```
class Stack
{

        private int max;
        private long[] S1;
        private int top;

        Stack(int s)
        {
                top=-1;
                max=s;
                S1=new long[max];
        }

        public void push(long x)
        {
                S1[++top] = x;
        }

        public long pop()
        {
                return S1[top--];
        }

        public long peek()
        {
                return S1[top];
        }

        public boolean isEmpty()
        {
                return (top == -1);
        }

        public boolean isFull()
        {
                return (top >= (max-1));
        }

        public void display()
        {
                for(int i=0;i<=top;i++)
                {
                        System.out.println(S1[i]);
                }
        }

}

class StackApp
```

```
{

        public static void main(String args[])
        {
                Stack s1 = new Stack(5);
                s1.push(11);
                s1.push(21);
                s1.push(31);
                s1.push(41);
                s1.push(51);
                s1.display();

                s1.pop();
                s1.display();
                s1.pop();
                s1.display();
                s1.pop();
                s1.display();


        }
}
```

```java
class Stack
{
        static final int Max=10;//size of stack
        int top;
        int s[] = new int[Max];//Maximum size of stack

        Stack()
        {
                top = -1;// to initialize the top
        }

        //to check stack is empty
        boolean isEmpty()
        {
                return (top < 0);
        }

//To perform insertion operation : PUSH
        boolean push(int x)
        {
                if(top >= Max-1){
                        System.out.println("Overflow !!!");
                        return false;
                }
                else {

                        s[++top] = x;
                        System.out.println(x+"----> Push operation!!!");
                        return true;

                }

        }
//To perform deletion operation : POP

        int pop()
        {
                if(top < 0){
                System.out.println("Underflow !!!");
                return 0;
                }
                else{
                        int x = s[top--];
                        return x;
                }
        }

//To get the current status of TOP
int peek()
{
        if(top < 0){
                System.out.println("Underflow !!!");
                return 0;
                }
        else{
                int x = s[top];
                return x;
        }
}


        public static void main(String args[])
        {
                Stack s1 = new Stack();
                s1.push(10);
                s1.push(20);
                s1.push(30);
                System.out.println(s1.pop()+ " Popped from stack !");

        }
}
```

```
class Sorting
{
        void bubbleSort(int a1[])
{
        int n = a1.length;
        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-i-1;j++)
                {
                        if(a1[j] > a1[j+1])
                        {
                                int temp = a1[j];
                                a1[j]=a1[j+1];
                                a1[j+1]=temp;
                        }
                }

        }
}

void selectionsort(int a1[])
{
        int n = a1.length;
        for(int i=0;i<=n-1;i++)
        {
                int min=i;
                for(int j=i+1;j<n;j++)
                {
                        if(a1[j] < a1[min])
                                min = j;
                }
                //swapping
                                int temp = a1[min];
                                a1[min] = a1[i];
                                a1[i]=temp;



        }

}

void insertionsort(int a1[])
{
        int n = a1.length;
        for(int i=1;i<n;i++)
        {
                int k = a1[i];
                int j = i-1;
                while(j>=0 && a1[j]>k)
                {
                        a1[j+1]=a1[j];
                        j=j-1;


                }
                a1[j+1]=k;
        }
}

        void display(int a1[])
        {
                int n = a1.length;
                for(int i=0;i<n;i++)
                {
                        System.out.print(a1[i]+" ");
                }
        }




                public static void main(String args[])
        {

                Sorting s1 = new Sorting();
                int a1[]={5,3,8,4,6,9,2,7};
                //s1.bubbleSort(a1);
                //s1.selectionsort(a1);
                s1.insertionsort(a1);
                s1.display(a1);



        }
}
```

```
class Sorting
{
        void bubbleSort(int a1[])
{
        int n = a1.length;
        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-i-1;j++)
                {
                        if(a1[j] > a1[j+1])
                        {
                                int temp = a1[j];
                                a1[j]=a1[j+1];
                                a1[j+1]=temp;
                        }
                }

        }
}

void selectionsort(int a1[])
{
        int n = a1.length;
        for(int i=0;i<=n-1;i++)
        {
                int min=i;
                for(int j=i+1;j<n;j++)
                {
                        if(a1[j] < a1[min])
                                min = j;
                }
                //swapping
                                int temp = a1[min];
                                a1[min] = a1[i];
                                a1[i]=temp;



        }

}
public void heapsort(int a1[])
        {
                int n = a1.length;


                for (int i = n / 2 - 1; i >= 0; i--)
                        heapify(a1, n, i);
                for (int i = n - 1; i > 0; i--) {

                        int temp = a1[0];
                        a1[0] = a1[i];
                        a1[i] = temp;


                        heapify(a1, i, 0);
                }
        }


        void heapify(int a1[], int n, int i)
        {
                int largest = i;
                int l = 2 * i ;
                int r = 2 * i + 1;


                if (l < n && a1[l] > a1[largest])
                        largest = l;


                if (r < n && a1[r] > a1[largest])
                        largest = r;


                if (largest != i) {
                        int temp = a1[i];
                        a1[i] = a1[largest];
                        a1[largest] = temp;

                        heapify(a1, n, largest);
                }
        }


void insertionsort(int a1[])
{
        int n = a1.length;
        for(int i=1;i<n;i++)
        {
                int k = a1[i];
                int j = i-1;

                while(j>=0 && a1[j]>k)
                {
                        a1[j+1]=a1[j];
                        j=j-1;
                }
                a1[j+1]=k;
        }
}

        void mergesort(int a1[],int l, int r)
        {
                if(l<r)
                {
                        int m=l+(r-l)/2;
                        mergesort(a1,l,m);
                        mergesort(a1,m+1,r);
                        merge(a1,l,m,r);
                }
        }
 void merge(int a1[], int l, int m, int r)
        {
                int n1= m-l+1;
                int n2= r-m;
                int L[] = new  int[n1];
                int R[] = new  int[n2];
                for(int i=0;i<n1;i++)
                        L[i] = a1[l+i];
                for(int j=0;j<n2;j++)
                        R[j] = a1[m+1+j];

                int i=0,j=0;

                int k=l;
                while(i<n1 && j<n2)
                {
                        if(L[i] <= R[j])
                        {
                                a1[k]=L[i];
                                i++;
                        }
                        else
                        {
                                a1[k]=R[j];
                                j++;
                        }
                        k++;

                }

                while(i<n1)
                {
                        a1[k] = L[i];
                        i++;
                        k++;

                }
                while(j<n2)
                {
                        a1[k] = R[j];
                        j++;
                        k++;

                }

        }


void QuickSort(int[] a1, int low, int high)
        {
                if (low < high)
                {


                        int pi = partition(a1, low, high);

                        QuickSort(a1, low, pi - 1);
                        QuickSort(a1, pi + 1, high);
                }
        }



        static int partition(int[] a1, int low, int high)
        {

                int pivot = a1[high];
                int i = (low - 1);

                for(int j = low; j <= high - 1; j++)
                {


                        if (a1[j] < pivot)
                        {


                                i++;
                                swap(a1, i, j);
                        }
                }
                swap(a1, i + 1, high);
                return (i + 1);
        }
        static void swap(int[] a1, int i, int j)
        {
                int temp = a1[i];
                a1[i] = a1[j];
                a1[j] = temp;
        }

        void display(int a1[])
        {
                int n = a1.length;
                for(int i=0;i<n;i++)
                {
                        System.out.print(a1[i]+" ");
                }
        }


                public static void main(String args[])
        {

                Sorting s1 = new Sorting();
                int a1[]={5,3,8,4,6,9,2,7,11,45,89,67};

                int n = a1.length;
                //s1.bubbleSort(a1);
                //s1.selectionsort(a1);
                //s1.insertionsort(a1);
                //s1.mergesort(a1,0,n-1);
                //s1.QuickSort(a1,0,n-1);
                s1.heapsort(a1);

                s1.display(a1);



        }
}
```

```
class Arrayapp
{
public static void main(String args[])
{
        int[]a1;
        a1=new int[100];
        int j;

        //----------------
        a1[0]= 55;
        a1[1]= 33;
        a1[2]= 22;
        a1[3]= 11;
        a1[4]= 66;
        a1[5]= 88;
        a1[6]= 0;
        a1[7]= 44;
        a1[8]= 99;
        a1[9]= 22;
        int n=10;

        //---------------
        for(int i=0;i<n;i++)
        {
                System.out.print(a1[i]+" ");
        }

        //-------------------
        int key=66;
        for(j=0;j<n;j++)
        {
                if(a1[j] == key)
                        break;
        }
        if(a1[j]==n)
                System.out.println("Not found");
        else
                System.out.println("Found");

        //------------------------
        key =66;
        for(j=0;j<n;j++)
        {
                if(a1[j] == key)
                        break;
        }
        for(int k=j;k<n;k++)
                a1[k]=a1[k+1];
        n--;
```

```
		//---------------
		for(int i=0;i<n;i++)
		{
			System.out.print(a1[i]+" ");
		}

	}

	}
```

```
class Stack
{

        private int max;
        private long[] S1;
        private int top;

        Stack(int s)
        {
                top=-1;
                max=s;
                S1=new long[max];
        }

        public void push(long x)
        {
                S1[++top] = x;
        }

        public long pop()
        {
                return S1[top--];
        }

        public long peek()
        {
                return S1[top];
        }

        public boolean isEmpty()
        {
                return (top == -1);
        }

        public boolean isFull()
        {
                return (top >= (max-1));
        }

        public void display()
        {
                for(int i=0;i<=top;i++)
                {
                        System.out.println(S1[i]);
                }
        }

}

class StackApp1
```

```
{
        public static void reverse(StringBuffer str)
                {
                        int n = str.length();
                        Stack s1 = new Stack(n);

                        int i;
                        for(i=0;i<n;i++)
                        {
                                s1.push(str.charAt(i));
                        }

                        for(i=0;i<n;i++)
                        {
                                char ch = (char)s1.pop();
                                str.setCharAt(i,ch);
                        }


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


                StringBuffer s = new StringBuffer("ABCD");
                reverse(s);
                System.out.println("Reverse of a string = "+s);


        }
}
```

```
class DLL1
{
        Node head;

        static class Node
        {
                int data;
                Node next;
                Node prev;

                Node(int d)
                {
                        data = d;
                        next = null;
                        prev = null;
                }
        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        new_node.prev=null;
        if(head != null )
                head.prev = new_node;
        head = new_node;

}

void display(Node n)
{
        Node p =null;
        System.out.println("Forward Display:");
        while( n != null)
        {
                System.out.print(n.data+ "--> ");
                p=n;
                n=n.next;
        }
        System.out.println("----------------");
        System.out.println("Reverse Display:");
        while( p != null)
        {
                System.out.print(p.data+"<-- ");
                p=p.prev;
        }
}
public static void main(String args[])
{
        DLL1 d1 = new DLL1();

        d1.insert(21);
        d1.insert(11);
        d1.insert(5);
        d1.display(d1.head);

}

}
```

```
static class Node
{
        int data;
        Node next;
        Node prev;

        Node(int d)
        {
                data = d;
                next = null;
                prev = null;
        }
}
```

```
class CQueue
{
        int max=7;//size of circular queue
        int CQ[] = new  int[max];
        int front, rear;

        CQueue()
        {
                front=-1;
                rear=-1;
        }

        boolean isFull()
        {
                //case 1
                if(front==0 && rear == max-1)
                {       return true;}
        //case 2
                if(front == rear+1)
                {       return true;}

                return false;
        }

        boolean isEmpty()
        {
                if(front == -1)
                {       return true;}
                return false;
        }


        void enqueue(int element)
        {
                        if(isFull())
                                System.out.println("Queue is Full !!!");
                        else
                        {
                                //Checking for first queue element
                                if(front == -1)
                                {
                                        front = 0;
                                }
                                rear=(rear+1)%max;
                                CQ[rear]=element;
                                System.out.println(element +" Insertion done !!!");

                        }

        }
```

```
int dequeue()
{
        int element;
        if(isEmpty())
        {
                System.out.println("Queue is Empty !!!");
                return  -1;
        }
        else
        {
                element = CQ[front];
                //remaining one element in queue
                if(front == rear)
                {
                        front=-1;
                        rear=-1;
                }
                else
                {
                        front=(front+1)%max;
                }
                System.out.println("Deleted element = "+ element);
                return element;


        }
}


void display()
{
        for(int i=front;i<= rear;i=(i+1)%max)
        {
                System.out.print(CQ[i]+" ");
        }

        System.out.println(rear+" => Rear pointer");
        System.out.println(front+" => Front pointer");

}


        public static void main(String args[])
{

        CQueue q = new CQueue();
        //q.dequeue();
        //q.display();
```

```
                q.enqueue(10);
                q.enqueue(20);
                q.enqueue(30);
                q.enqueue(40);
                q.enqueue(50);
                q.enqueue(60);
                q.display();

                System.out.println(" ");
                q.dequeue();
                q.display();


        }
}
```

```
class BST
{
        Node root;

        static class Node
        {
                int data;
                Node left;
                Node right;

                Node(int d)
                {
                        data = d;
                        left = null;
                        right = null;
                }
        }
        BST(int d)
        {
                root = new Node(d);
        }

void printInorder(Node n)//Lc,Root, RC
{
        if(n == null)
                return;
        printInorder(n.left);
        System.out.println(n.data);
        printInorder(n.right);


}

void printPreorder(Node n)//Root, LC, RC
{
        if(n == null)
                return;
        System.out.println(n.data);//11 22 44 55 33 66
        printPreorder(n.left);
        printPreorder(n.right);
}


void printPostorder(Node n)//LC,RC,Root
{
        if(n == null)
                return;

        printPostorder(n.left);
        printPostorder(n.right);
        System.out.println(n.data);
}

void preorder()
{
        printPreorder(root);

}

void inorder()
{
        printInorder(root);

}

void postorder()
{
        printPostorder(root);

}

Node insertdata(Node root, int key)
{
                if(root == null)
                {
                        root = new Node(key);
                        return root;
                }
                if(key <= root.data)
                        root.left = insertdata(root.left,key);
                else
                        root.right = insertdata(root.right,key);
                        return root;
}

void insert(int key)
{
        root = insertdata(root, key);
}
//recursive function
Node deletedata(Node root, int key)
{
        if(root == null)
                return root;
        if(key < root.data)
                root.left= deletedata(root.left, key);
        else if(key > root.data)
                root.right= deletedata(root.right, key);
        else
        {
        //case 1, 2
        if(root.left == null)
                return root.right;
        else if(root.right == null)
                return root.left;

        //case 3

        root.data = minvalue(root.right);
        //call inorder method and replace with succesor node.
        root.right = deletedata(root.right, root.data);
        }
        return root;
}
//replacement of node in case of 2 children.
int minvalue(Node root)
{
        int x = root.data;
        while(root.left != null)
        {
                x = root.left.data;
                root =root.left;
        }
        return x;
}

void delete(int key)
{
        root = deletedata(root, key);
}

public static void main(String args[])
{
        BST b1 = new BST(33);
        b1.insert(3);
        b1.insert(43);
        b1.insert(23);
        b1.insert(13);
        b1.insert(73);
        b1.insert(28);
        b1.insert(24);
        b1.insert(30);


        //System.out.println("Preorder:");
        //b1.preorder();

        System.out.println("Inorder:");
        b1.inorder();
        b1.delete(15);

        //b1.delete(23);
        //b1.delete(33);
        System.out.println();
        System.out.println("Inorder:");
        b1.inorder();

        //System.out.println("Postorder:");
        //b1.postorder();
}

}
```

```
class BT
{
        Node root;

        static class Node
        {
                int data;
                Node left;
                Node right;

                Node(int d)
                {
                        data = d;
                        left = null;
                        right = null;
                }
        }
        BT(int d)
        {
                root = new Node(d);
        }

void printInorder(Node n)//Lc,Root, RC
{
        if(n == null)
                return;
        printInorder(n.left);
        System.out.println(n.data);
        printInorder(n.right);


}

void printPreorder(Node n)//Root, LC, RC
{
        if(n == null)
                return;
        System.out.println(n.data);
        printPreorder(n.left);
        printPreorder(n.right);
}

void printPostorder(Node n)//LC,RC,Root
{
        if(n == null)
                return;

        printPostorder(n.left);
        printPostorder(n.right);
        System.out.println(n.data);
}

void preorder()
{
        printPreorder(root);

}

void inorder()
{
        printInorder(root);

}

void postorder()
{
        printPostorder(root);

}
public static void main(String args[])
{
        BT b1 = new BT(11);
        //b1.root = new Node(11);
        b1.root.left = new Node(22);
        b1.root.right = new Node(33);
        b1.root.left.left = new Node(44);
        b1.root.left.right = new Node(55);
        b1.root.right.right = new Node(66);

        System.out.println("Preorder:");
        b1.preorder();

        System.out.println("Inorder:");
        b1.inorder();

        System.out.println("Postorder:");
        b1.postorder();
}

}
```

```
class BS
{
        public static int bsearch(int a1[],int x,int l, int r)
        {
                if(r>=l)
                {
                                int mid=(l+(r-l))/2;
                                if(a1[mid] == x)
                                                return mid;
                                if(a1[mid] > x)
                                        return bsearch(a1,x,l,mid-1);
                                return bsearch(a1,x,mid+1,r);

                }
                return -1;

        }
public static void main(String args[])
{
        int a1[]={2,4,7,9,15};
        int x=7;
        int n = a1.length;

        int result = bsearch(a1,x,0,n-1);

        if(result == -1)
                System.out.println("Element Not Found");
        else
                System.out.println("Element Found");

}

}
```

```
class DLL4
{
        Node head;

        static class Node
        {
                int data;
                Node next;
                Node prev;

                Node(int d)
                {
                        data = d;
                        next = null;
                        prev = null;
                }
        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        new_node.prev=null;
        if(head != null )
                head.prev = new_node;
        head = new_node;

}

void display(Node n)
{
        Node p =null;
        System.out.println("Forward Display:");
        while( n != null)
        {
                System.out.print(n.data+ "--> ");
                p=n;
                n=n.next;
        }
        System.out.println("----------------");
        System.out.println("Reverse Display:");
        while( p != null)
        {
                System.out.print(p.data+"<-- ");
                p=p.prev;
        }
}

void insertAfter(Node prev,int new_data)
```

```
{
        if(prev == null)
                {return;}
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;
        new_node.prev = prev;
        new_node.next.prev = new_node;

}

void append(int new_data)
{
        Node new_node = new Node(new_data);
        Node n = head;
        new_node.next = null;
        if(head == null)
        {
                new_node.prev = null;
                head = new_node;
                return;

        }
        while(n.next != null)
        {
                n=n.next;
        }
        n.next = new_node;
        new_node.prev = n;


}
void delete(Node n)
{
        if(head == null)
                return;
        if(head == n)
        {
                head = n.next;
        }
        //It is not a last node
        if(n.next != null)
        {
                n.next.prev = n.prev;
        }
        if(n.prev != null)
        {
                n.prev.next = n.next;
        }
        return;
}
public static void main(String args[])
{
        DLL4 d1 = new DLL4();

        d1.append(90);

        d1.insert(21);
        d1.insert(11);
        d1.insert(5);
        d1.display(d1.head);
        System.out.println();
        d1.delete(d1.head.next);
        d1.display(d1.head);
        System.out.println();

        /*d1.insertAfter(d1.head, 45);
        d1.insertAfter(d1.head, 56);
        d1.insertAfter(d1.head, 75);
        d1.display(d1.head);
        System.out.println();


        d1.append(78);
        d1.display(d1.head);
        System.out.println();
        */


}

}
```

```
class DLL2
{
        Node head;

        static class Node
        {
                int data;
                Node next;
                Node prev;

                Node(int d)
                {
                        data = d;
                        next = null;
                        prev = null;
                }
        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        new_node.prev=null;
        if(head != null )
                head.prev = new_node;
        head = new_node;

}

void display(Node n)
{
        Node p =null;
        System.out.println("Forward Display:");
        while( n != null)
        {
                System.out.print(n.data+ "--> ");
                p=n;
                n=n.next;
        }
        System.out.println("----------------");
        System.out.println("Reverse Display:");
        while( p != null)
        {
                System.out.print(p.data+"<-- ");
                p=p.prev;
        }
}

void insertAfter(Node prev,int new_data)
```

```
{
        if(prev == null)
                {return;}
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;
        new_node.prev = prev;
        new_node.next.prev = new_node;

}

void append(int new_data)
{
        Node new_node = new Node(new_data);
        Node n = head;
        new_node.next = null;
        if(head == null)
        {
                new_node.prev = null;
                head = new_node;
                return;

        }
        while(n.next != null)
        {
                n=n.next;
        }
        n.next = new_node;
        new_node.prev = n;


}

void deleteNode(Node del)
                {

                        // Base case
                        if (head == null || del == null) {
                                return;
                        }

                        if (head == del) {
                                head = del.next;
                        }
                        if (del.next != null) {
                                del.next.prev = del.prev;
                        }
                        if (del.prev != null) {
                                del.prev.next = del.next;
                        }
                        return;
                }


public static void main(String args[])
{
        DLL2 d1 = new DLL2();

        d1.append(90);

        d1.insert(21);
        d1.insert(11);
        d1.insert(5);
        d1.display(d1.head);
        System.out.println();

        d1.insertAfter(d1.head, 45);
        d1.insertAfter(d1.head, 56);
        d1.insertAfter(d1.head, 75);
        d1.display(d1.head);
        System.out.println();


        d1.append(78);
        d1.display(d1.head);
        System.out.println();


}

}
```

```
class DLL2
{
        Node head;

        static class Node
        {
                int data;
                Node next;
                Node prev;

                Node(int d)
                {
                        data = d;
                        next = null;
                        prev = null;
                }
        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        new_node.prev=null;
        if(head != null )
                head.prev = new_node;
        head = new_node;

}

void display(Node n)
{
        Node p =null;
        System.out.println("Forward Display:");
        while( n != null)
        {
                System.out.print(n.data+ "--> ");
                p=n;
                n=n.next;
        }
        System.out.println("----------------");
        System.out.println("Reverse Display:");
        while( p != null)
        {
                System.out.print(p.data+"<-- ");
                p=p.prev;
        }
}

void insertAfter(Node prev,int new_data)
```

```
{
        if(prev == null)
                {return;}
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;
        new_node.prev = prev;
        new_node.next.prev = new_node;

}

void append(int new_data)
{
        Node new_node = new Node(new_data);
        Node n = head;
        new_node.next = null;
        if(head == null)
        {
                new_node.prev = null;
                head = new_node;
                return;

        }
        while(n.next != null)
        {
                n=n.next;
        }
        n.next = new_node;
        new_node.prev = n;


}
public static void main(String args[])
{
        DLL2 d1 = new DLL2();

        d1.append(90);

        d1.insert(21);
        d1.insert(11);
        d1.insert(5);
        d1.display(d1.head);
        System.out.println();

        d1.insertAfter(d1.head, 45);
        d1.insertAfter(d1.head, 56);
        d1.insertAfter(d1.head, 75);
        d1.display(d1.head);
        System.out.println();
        d1.append(78);
        d1.display(d1.head);
        System.out.println();


}

}
```

```
class LinkedList1
{
        Node head;
        static class Node
        {
                int data;
                Node next;
                Node(int d)
                {
                        data=d;
                        next=null;
                }
        }



public static void main(String args[])
{
        LinkedList l1 = new LinkedList();
        l1.head = new Node(10);
        Node second = new Node(20);
        Node third = new Node(30);

        l1.head.next = second;
        second.next = third;

}

}
```

```
class LinkedList
{
        static class Node
        {
                int data;
                Node next;
                Node(int d)
                {
                        data=d;
                        next=null;
                }
        }

        void display()
        {
                Node n=head;
                while(n != null)
                {
                        System.out.println(n.data);
                        n=n.next;
                }


        }
}
```

```
public class heapsort {
        public void sort(int a1[])
        {
                int n = a1.length;


                for (int i = n / 2 - 1; i >= 0; i--)
                        heapify(a1, n, i);


                for (int i = n - 1; i > 0; i--) {

                        int temp = a1[0];
                        a1[0] = a1[i];
                        a1[i] = temp;


                        heapify(a1, i, 0);
                }
        }


        void heapify(int a1[], int n, int i)
        {
                int largest = i;
                int l = 2 * i ;
                int r = 2 * i + 1;


                if (l < n && a1[l] > a1[largest])
                        largest = l;


                if (r < n && a1[r] > a1[largest])
                        largest = r;


                if (largest != i) {
                        int temp = a1[i];
                        a1[i] = a1[largest];
                        a1[largest] = temp;

                        heapify(a1, n, largest);
                }
        }


        static void display(int a1[])
        {
                int n = a1.length;
                for (int i = 0; i < n; ++i)
                        System.out.print(a1[i] + " ");
                System.out.println();
        }


        public static void main(String args[])
        {
                int a1[] = { 22, 11, 33, 55, 66, 77 ,99,5 };
                int n = a1.length;

                heapsort ob = new heapsort();
                ob.sort(a1);

                System.out.println("Sorted array is");
                display(a1);
        }
}
```

Insertion Sort:

```
void insertionsort(int a1[])
{
        int n=a1.length;
        for(int i=1;i<n;i++)
        {
                int key = a1[i];
                int j= i-1;//0

                while(j>=0 && a1[j] > key)
                {
                                a1[j+1] = a1[j];
                                j=j-1;
                }
                a1[j+1] = key;
        }
}
```

```
class QueueLL1 {
        QNode front, rear;

        static class QNode {
                int key;
                QNode next;

                // constructor to create a new linked list node
                public QNode(int key)
                {
                        this.key = key;
                        this.next = null;
                }
        }

        public QueueLL1()
        {
                this.front = this.rear = null;
        }

        // Method to add an key to the queue.
        void enqueue(int key)
        {

                // Create a new LL node
                QNode temp = new QNode(key);

                // If queue is empty, then new node is front and rear both
                if (this.rear == null) {
                        this.front = this.rear = temp;
                        return;
                }

                // Add the new node at the end of queue and change rear
                this.rear.next = temp;
                this.rear = temp;
        }

        // Method to remove an key from queue.
        void dequeue()
        {
                // If queue is empty, return NULL.
                if (this.front == null)
                        return;

                // Store previous front and move front one node ahead
                QNode temp = this.front;
                this.front = this.front.next;

                // If front becomes NULL, then change rear also as NULL
                if (this.front == null)
                        this.rear = null;
        }


        public static void main(String[] args)
        {
                QueueLL1 q = new QueueLL1();
                q.enqueue(10);
                q.enqueue(20);
                q.dequeue();
                q.dequeue();
                q.enqueue(30);
                q.enqueue(40);
                q.enqueue(50);
                q.dequeue();
                System.out.println("Queue Front : " + q.front.key);
                System.out.println("Queue Rear : " + q.rear.key);
        }
}
```

```
class Queue
{
        int max=5;
        int Q[] = new  int[max];
        int front, rear;

        Queue()
        {
                front=-1;
                rear=-1;
        }

        boolean isFull()
        {
                if(front==0 && rear == max-1)
                {       return true;}
                return false;
        }

        boolean isEmpty()
        {
                if(front == -1)
                {       return true;}
                return false;
        }


        void enqueue(int element)
        {
                        if(isFull())
                                System.out.println("Queue is Full !!!");
                        else
                        {
                                //Checking for first queue element
                                if(front == -1)
                                {
                                        front = 0;
                                }
                                rear++;
                                Q[rear]=element;
                                System.out.println("Insertion done !!!");

                        }

        }


        int dequeue()
        {
                int element;
                if(isEmpty())
                {
                        System.out.println("Queue is Empty !!!");
                        return  -1;
                }
                else
                {
                        element = Q[front];
                        //remaining one element in queue
                        if(front > rear)
                        {
                                front=-1;
                                rear=-1;
                        }
                        else
                        {
                                front++;
                        }
                        System.out.println("Deleted element = "+ element);
                        return element;


                }
        }


        void display()
        {
                for(int i=front;i<=rear;i++)
                {
                        System.out.print(Q[i]+" ");
                }

                System.out.println(rear+" => Rear pointer");
                System.out.println(front+" => Front pointer");

        }


                public static void main(String args[])
        {

                Queue q = new Queue();
                q.dequeue();
                q.display();

                q.enqueue(10);
                q.enqueue(20);
                q.enqueue(30);
                q.enqueue(40);
                q.enqueue(50);
                //q.enqueue(60);
                q.display();

                System.out.println(" ");
                q.dequeue();
                q.display();


        }
}
```

```
class LS
{
        public static int lsearch(int a1[],int x)
        {
                int n=a1.length;
                for(int i=0;i<n;i++)
                {
                        if(a1[i] == x)
                                return i;
                }
                return -1;
        }
public static void main(String args[])
{
        int a1[]={2,4,7,9,5};
        int x=5;

        int result = lsearch(a1,x);

        if(result == -1)
                System.out.println("Element Not Found");
        else
                System.out.println("Element Found");

}

}
```

```
public class Recursion9 {



                static void display(String str, String ans)
                {

                        if (str.length() == 0) {
                                System.out.print(ans + " ");
                                return;
                        }

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

                                char ch = str.charAt(i);
                                String res = str.substring(0, i) + str.substring(i +
1);

                                display(res, ans + ch);
                        }
                }


                public static void main(String[] args)
                {
                        String s = "abc";
                        display(s, "");
                }


}
```

```
class LinkedList4
{
        Node head;

        static class Node
        {
        int data;
        Node next;
        Node(int d)
        {
                data=d;
                next=null;
        }
        }

        void display()
        {
                Node n=head;
                while(n != null)
                {
                        System.out.print(n.data+"--->");
                        n=n.next;
                }


        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        head= new_node;


}


void insertAfter(Node prev,int new_data)
{
        if(prev == null)
        {
                System.out.println("Insertion is not possible.");
                return;
        }
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;

}
```

```
void append(int new_data)
{
        Node new_node = new Node(new_data);
        if(head == null)
        {
                head=new_node;
                return;
        }
        new_node.next = null;
        Node n = head;
        while(n.next != null)
        {
                n = n.next;
        }
        n.next = new_node;
        return;

}

void delete(int key)
{
        Node temp = head, prev =null;
        if(temp.data == key && temp !=null)
        {
                head = temp.next;
                return;
        }
        while(temp !=null && temp.data != key)
        {
                prev = temp;
                temp = temp.next;
        }
        if(temp==null)
                {return;}
        prev.next = temp.next;
}

public static void main(String args[])
{
        LinkedList3 l1 = new LinkedList3();


        l1.append(33);



        l1.insert(10);
        l1.insert(20);
        l1.insert(30);
        l1.insert(40);
        l1.display();
        l1.insertAfter(l1.head,50);
        System.out.println();
        l1.display();
        l1.insertAfter(l1.head.next.next,60);
        l1.append(44);
        l1.append(55);
        System.out.println();
        l1.display();
        l1.delete(50);//In between element
        System.out.println();
        l1.display();
        l1.delete(40);//First elemet
        System.out.println();
        l1.display();
        l1.delete(33);//Last element
        System.out.println();
        l1.display();

    }

    }
```

```
}



class LinkedList3
{
        Node head;

        static class Node
        {
        int data;
        Node next;
        Node(int d)
        {
                data=d;
                next=null;
        }
        }

        void display()
        {
                Node n=head;
                while(n != null)
                {
                        System.out.print(n.data+"--->");
                        n=n.next;
                }


        }

        void insert(int new_data)
{
        Node new_node = new Node(new_data);
        new_node.next = head;
        head= new_node;


}


void insertAfter(Node prev,int new_data)
{
        if(prev == null)
        {
                System.out.println("Insertion is not possible.");
                return;
        }
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;

}
```

```
}



void append(int new_data)
{
        Node new_node = new Node(new_data);
        if(head == null)
        {
                head=new_node;
                return;
        }
        new_node.next = null;
        Node n = head;
        while(n.next != null)
        {
                n = n.next;
        }
        n.next = new_node;
        return;

}



public static void main(String args[])
{
        LinkedList3 l1 = new LinkedList3();


        l1.append(33);



        l1.insert(10);
        l1.insert(20);
        l1.insert(30);
        l1.insert(40);
        l1.display();
        l1.insertAfter(l1.head,50);
        System.out.println();
        l1.display();
        l1.insertAfter(l1.head.next.next,60);
        l1.append(44);
        l1.append(55);
        System.out.println();
        l1.display();


}

}
```

```
class LinkedList2
{
        Node head;

        static class Node
        {
        int data;
        Node next;
        Node(int d)
        {
                data=d;
                next=null;
        }
        }

        void display()
        {
                Node n=head;
                while(n != null)
                {
                        System.out.println(n.data);
                        n=n.next;
                }


        }

public static void main(String args[])
{
        LinkedList l1 = new LinkedList();
        l1.head = new Node(10);
        Node second = new Node(20);
        Node third = new Node(30);

        l1.head.next = second;
        second.next = third;

        l1.display();

}

}
```

```
//Finite Loop
class Recursion4
{

	static int fact(int n)//4
	{
		if(n<=1)
				return 1;
		else
			return n*fact(n-1);
	}

public static void main(String args[])
{
	System.out.println(fact(5));
}

}
```

```
//Finite Loop
class Recursion3
{

        static int display(int n)//4
        {
                if(n==4)
                        return n;
                else
                        return 2*display(n+1);//3


        }

public static void main(String args[])
{
        System.out.println(display(2));
}

}
```

```
//Finite Loop
class Recursion2
{       static int i=0;

        static void display()
        {
                ++i;
                if(i<=5)
                {
                System.out.println("Hi Everyone !!!");
                display();
                }
        }

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

}
```

```
class BS
{
        public static int bsearch(int a1[],int x,int l, int r)
        {
                if(r>=l)
                {
                                int mid=(l+(r-l))/2;
                                if(a1[mid] == x)
                                                return mid;
                                if(a1[mid] > x)
                                        return bsearch(a1,x,l,mid-1);
                                return bsearch(a1,x,mid+1,r);

                }
                return -1;

        }
public static void main(String args[])
{
        int a1[]={2,4,7,9,15};
        int x=7;
        int n = a1.length;

        int result = bsearch(a1,x,0,n-1);

        if(result == -1)
                System.out.println("Element Not Found");
        else
                System.out.println("Element Found");

}

}
```

```
import static java.lang.System.exit;

        class StackLL {

                // A linked list node
                private class Node {

                        int data;
                        Node link;
                }

                Node top;

                StackLL()
                {
                        this.top = null;
                }


                public void push(int x)
                {

                        Node temp = new Node();


                        if (temp == null) {
                                System.out.print("\nHeap Overflow");
                                return;
                        }


                        temp.data = x;


                        temp.link = top;

                        top = temp;
                }


                public boolean isEmpty()
                {
                        return top == null;
                }


                public int peek()
                {
```

```
        if (!isEmpty()) {
                return top.data;
        }
        else {
                System.out.println("Stack is empty");
                return -1;
        }
}


public void pop()
{

        if (top == null) {
                System.out.print("\nStack Underflow");
                return;
        }

        top = (top).link;
}

public void display()
{

        if (top == null) {
                System.out.printf("\nStack Underflow");
                exit(1);
        }
        else {
                Node temp = top;
                while (temp != null) {


                        System.out.printf("%d->", temp.data);


                        temp = temp.link;
                }
        }
}



public static void main(String[] args)
{

        StackLL s1 = new StackLL();

        s1.push(11);
        s1.push(22);
```

```
                s1.push(33);
                s1.push(44);


                s1.display();


                System.out.printf("\nTop element is %d\n", s1.peek());


                s1.pop();
                s1.pop();


                s1.display();


                System.out.printf("\nTop element is %d\n", s1.peek());
        }



}
```

```
//Finite Loop
class Recursion5
{
//0 1 1 2  3  5 8....
        static int fibonacci(int n)//4
        {
                if(n<=0)
                                return 1;
//fib(n)=fib(n-1)+fib(n-2);
                return fibonacci(n-1) + fibonacci(n-2);
        }

public static void main(String args[])
{

        System.out.println("Enter series for 10 numbers:");
        for(int i=1;i<=10;i++)
        {
                System.out.println(fibonacci(i)+ " ");

        }
}

}
```

```
import java.util.*;
class Stack{
        private int[] arr;
        private int size;
        private int top;

        Stack(int size){
                this.size=size;
                arr = new int[size];
                top=-1;
        }
        boolean isEmpty(){
                if(top == -1)
                        return true;
                else
                        return false;
        }

        boolean isFull(){
                if(top == size - 1){
                        return true;
                }else{
                        return false;
                }
        }

        void push(int x){
                if(isFull()){
                        System.out.println("Stack is full");
                }else{
                        top++;
                        arr[top]=x;
                        System.out.println(x+" pushed");
                }
        }

        void pop(){
                if(isEmpty()){
                        System.out.println("Stack is empty");
                }else{
                        int x = arr[top];
                        System.out.println(x+" poped");
                        top--;
                }
        }

        void display(){
                if(isEmpty()){
                        System.out.println("Stack is empty");
                }else{
```

```
                for(int i=0;i<=top;i++){
                        System.out.print(arr[i]+" ");
                }
                System.out.println();
        }
    }
    public static void main(String[] args){
        System.out.print("Enter the size of stack:");
        Scanner sc = new Scanner(System.in);
        int size = sc.nextInt();
        Stack s = new Stack(size);
        while(true){
                System.out.println("1. Push\n2. Pop\n3. Display\n4. Exit");
                System.out.println("Enter your choice:");
                int choice = sc.nextInt();
                switch(choice){
                        case 1:
                                System.out.print("Enter the no:");
                                int num = sc.nextInt();
                                s.push(num);
                                break;
                        case 2:
                                s.pop();
                                break;
                        case 3:
                                s.display();
                                break;
                        case 4:
                                System.exit(0);
                                break;
                        default:
                                System.out.println("Enter valid choice");
                                break;
                }
        }
    }
}
```

```
import java.util.*;
class Queue{
        private int[] arr;
        private int size;
        private int front;
        private int rear;

        Queue(int size){
                arr = new int[size];
                this.size = size;
                front = rear = -1;
        }

        boolean isEmpty(){
                if((front == -1 && rear == -1) || front> rear){
                        return true;
                }else{
                        return false;
                }
        }

        boolean isFull(){
                if(rear == size-1)
                        return true;
                else
                        return false;
        }

        public void enqueue(int x){
                if(isFull())
                        System.out.println("Queue is full");
                else
                {
                        if(front == -1 && rear == -1)
                                front=0;
                        rear++;
                        arr[rear]=x;
                        System.out.println(x+" enqueued");
                }
        }

        void dequeue(){
                if(isEmpty()){
                        System.out.println("Queue is empty");
                }else{
                        int x = arr[front];
                        front++;
                        System.out.println(x+" dequeued");
                }
        }
```

```
	void display(){
		if(isEmpty()){
			System.out.println("Queue is empty");
		}else{
			for(int i=front;i<=rear;i++){
				System.out.print(arr[i]+" ");
			}
			System.out.println();
		}
	}
	public static void main(String[] args){
		System.out.print("Enter the size of queue:");
		Scanner sc = new Scanner(System.in);
		int size = sc.nextInt();
		Queue s = new Queue(size);
		while(true){
			System.out.println("1. Enqueue\n2. Dequeue\n3. Display\n4. 
Exit");
			System.out.println("Enter your choice:");
			int choice = sc.nextInt();
			switch(choice){
				case 1:
					System.out.print("Enter the no:");
					int num = sc.nextInt();
					s.enqueue(num);
					break;
				case 2:
					s.dequeue();
					break;
				case 3:
					s.display();
					break;
				case 4:
					System.exit(0);
					break;
				default:
					System.out.println("Enter valid choice");
					break;
			}
		}
	}
}
```