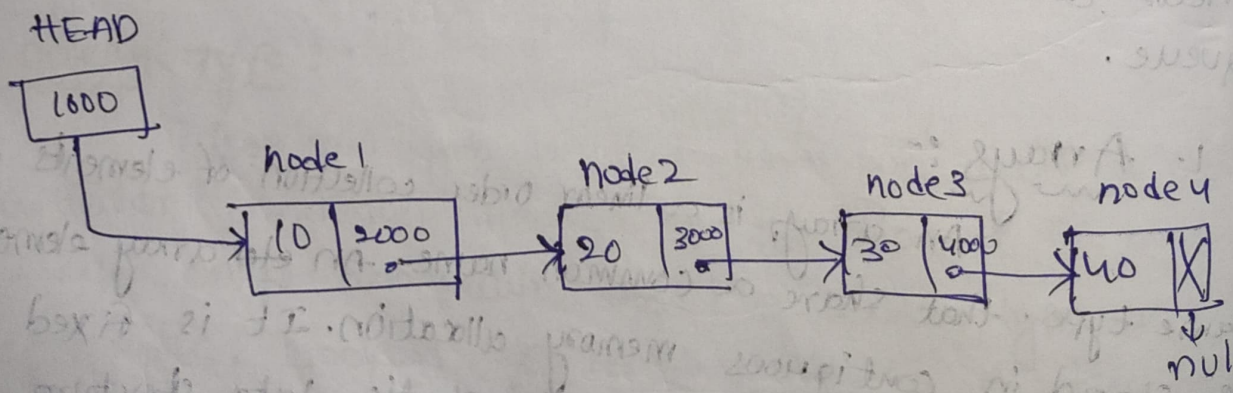
1. Data part :-

It contains any kind of data such as Image, Audio, Video, text.

2. Link part :-

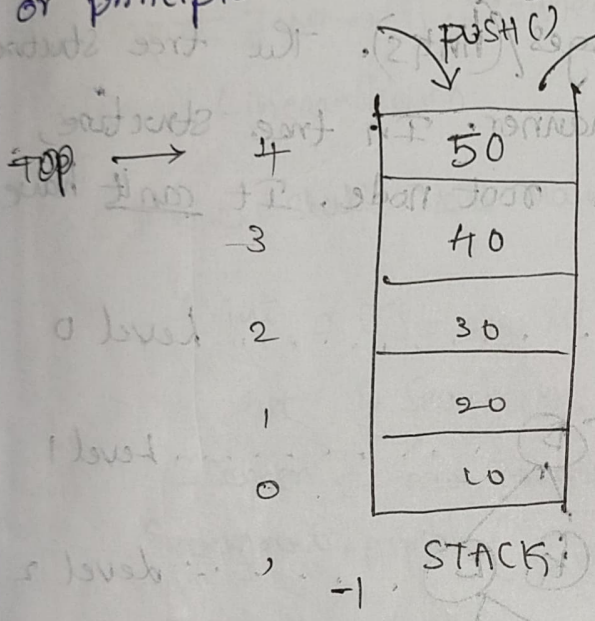
It contains address of another node.

It is flexible size and it is a dynamic data structure



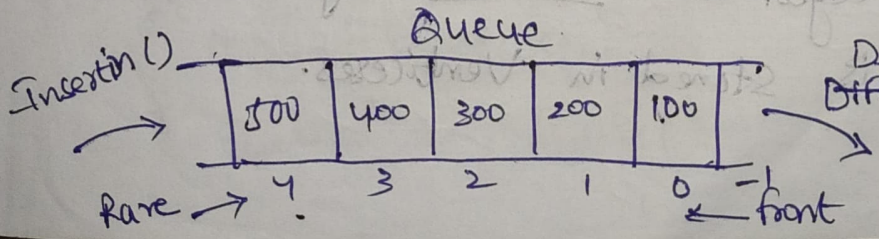
3. Stack:-

A Stack is a linear order collection of elements in which elements are inserted and deleted at only one end point. Inserting an element at top of the stack is called "push()" and Deleting an element at top of the stack is called "POP()". It has only one pointer called "Top". which means indicates the indexes of the stack. An order or principle of the stack is "LIFO". (last in first out).



4. Queue:-

Queue is a linear order collection of elements in which elements are inserted at one end and deleted at another end. It has An inserting an element at the rare of the Queue is called "insertion()". and deleting an element at the Front of the Queue is called "Deletion()". It has two pointers called "Front" and "Rare". An order or principle of the Queue is "FIFO"(First in first out).

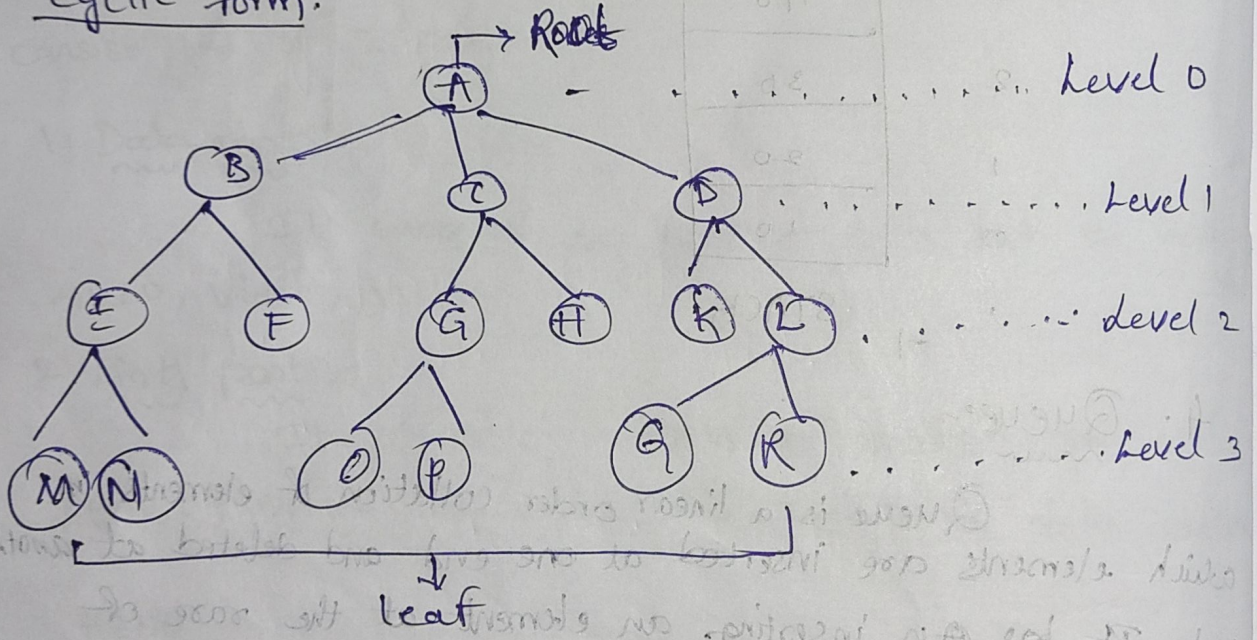


(9) Non-linear Data Structure :-

Non-linear data structures are used to perform rearrenged the elements in a random order. or non-linear order. there are two ways of data structures such as. 1. Trees and 2. Graphs.

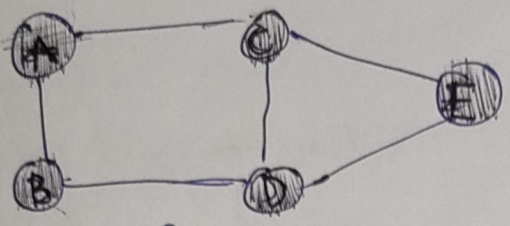
1:- Trees :-

A tree is a finite set of elements called nodes. These nodes connected by edges (links). The tree structure formation in hierarchical manner. In tree structure, the unique node is called root node. It can't have "cyclic form."



2:- Graphs :-

A Graph is a collection verticeses which are connected by edges. It can have "cyclic form". In graph, data is stored in "Verticeses".



G

$G = \{V, E\}$

$V(G) = \{A, B, C, D, E\}$ → set of vertices

$E(G) = \{(A,B)(A,C), (B,D) (C,D) (C,E) (D,E)\}$.

Arrays:-

Linear Search :-

```
import java.util.Scanner;
class Linear-Search
{
  public static void main(String args[])
  {
    int a[]= new int[50];
    int i, search, n;
    Scanner s= new Scanner(System.in);
    System.out.println("Enter n value = ");
    n= s.nextInt();
    System.out.println("enter an elements");
      for (i=0 ; i<n ; i++)
    {
      a[i] = s.nextInt();
    }
    System.out.println("Enter Search element");
    Search = s.nextInt();
    for (i=0 ; i<n ; i++)
    {
      if (a[i]==search)
      {
        System.out.println(search +" is found in ten
                                    list");
```

```
        break;
      }
    }
    if (i==n)
    {
      System.out.println(search + "It is not found in the list");
    }
  }
}
```

Binary Search :-

```
import java.util.Scanner;
Class Binary_search
{
  Public Static void main (String args[])
  {
    int a[] = new int[50];
    int i, search, n, first, middle, last;
    Scanner s = new Scanner(System.in);
    System.out.println("Enter n value");
    n = s.nextInt();
    System.out.println("enter an elements");
      for(i=0; i<n; i++)
      {
        a[i] = s.nextInt();
      }
      System.out.println("enter search element");
      search = s.nextInt();
```

```
first=0;
last=n-1;
middle=(first+last);
while(first<last)
{
  if(a[middle]<search)
  {
    first=middle+1;
  }
Else if (a[middle]==search)
{
  System.out.println(search+" is ~~not~~ found in the list");
  breaks;
}
else
  {
    last=middle-1;
  }
  middle=(first+last);
  }
If (first>last)
  {
  System.out.println(search+" is not found in the list");
}
}
}
```

Bubble Sorting :-

Bubble sort is used to rearrage the elements in the particular order such as ascending order and descending order. The time complexity of bubble sorting is Big Oh(n^2). It is belongs to under "worst case".

Procedure :-

Step1 :-

a[1]		a[2]		a[3]		a[4]
50	>	40		30		20
40		50	>	30		20
40		30		50	>	20
40		30		20		50

Step2 :-

40	>	30		20		50
30		40	>	20		50
30		20		40	>	50
30		20		40		50

Step3 :-

30	>	20		40		50
20		30	>	40		50
20		30		40	>	50
20		30		40		50

Step4 :-

20	>	30		40		50
20		30	>	40		50
20		30		40	>	50
20		30		40		50

Program:-

```
import java.util.Scanner;
class bubblesort
{
  public static void main(String args[])
  {
    int a[] = new int[50];
    int n,i;
   Scanner s = new Scanner(System.in);
   System.out.println("\n Enter the number elements for sorting\n");
   n= s.nextInt();
 System.out.println("\n Enter an elements\n");
 for(i=0; i<n; i++)
  {
     a[i] = s.nextInt();
  }
  System.out.println("\n before sorting\n");
   for(i=0; i<n; i++)
    {
     System.out.println("\n" +a[i]);
    }
  Bsort(n,a);
  System.out.println("\n after sorting\n");
   for(i=0; i<n; i++)
   {
     System.out.println("\n" +a[i]);
   }
}
public static void Bsort(int n,int a[])
  {
     int i, j, temp;
```

```
for(i=0; i<n-1; i++)
{
    for(j=0; j<n-i-1; j++)
    {
        if(a[j]>a[j+1])
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
        }
    }
}
```

Selection Sort :-

Selection Sort is used to perform rearranged an elements in the particular. Such as Ascending or Descending order. The time complexity of selection sort is **Big O^n (n)**

Because of it reduces number of swapping. If it is more efficiency of than bubble sort. It belongs to order worst case.

Eg:- a[6] = {50, 25, 61, 44, 3, 6}

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
Pass=0, Loc=4	50	25	61	44	2	6
pass=1, Loc=5	2	25	61	44	50	6
pass=2, Loc=5	2	6	61	44	50	25
pass=3, Loc=3	2	6	25	44	50	61
pass=4, Loc=4	2	6	25	44	50	61
pass=5, Loc=5	2	6	25	44	50	61
Sorted order	2	6	25	44	50	61

Program :-

→ Same as bubble sort

```
import java.util.Scanner;
class Public static void ssort(int a[], int n)
{
    int min, loc, i, j;
    for(i=0; i<n-1; i++)
    {
        min = a[i];
        loc = i;
        for(j=i+1; j<n; j++)
        {
            min=a[i]; if(min>=a[j])
                {
                    min=a[j];
                    loc=j;
                }
        }
    int t=a[i];
    a[i]=a[loc];
    a[loc]=t;
}}
```

Insertion Sort :-

Insertion sort is used to perform re-arrange the elements in particular order such as ascending & descending order. Insertion sort is more efficiency then selection sort and bubble sort. Because it is also reduse no. of swappings time complexity of insertion sort Big O(log n). It belongs to under "worst case"

Procedure :-

Index	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
S1	27	34	2	8	1	17	14	29
2	27	34	2	8	1	17	14	29
3	27	34	2	8	1	17	14	29
4	2	27	34	8	1	17	14	29
5	2	8	27	34	1	17	14	29
6	1	2	8	27	34	17	14	29
7	1	2	8	17	27	34	14	29
8	1	2	8	14	17	27	34	29
sorted order	1	2	8	14	17	27	29	34

Insertion sort program:-

```
import java.util.*;
class insertionsort
{
public static void main (String args[])
 {
   int a[]= new int[50];
   int n,i;
   Scanner s= new .scanner(System.in);
   System.out.println("Enter no.of elements");
    n= nextInt();
   System.out.println("Enter array elements");
   for(i=0; i<n; i++)
    {
      a[i] = s.nextInt();
    }
  System.out.println("Before sorting");
   for(i=0; i<n; i++)
   {
     System.out.println("\n" +a[i]);
   }
isort(a,n);
System.out.println("After sorting");
for(i=0; i<n; i++)
 {
  System.out.println("\n" +a[i]);
 }
Public static void isort(int a[], int n)
{
 int t,i,j;
 for(i=0; i<n; i++)
  {
    j=1
```

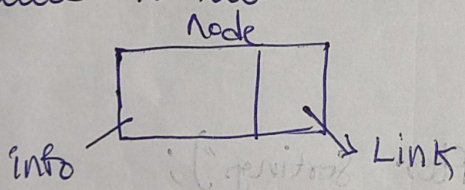
```
        t=a[i];
        while ((j>0 && a[j-1])>t)
        {
            a[j] = a[j-1]
            j--;
        }
        a[j]=t;
    }
}
```

Linked list :-

Creation of node :-

```
class node
{
    int info;
    node link;
}
node node1 = new node();
```

Representation of node :-



info :- contains data.

Link :- contains address of another node

```
node1.info = Data(20)
node1.link = NULL
```

Operations on linked list :-

linked list are classified into 2 types of operations such as
1. Insertion
2. Deletion

1. Insertion :-

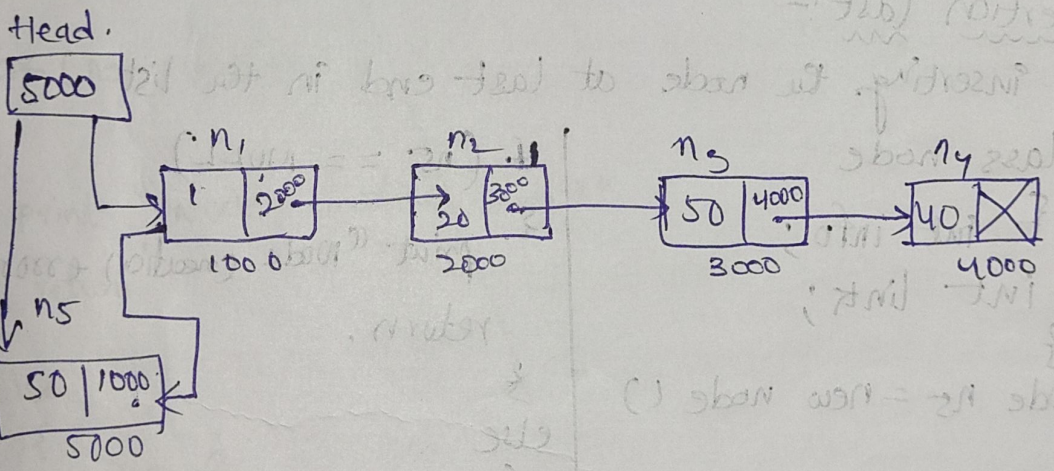
Insertion operation is used to perform inserting the node in any where in the list. Insertion operation is classified into 3 categories

1. Insertion first
2. Insertion middle
3. Insertion last

1. Insertion First :-

An inserting the node at first end in the list

```
class node
{ int info;
  node link;
}
node n5 = new (node) node ();
```



Source Code :-

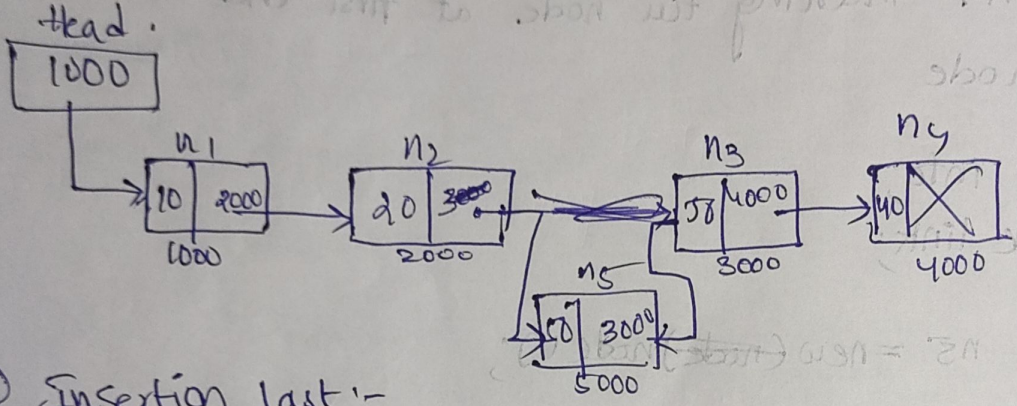
```
if (n5 == NULL)
{
  print "Node creation error in the memory"
  return
}
else
{
  n5.info = 50
  n5.link = Head.link
```

head = n5
return
}

(2) Insertion middle :-

An inserting the node at middle end in the list

```
class node
{
    int info;
    node link;
}
node n5 = new node();
```

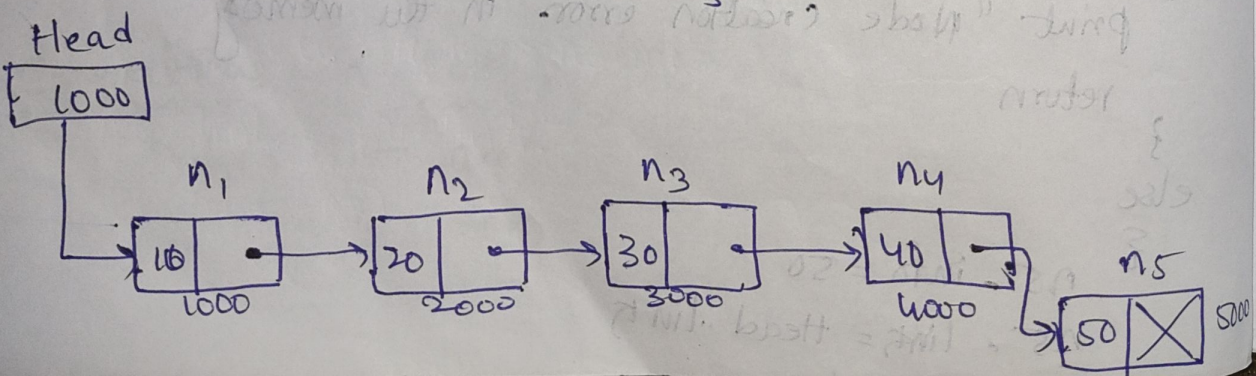


(3) Insertion last :-

An inserting the node at last end in the list

```
class node
{
    int info;
    int link;
}
node n5 = new node()
```

```
if (n5 == NULL)
{
    print "Node creation error"
    return.
}
else
{
    n5.info = data(50)
    n5.link = NULL
    n4.link = n5
    return
}
```



2. Deletion operation :-

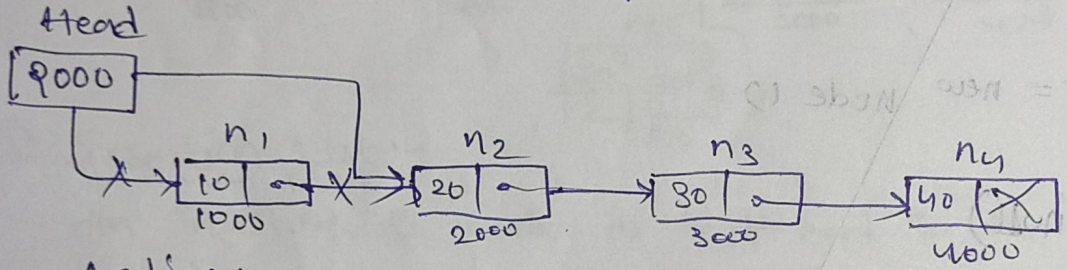
Deletion operation is used to perform deleting the node from anywhere in the given list.

Deletion operation is classified into 3 ways.

1. Deletion first :-
2. Deletion middle
3. Deletion last

1. Deletion first :-

Deleting the node at first position from the list. That node removed from memory.



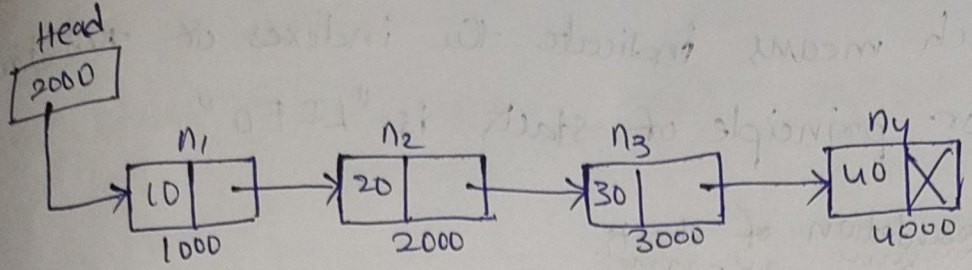
Coding :-

```
print "Deleted Item → n1.info"
Head = n2.link
Head = n2
free (n1)
```

2. Deletion middle :-

Deleting the node at middle position.

```
print "Delete Item → n3.info"
n2.link = n4.link.
free (n3)
```



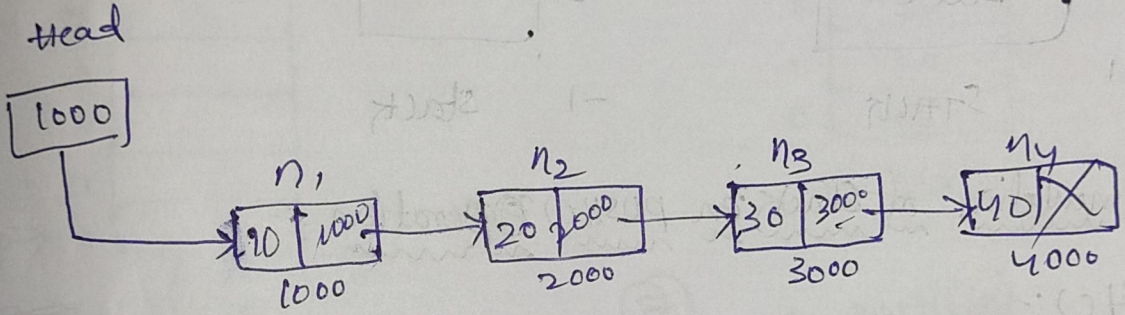
3. Deletion Last :-

Deleting the node at last position,

print "Delete item → n4.info."

n3.link = NULL

free (n4)



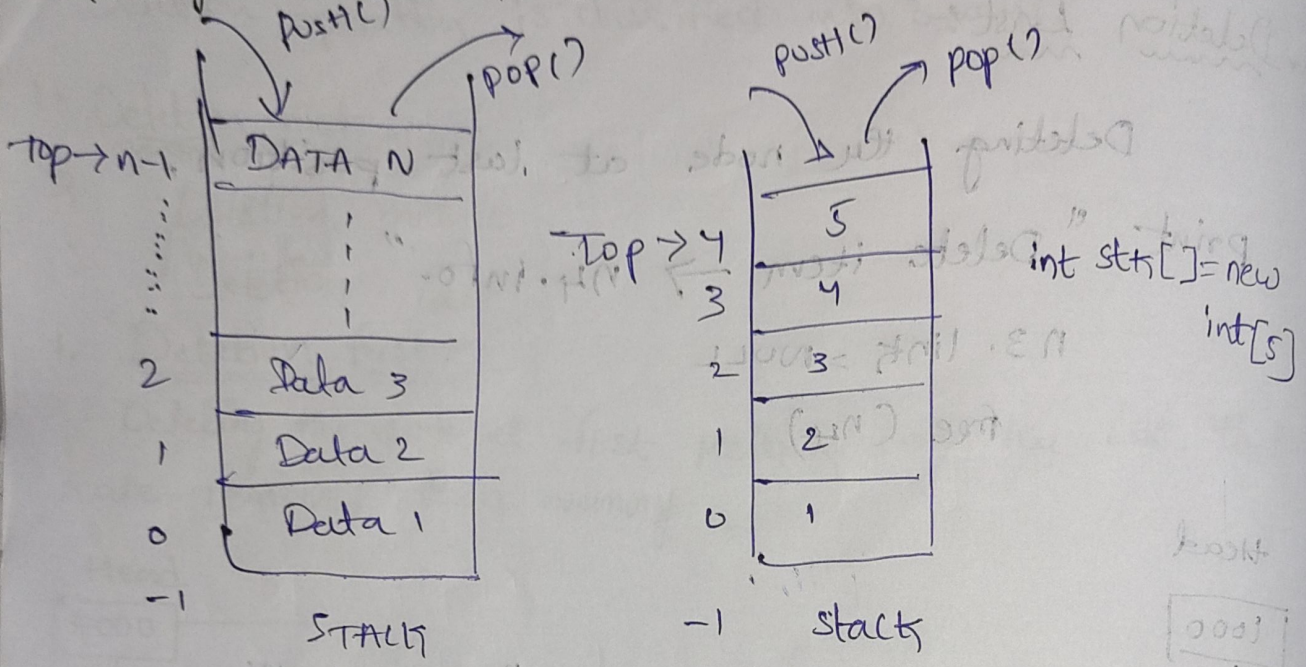
STACKS :-

A stack is a linear order collection of elements in which elements are inserted and deleted at only one end point. Inserting an element at top of the stack is called "push ()" and deleting an element at top of the stack is called "pop ()". It has only one pointer called

"Top", which means indicate the indexes of the stack.

An order or principle of stack is "LIFO".

memory representation of stack.



Implimentation of stack on PUSH() Operation :-

PUSH() :-

PUSH operation is used to perform inserting an element into the ~~stack~~ top of the stack.

Overflow :-

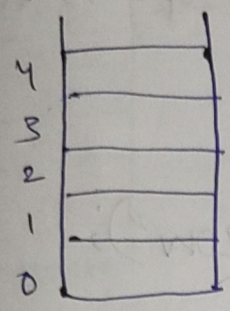
If stack is full then flash the message is "STACK IS OVERFLOW"

TOP :-

Top is a pointer, and it indicates the indexes of top of the element. In PUSH() operation, TOP is increased

operations on PUSH() :-

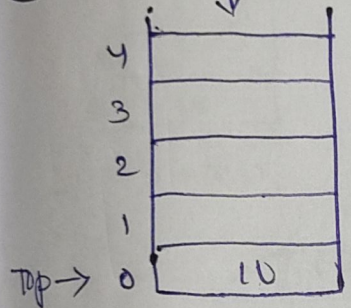
① int stack[] = new int[5]



Top → -1 STACK

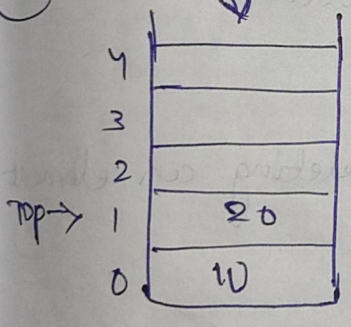
STACK is Underflow

② PUSH(10)



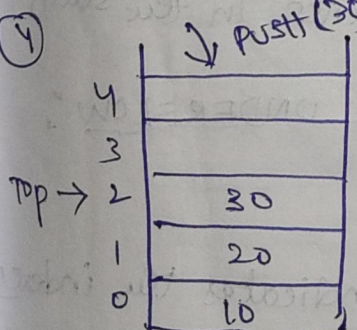
~~Top~~ -1 STACK

③ PUSH(20)



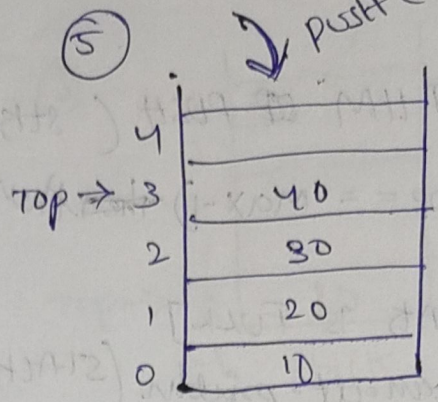
-1 STACK

④ PUSH(30)



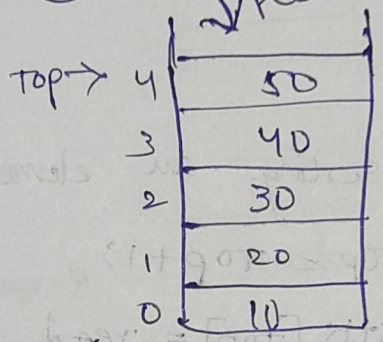
-1 Stack.

⑤ PUSH(40)



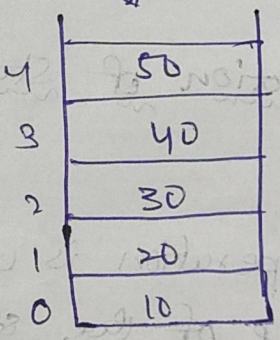
-1 Stack.

⑥ PUSH(50)



-1 Stack.

⑦ PUSH(60)



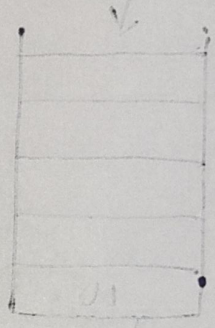
-1 Stack.

STACK is overflow

Algorithm of PUSH() :-

ALGORITHM PUSH (stk[max], max=5, TOP=-1, item)

```
If (TOP == MAX-1) then
{
  [STAK is FULL]
  System.out.println (STACK IS OVERFLOW");
  return;
}
else
{
  [Inserting an element into stack]
  top = top+1;
  stk [TOP] = read item;
  return;
}
}
```



Implimentation of stack on pop operation :-

POP() :-

pop operation is used to perform deleting an element from the top of the stack.

Underflow() :-

If doesn't have any elements in the stack then flash the message is "STACK IS UNDERFLOW".

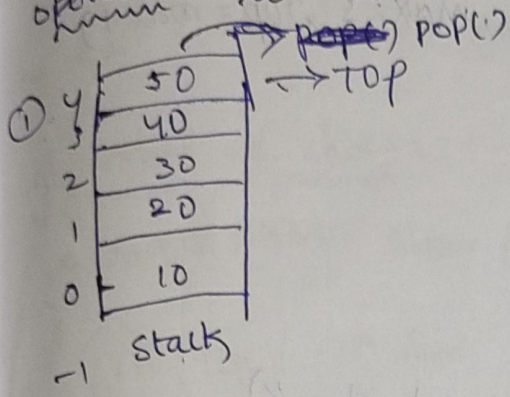
In pop operation TOP

TOP :-

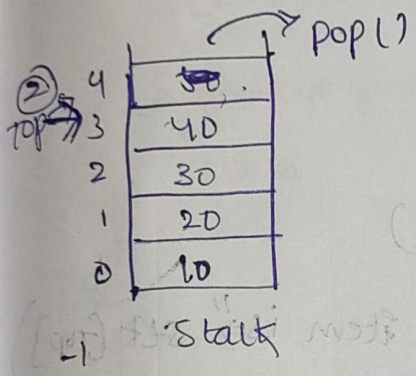
It is a pointer and it indicates the index of top of the element. In POP() operation

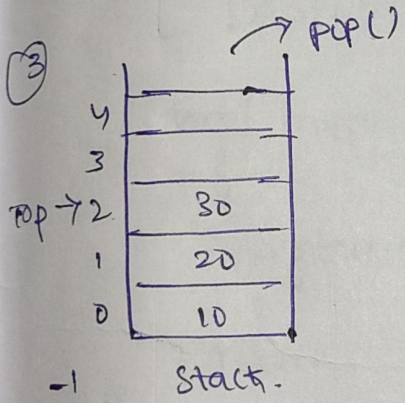
Top is decreased.

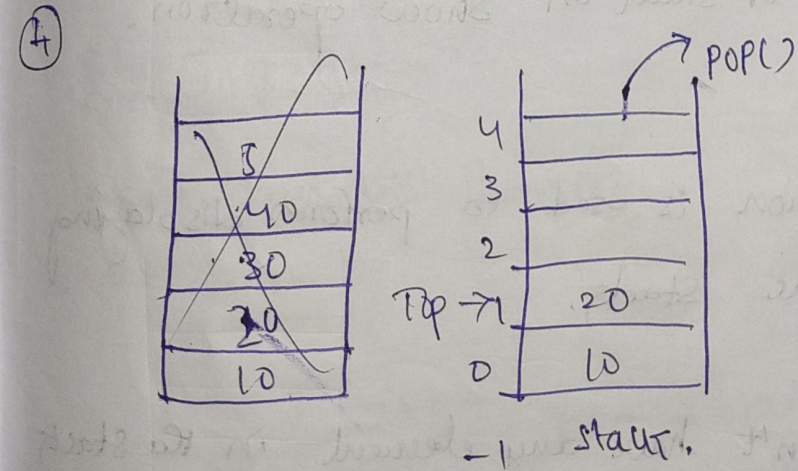
operation on Pop():-

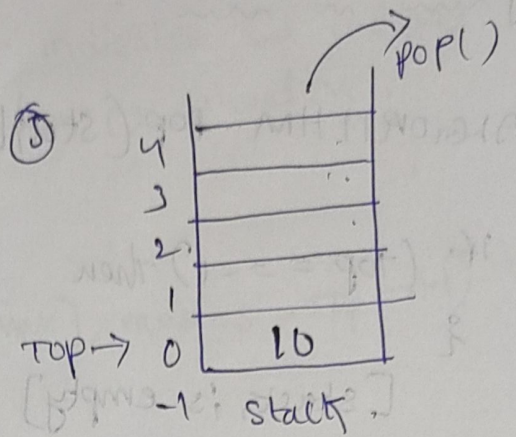
① 

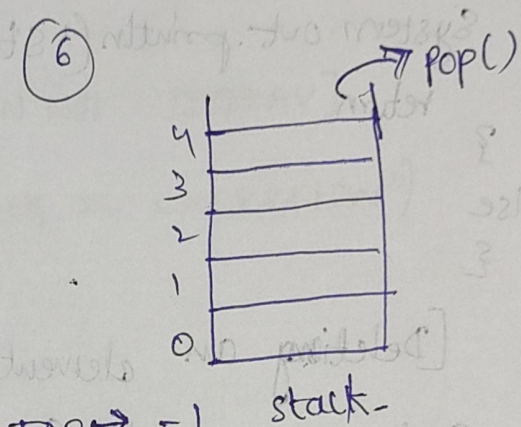
Stack is overflow

② 

③ 

④ 

⑤ 

⑥ 

Stack is underflow.

Algorithm of POP():-

ALGORITHM POP(stack[max], max=5, TOP=max-1)

{

if (top == -1) then

{

[stack is empty]

System.out.println("Stack is underflow")

return

}

else

{

[Deleting an element in stack]

System.out.println("Deleting an item is=>" stk[top])

Top = Top-1

return

}

}

Show(): Algorithm:-

ALGORITHM

Implementation of stack on Show operation.

Show:-

Show operation is used to perform displaying all elements of the stacks.

Underflow:-

If doesn't have any element in the stack then flash the message is "STACK IS UNDERFLOW"

top():-

It is a pointer and it indicates the indexes of top of the element. In ~~pop()~~ show() operation.

Algorithm of show():-

ALGORITHM show (stk[max], max=5, TOP=n-1)

{

if (TOP==-1) then.

{

[ELEMENTS ~~CAN~~ CAN NOT DISPLAY]

System.out.println("STACK IS UNDERFLOW")

return

}

else

{

[DISPLAY ALL ELEMENTS OF STACK]

for (temp= top; temp>=0; temp--)

{

System.out.println (stk[temp])

}

} return.

}

Stack program:-

```
import java.util.Scanner;
class stack
    {
        int stk[], top, max=5;
    Scanner s = new scanner (system.in);
        stack()
```

```
{
    stk = new int [max]
    top = -1;
}
void push()
  {
    if (top == max-1)
      {
        System.out.println (" stack is overflow\n")
        return;
      }
    else
     {
       top = TOP + 1;
       stk [top];
      System.out.println ("Enter an element\n");
       stk [top] = s.nextInt();
     System.out.println ("Insertion successfully\n");
      return;
     }
}
void pop()
  {
    if (top == -1)
    {
      System.out.println ("stack is underflow\n");
      return;
    }
  else
    {
      System.out.println("\n Deleted item is => "
                          + stk[top]);
```

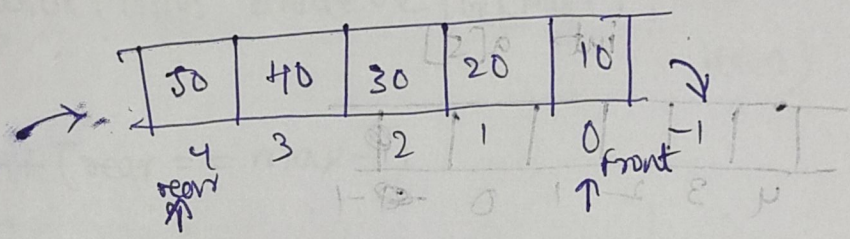
```
top= top-1;
return System.out.println("Deletion successfully \n");
return;
}
}

void show()
{
    if (top==-1)
    {
    System.out.println("\n stack is underflow \n");
     return;
    }
   else
    {
      int temp;
      for (temp=top; temp>=0; temp--)
      {
        System.out.println(stk[temp]);
      }
     return;
    }
}
void menu()
   {
      int choice
      do
       {
        System.out.println("\n****main menu****\n");
       System.out.println("1. push() operation\n");
       System.out.println("2. pop() operation\n");
       System.out.println("3. show() operation\n");
       System.out.println("4. Exit() operation\n");
```

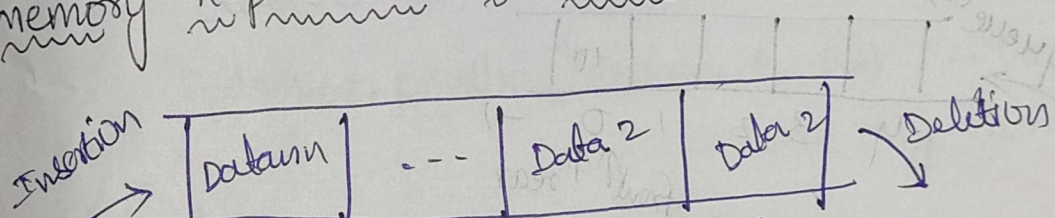
```
System.out.println("Enter your choice\n");
choice = s.nextInt();
switch (choice)
    {
      case 1 : push();
               break;
      case 2 : pop();
               break;
      case 3 : show();
               break;
      default : system.out.println("sorry! wrong option
    } while (choice != 4);
}
Publich static void main (String args[])
{
    Stack obj = new Stack();
    obj.menu();
}
}
```

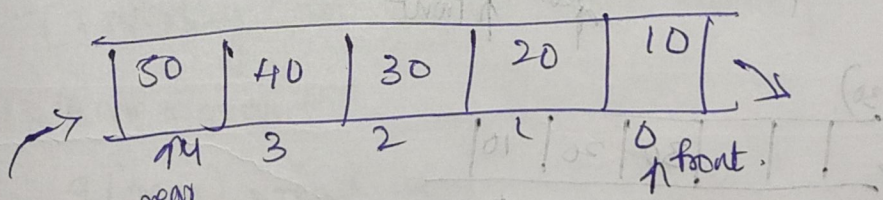
Queue:-

Queue is a linear order collection of elements in which element are inserted at one end and deleted at other end. An inserting an element at the rear of the queue is called "Insertion operation" and deleting an element at the front of the queue is called "deletion operation" it has two end pointers called "Front and rear". An order (or) principle of the queue is "LIFO" (last in first out)



memory Representation of Queue:-





Implementation of queue on Enqueue operation:-

Enqueue []:-

Enqueue operation is used to perform inserting an element into the rear of the queue.

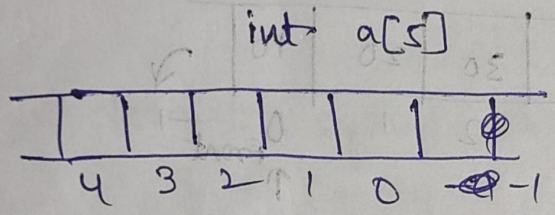
Is full :-

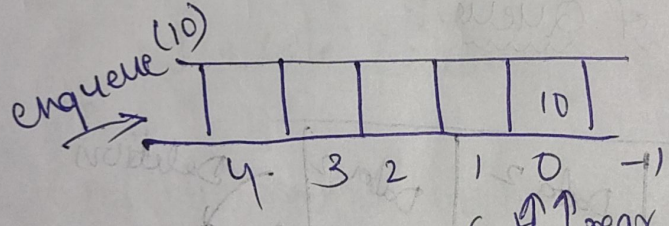
If queue is full then can't insert any element into the queue. Here, flash the message is "Queue is full".

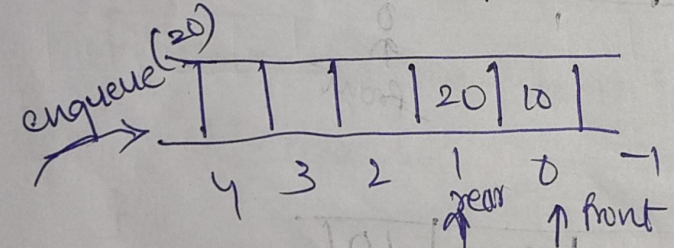
Rear :-

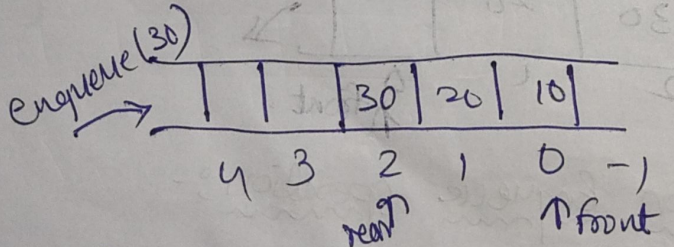
Rear is the "pointer" and it indicates the index. In enqueue operation rear pointer is increased by "1".

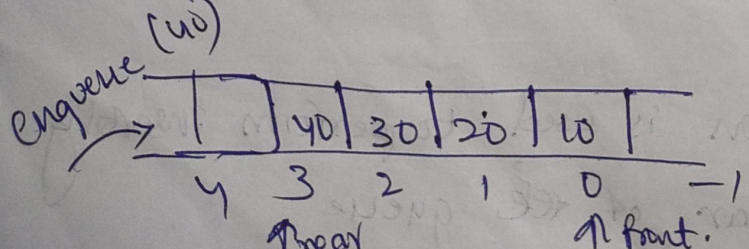
Procedure :-

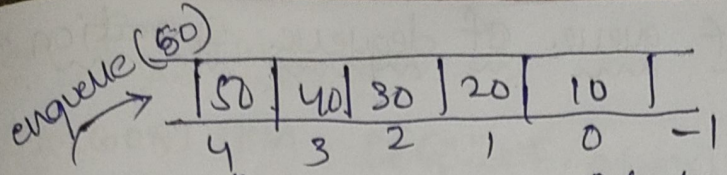


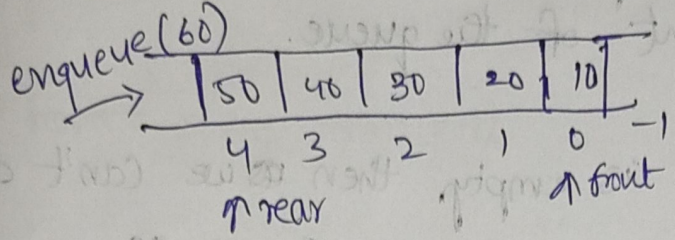












"Queue is full"

Algorithm of Queue :-

```
ALGORITHM ENQEUE (Q[max], max=4, front=-1,
                  item)
{
  if (rear == max-1)
  {
    [QUEUE IS FULL)
    System.out.println ("QUEUE is full --- can not
                         Insert an any element");
    return
  }
  else
  {
    [Inserting an element into queue]
    rear = rear + 1
    q(rear) = read item
    if (front == -1)
    {
      front = 0
    }
    return
  }
}
```

Deimplementation of queue of dequeue operation :-

Dequeue():-

Dequeue operation is used to perform deleting an element at front of the queue.

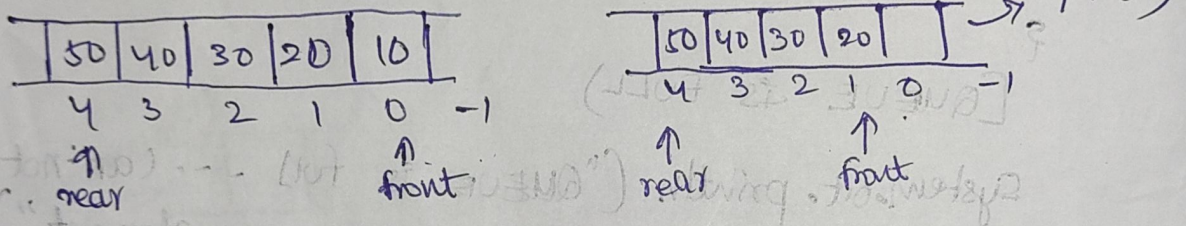
Is empty:-

If queue is empty then we can't delete here. Display the message is "empty".

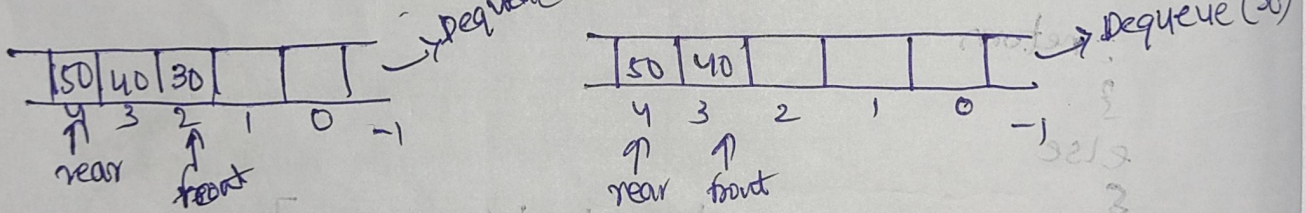
Front :-

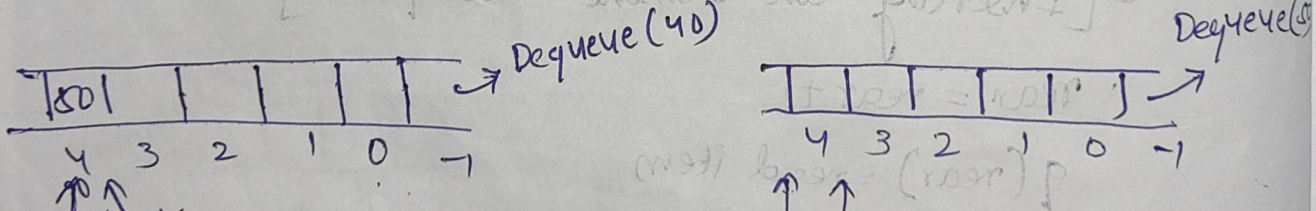
Front is a pointer and it indicates the starting index (0) in dequeue operation front increased by "1".

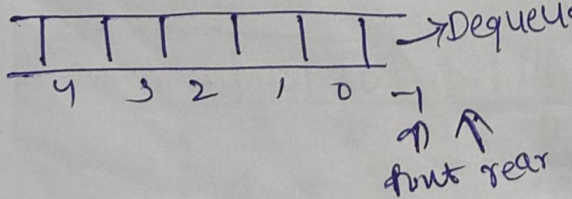
Procedure :-



Queue is full







Queue is empty.

Algorithm of dequeue:-

ALGORITHM DEQUEUE(Q[max], max=n, front=0, rear=n-1)

{

if (front == -1 && rear == -1)

{

[Queue is empty]

System.out.println("Queue is empty... Can't Delete")

return.

}

else

{

[Delete an element from the queue]

System.out.println("The deleted item => Q[front]")

front = front + 1

if (rear + 1 == front)

{

front = -1

rear = -1

}

return

}

}

Show() operation

Show operation is used to perform displaying all elements of the queue

Is empty.

If queue is empty at then can't show an element of the queue.

Front:-

Front is a pointer and it indicates the starting index(0) in dequeue operation front increased by "1".

Algorithm of show():

```
Algorithm show(Q[max], max=n, front=0, rear=n-1)
{
  if (front==-1 && rear==-1)
  {
    [queue is empty]
System.out.println("Queue is empty.. can not show")
   return
}
else
 {
  [Display All elements of the queue]
  for(temp=front; temp<=rear; temp++)
   {
     System.out.println(Q[temp])
   }
  return
 }
}
```

Queue programming :-

```
import java.util.Scanner;
class queue
{
  int Q[], front, rear, max=5;
  Scanner s= new Scanner(System.in);
  queue()
  {
    Q= new int[max];
  }
  void Enqueue()
  {
    if (rear == max-1)
    {
      System.out.println("Queue is full \n");
      return;
    }
    else
    {
      rear= rear+1;
      System.out.println("\n Enter an element \n");
      Q[rear] = s.nextInt();
      System.out.println("Insertion successfully. \n");
      if (front == -1)
      {
        front=0;
      }
      return;
    }
  }
  Void Dequeue()
  {
    if (front == -1 && rear == -1)
```

```
{
System.out.println("Queue is empty\n");
return;
}
else
{
System.out.println("\n Deleted item is ==> "+Q[front]);
front = front+1;
System.out.println("Deletion successfully\n");
if (rear+1 == front)
{
front = rear = -1;
}
return;
}
}
void show()
{
if (front == -1 && rear == -1)
{
System.out.println("\n Queue is empty\n");
return;
}
else
{
int temp;
System.out.println("\n Queue elements are\n");
for (temp = front; temp <= rear; temp++)
{
system.out.println(Q[temp]);
}
```

```
return;
}
}
void menu()
{
int choice;
do
{
System.out.println("\n *** Main menu ***\n");
System.out.println("1. Enqueue()\n");
System.out.println("2. Dequeue()\n");
System.out.println("3. show()\n");
System.out.println("4. Exit()\n");
System.out.println("Enter your choice\n");
choice = S.nextInt();
switch (choice)
{
case 1 : Enqueue();
break;
case 2 : Dequeue();
break;
case 3 : show();
break;
Default : System.out.println("Sorry! wrong option\n");
}
} while (choice != 4);
}
```

```
Public static void main ( String args [])
{
    queue obj = new queue();
      obj.menu();
    }
}.
```