```
//Stack
import java.util.*;
class StackClass {
    int s[],tos,MaxSize;

    StackClass(int n)
    {
        MaxSize=n;
        tos=-1;
        s=new int[MaxSize];

    }

    void push(int element)
    {
        tos++;
        s[tos]=element;

    }

    int pop()
    {
        int t=s[tos];
        tos--;
        return(t);
    }

    int elementattos()
    {
        return(s[tos]);
    }

    void printall()
    {
        for(int i =tos;i>=0;i--)
            System.out.println(s[i]);

    }


    boolean is_empty()
    {
        if(tos==-1)
            return(true);
        else
            return(false);
    }
     boolean is_full()
    {
        if(tos==MaxSize-1)
```


```
            return(true);
        else
            return(false);
    }
}

class Stack
{

    public static void main(String args[])
    {
        StackClass s=null;
        int size,choice,e;
        Scanner in=new Scanner(System.in);
        System.out.println("eneter size of stack");
        size=in.nextInt();
        s=new StackClass(size);
        do
        {
            System.out.println("1. Push\n2. Pop\n3. Element at top\n4. Print Stack
Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                    if(!s.is_full())
                    {
                    System.out.println("enter element:");
                    e=in.nextInt();
                    s.push(e);
                    }
                    else
                    {
                     System.out.println("Sorry Stack Full");
                    }
                     break;

                case 2:
                    if(!s.is_empty())
                    {
                    System.out.println("Element poped is:"+s.pop());

                    }
                    else
                    {
                     System.out.println("Sorry Stack Empty");
                    }
                    break;

                case 3:
```


```
                if(!s.is_empty())
                {
                System.out.println("Element At Top is:"+s.elementattos());

                }
                else
                {
                 System.out.println("Sorry Stack Empty");
                }
                break;

            case 4:
                if(!s.is_empty())
                {
                System.out.println("Element OnStack Are:\n");
                s.printall();

                }
                else
                {
                 System.out.println("Sorry Stack Empty");
                }
                break;

            case 0:
                System.out.println("Exiting program");
                 break;

            default:
                System.out.println("Wrong input");
                 break;


          }

        }while(choice!=0);
    }
}
```


//TOWER OF HANOI

```
public class TowerOfHanoi {
    public static void main(String args[])
    {
        sol(3,'a','b','c');
    }

    static void sol(int disk,char Source,char Aux,char Dest)
    {
        if(disk==0)
            return;
        else
        {
            sol(disk-1,Source,Dest,Aux);
            System.out.println(" Move disk "+disk+" from "+Source+" to "+Dest);
            sol(disk-1,Aux,Source,Dest);
        }

    }
}
```


//INFIX TO POSTFIX

```
import java .util.*;
class Stackchar
{
  int MaxSize, tos;
  char s[];
  Stackchar(int size)
  {
    MaxSize = size;
    tos = -1;
    s= new char[MaxSize];
  }

  void push(char x)
  {
    tos++;
    s[tos]= x;
  }

  char pop()
  {
    char t = s[tos];
    tos--;
    return t;
  }

  boolean is_full()
  {
    if(tos==MaxSize-1)
    return true;
   else
     return false;
  }

   boolean is_empty()
  {
    if(tos==-1)
    return true;
   else
     return false;
  }

  char at_tos()
  {
    if (tos==-1)
          return(' ');
    else
        return(s[tos]);
  }
```


```
  void print()
  {
    for(int i= tos;i>=0;i--)
    System.out.println(s[i]);
  }

}

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

    Scanner in = new Scanner(System.in);

    Stackchar s= null;

    String post="", infix;

    System.out.println("Enter Infix:");
    infix= in.nextLine();

    int size = infix.length();

    s = new Stackchar(size);

    for(int i=0; i<size ; i++)
    {
      char c = infix.charAt(i);

      switch(c)
      {
        case'(': s.push(c); break;

        case')': while(s.at_tos()!='(')
                 {
                    post = post + s.pop();
                 }
                 char g = s.pop();
                 break;

        case'+': case'-': case'*': case'/':

                 while(precedence(c)<=precedence(s.at_tos()) && !s.is_empty())
                 {
                   post = post+s.pop();
                 }
                   s.push(c);
                 break;
```


```
        default: post= post+c;
      }
    }

    while(!s.is_empty())
    {
      post = post + s.pop();
    }

    System.out.println("Postfix= "+post);
  }

  public static int precedence(char ch)
  {
    if(ch=='*' || ch=='/')
     return(2);

    else if(ch=='+' || ch=='-')
     return(1);

    else
     return(0);
  }
}
```


//INFIX TO PREFIX

```
import java .util.*;
class Stackchar
{
  int MaxSize, tos;
  char s[];
  Stackchar(int size)
  {
    MaxSize = size;
    tos = -1;
    s= new char[MaxSize];
  }

  void push(char x)
  {
    tos++;
    s[tos]= x;
  }

  char pop()
  {
    char t = s[tos];
    tos--;
    return t;
  }

  boolean is_full()
  {
    if(tos==MaxSize-1)
    return true;
   else
     return false;
  }

   boolean is_empty()
  {
    if(tos==-1)
    return true;
   else
     return false;
  }

  char at_tos()
  {
    if (tos==-1)
          return(' ');
    else
        return(s[tos]);
  }
```


```
  void print()
  {
    for(int i= tos;i>=0;i--)
    System.out.println(s[i]);
  }

}

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

    Scanner in = new Scanner(System.in);

    Stackchar s= null;

    String pre="", infix;

    System.out.println("Enter Infix:");
    infix= in.nextLine();

    int size = infix.length();

    s = new Stackchar(size);

    for(int i=size-1; i>=0 ; i--)
    {
      char c = infix.charAt(i);

      switch(c)
      {
        case')': s.push(c); break;

        case'(': while(s.at_tos()!=')')
                 {
                    pre = pre + s.pop();
                 }
                 char g = s.pop();
                 break;

        case'+': case'-': case'*': case'/':

                 while(precedence(c)<precedence(s.at_tos()) && !s.is_empty())
                 {
                   pre = pre+s.pop();
                 }
                   s.push(c);
                 break;
```


```
        default: pre= pre+c;
      }
    }

    while(!s.is_empty())
    {
      pre = pre + s.pop();
    }

    System.out.println("Prefix= \n");
    for(int i=pre.length()-1;i>=0;i--)
        System.out.print(pre.charAt(i));

  }

  public static int precedence(char ch)
  {
    if(ch=='*' || ch=='/')
     return(2);

    else if(ch=='+' || ch=='-')
     return(1);

    else
     return(0);
  }
}
```


//Queue

```
import java.util.*;
class QueueClass {
    int q[],front,rear,MaxSize;

    QueueClass(int n)
    {
        MaxSize=n;
        front=0;
        rear=-1;
        q=new int[MaxSize];

    }

    void enqueue(int element)
    {
       rear++;
       q[rear]=element;

    }

    int dequeue()
    {
       int t=q[front];
       front++;
       return(t);
    }

    int elementatfront()
    {
        return(q[front]);
    }
    int elementatrear()
    {
        return(q[rear]);
    }

    void printall()
    {
        for(int i =front;i<=rear;i++)
            System.out.print(q[i]+"-");

    }

    boolean is_empty()
    {
        if(front>rear)
            return(true);
        else
            return(false);
```


```
    }
     boolean is_full()
    {
        if(rear==MaxSize-1)
            return(true);
        else
            return(false);
    }
}

class queue
{


    public static void main(String args[])
    {
        QueueClass q=null;
        int size,choice,e;
        Scanner in=new Scanner(System.in);
        System.out.println("eneter size of queue");
        size=in.nextInt();
        q=new QueueClass(size);
        do
        {
            System.out.println("\n1. Enqueue\n2. Dequeue\n3. Element at Front\n4.
Element at Rear\n5. Print Queue Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                    if(!q.is_full())
                    {
                    System.out.println("enter element:");
                    e=in.nextInt();
                    q.enqueue(e);
                    }
                    else
                    {
                     System.out.println("Sorry Queue Is Full");
                    }
                     break;

                case 2:
                    if(!q.is_empty())
                    {
                    System.out.println("Element poped is:"+q.dequeue());

                    }
                    else
                    {
```


```
            System.out.println("Sorry Queue Empty");
           }
           break;

     case 3:
           if(!q.is_empty())
           {
           System.out.println("Element At Front is:"+q.elementatfront());

           }
           else
           {
            System.out.println("Sorry Queue Empty");
           }
           break;

     case 4:
           if(!q.is_empty())
           {
           System.out.println("Element At Rear is:"+q.elementatrear());

           }
           else
           {
            System.out.println("Sorry Queue Empty");
           }
           break;

     case 5:
           if(!q.is_empty())
           {
           System.out.println("Element OnQueue Are:\n");
           q.printall();

           }
           else
           {
            System.out.println("Sorry Queue Empty");
           }
           break;

     case 0:
           System.out.println("Exiting program");
            break;

     default:
           System.out.println("Wrong input");
            break;
  }
}while(choice!=0);
```


```
    }
}
```

//Circular queue

```
import java.util.*;
class QueueClass {
    int q[],front,rear,MaxSize,count;

    QueueClass(int n)
    {
        MaxSize=n;
        front=0;
        rear=-1;
        count=0;
        q=new int[MaxSize];

    }

    void enqueue(int element)
    {
       rear=(rear+1)%MaxSize;
       q[rear]=element;
       count++;

    }

    int dequeue()
    {
       int t=q[front];
       rear=(rear+1)%MaxSize;
       count--;
       return(t);
    }

    int elementatfront()
    {
        return(q[front]);
    }
    int elementatrear()
    {
        return(q[rear]);
    }

    void printall()
    {
       int i=front;
       int c=0;
        for(; c<=count ;c++,i=(i+1)%MaxSize)
            System.out.print(q[i]+"-");
```


```
    }

    boolean is_empty()
    {
        if(count==0)
            return(true);
        else
            return(false);
    }
     boolean is_full()
    {
        if(count==MaxSize-1)
            return(true);
        else
            return(false);
    }
}


public class Circularqueue
{

   public static void main(String args[])
    {
        QueueClass q=null;
        int size,choice,e;
        Scanner in=new Scanner(System.in);
        System.out.println("eneter size of queue");
        size=in.nextInt();
        q=new QueueClass(size);
        do
        {
            System.out.println("\n1. Enqueue\n2. Dequeue\n3. Element at Front\n4.
Element at Rear\n5. Print Queue Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                    if(!q.is_full())
                    {
                    System.out.println("enter element:");
                    e=in.nextInt();
                    q.enqueue(e);
                    }
                    else
                    {
                     System.out.println("Sorry Queue Is Full");
                    }
                     break;
```


```
 case 2:
       if(!q.is_empty())
       {
       System.out.println("Element poped is:"+q.dequeue());

       }
       else
       {
        System.out.println("Sorry Queue Empty");
       }
       break;

 case 3:
       if(!q.is_empty())
       {
       System.out.println("Element At Front is:"+q.elementatfront());
       }
       else
       {
        System.out.println("Sorry Queue Empty");
       }
       break;

case 4:
       if(!q.is_empty())
       {
       System.out.println("Element At Rear is:"+q.elementatrear());
       }
       else
       {
        System.out.println("Sorry Queue Empty");
       }
       break;

case 5:
       if(!q.is_empty())
       {
       System.out.println("Element OnQueue Are:\n");
       q.printall();
       }
       else
       {
        System.out.println("Sorry Queue Empty");
       }
       break;
case 0:
       System.out.println("Exiting program");
        break;

default:
```


```
                System.out.println("Wrong input");
                 break;
            }
          }while(choice!=0);
      }
}
```


```
//LinkedList
import java.util.*;
class node
{
    int data;
    node next;
    node(int e)
    {
        data=e;
        next=null;
    }
}
class LinkedListClass
{
    node root,newnode,t,t2;
    LinkedListClass()
    {
        root=null;
    }
void insert_left(node n)
{
    if(root==null)
    {
        root=n;
    }
    else
    {
        n.next=root;
        root=n;
    }
}
node delete_left()
{
    t=null;
    if(root!=null)
    {
      t=root;
      root=root.next;

    }
  return(t);
}
void insert_right(node n)
{
    if(root==null)
    {
        root=n;
    }
    else
    {
```


```
        t=root;
        while(t.next!=null)
            t=t.next;
        t.next=n;

    }
}
node delete_right()
{
    t=null;
    if(root!=null)
    {
      t=root;
       while(t.next!=null)
       {
           t2=t;
           t=t.next;
       }
      t2.next=null;
    }
  return(t);
}

void printall()
{
    if(root!=null)
    {
        t=root;
        while(t!=null)
        {
            System.out.print("|" + t.data + "|->|");
            t=t.next;
        }

    }
}
boolean is_empty()
{
    if(root==null)
        return true;
    else
        return false;

}
}

public class LinkedList {
     public static void main(String args[])
    {
        int choice,e;node n;
```


```
        Scanner in=new Scanner(System.in);
         LinkedListClass l=new LinkedListClass();
        do
        {
            System.out.println("1. Insert to Left\n2. Insert to right\n3. Delete Left\n4.
Delete Right\n5. Print Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                      System.out.println("enter element:");
                      e=in.nextInt();
                      n=new node(e);
                      l.insert_left(n);
                      break;

                case 2:
                      System.out.println("enter element:");
                      e=in.nextInt();
                      n=new node(e);
                      l.insert_right(n);
                      break;

                case 3:
                     if(!l.is_empty())
                        System.out.println("element removed is "+(l.delete_left()).data);
                     else
                        System.out.println("Sorry Linkelist is empty");
                      break;

               case 4:
                      if(!l.is_empty())
                        System.out.println("element removed is "+(l.delete_right()).data);
                     else
                        System.out.println("Sorry Linkelist is empty");
                      break;

               case 5:
                      if(!l.is_empty())
                        l.printall();
                      else
                        System.out.println("Sorry Linkelist is empty");
                      break;

               case 0:
                      System.out.println("Exiting program");
                       break;

               default:
                      System.out.println("Wrong input");
```


```
                    break;


                }

            }while(choice!=0);
        }
    }
```


//Circular LinkedList

```
import java.util.*;
class node
{
    int data;
    node next;
    node(int e)
    {
        data=e;
        next=null;
    }
}
class LinkedListClass
{
    node root,newnode,t,t2,last;
    LinkedListClass()
    {
        root=null;
        last=null;
    }
void insert_left(node n)
{
    if(root==null)
    {
        root=n;
        last=n;
        last.next=root;

    }
    else
    {
        n.next=root;
        root=n;
        last.next=root;
    }
}
node delete_left()
{
    t=null;
    if(root!=null)
    {
      t=root;
      root=root.next;
      last.next=root;

    }
  return(t);
}
void insert_right(node n)
{
```


```
    if(root==null)
    {
        root=n;
        last=n;
        last.next=root;
    }
    else
    {
        t=root;
        while(t!=last)
            t=t.next;
        t.next=n;
        last=n;
        last.next=root;

    }
}
node delete_right()
{
    t=null;
    if(root!=null)
    {
      t=root;
       while(t!=last)
       {
          t2=t;
          t=t.next;
       }
      last=t2;
      last.next=root;
      return(t);
    }
  return(t);
}

void printall()
{
    if(root!=null)
    {
        t=root;
        while(t.next!=last)
        {
            System.out.print("|" + t.data + "|->|");
            t=t.next;
        }
        System.out.print("|" + t.data + "|->|");
    }
}
boolean is_empty()
{
```


```
    if(root==null)
        return true;
    else
        return false;

}
}

public class CircularLinkedList {
     public static void main(String args[])
    {
        int choice,e;node n;
        Scanner in=new Scanner(System.in);
         LinkedListClass l=new LinkedListClass();
        do
        {
            System.out.println("\n1. Insert to Left\n2. Insert to right\n3. Delete Left\n4.
Delete Right\n5. Print Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                     System.out.println("enter element:");
                     e=in.nextInt();
                     n=new node(e);
                     l.insert_left(n);
                     break;

                case 2:
                     System.out.println("enter element:");
                     e=in.nextInt();
                     n=new node(e);
                     l.insert_right(n);
                     break;

                case 3:
                    if(!l.is_empty())
                        System.out.println("element removed is "+(l.delete_left()).data);
                    else
                        System.out.println("Sorry Linkelist is empty");
                     break;

                case 4:
                     if(!l.is_empty())
                        System.out.println("element removed is "+(l.delete_right()).data);
                    else
                        System.out.println("Sorry Linkelist is empty");
                     break;

                case 5:
```


```
                if(!l.is_empty())
                  l.printall();
                else
                  System.out.println("Sorry Linkelist is empty");
                break;

            case 0:
                System.out.println("Exiting program");
                 break;

            default:
                System.out.println("Wrong input");
                 break;


          }

        }while(choice!=0);
    }
}
```


//Doubly LinkeList

```
import java.util.*;

class node
{
  int data;

  node left , right;

  node(int e)
  {
    data =e;
    left = right =null;
  }
}

class DLL
{
    node root,t,t2;

  DLL()
  {
    root = null;
  }

  public void insert_left(node n)
  {


    if(root == null)
       root = n;
    else
    {
      n.right= root;
      root.left= n;
      root = n;
    }

  }

  public void insert_right(node n)
  {
   node t;
    if(root == null)
      root = n;
    else
    {
     t= root;
     while(t.right!=null)
      t=t.right;
```


```
      t.right= n;
      n.left= t;
    }
  }

  public node delete_left()
  {
    t =null;
    if(root != null)
    {
      t = root;
      root = root.right;
      root.left = null;
    }
      return t;
    }



  public node delete_right()
  {
   t= null;

    if(root!= null)
     {
      t = root;
      while(t.right!=null)
        t= t.right;

      t2 = t.left;
      t2.right = null;
     }
      return t;
    }



  public void printall()
  {
    t = root;
    while(t!=null)
   {
      System.out.print("<-|"+t.data+"|->");
      t = t.right;
   }
  }
  boolean is_empty()
{
    if(root==null)
```


```
        return true;
    else
        return false;

}
}
```



```
 public class DoublyLinkedList
{
  public static void main(String args[])
  {
     Scanner in = new Scanner(System.in);

     DLL l = new DLL();

     int e, choice;

     node n;

      do
        {
            System.out.println("\n1. Insert to Left\n2. Insert to right\n3. Delete Left\n4.
Delete Right\n5. Print Data\n0. Exit\n: ");
            choice=in.nextInt();
            switch(choice)
            {
                case 1:
                     System.out.println("enter element:");
                     e=in.nextInt();
                     n=new node(e);
                     l.insert_left(n);
                     break;

                case 2:
                     System.out.println("enter element:");
                     e=in.nextInt();
                     n=new node(e);
                     l.insert_right(n);
                     break;

                case 3:
                    if(!l.is_empty())
                       System.out.println("element removed is "+(l.delete_left()).data);
                    else
                       System.out.println("Sorry Linkelist is empty");
                     break;

                case 4:
                     if(!l.is_empty())
```


```
                System.out.println("element removed is "+(l.delete_right()).data);
              else
                System.out.println("Sorry Linkelist is empty");
              break;

        case 5:
              if(!l.is_empty())
                l.printall();
              else
                System.out.println("Sorry Linkelist is empty");
              break;

        case 0:
              System.out.println("Exiting program");
               break;

        default:
              System.out.println("Wrong input");
               break;


      }

    }while(choice!=0);
  }
}
```

//Binary Search Tree

```
import java.util.*;

class node
{
  int data;

  node left , right;

  node(int e)
  {
    data = e;

    left = right = null;
  }
}

class tree
{
  node root;

  tree()
```


```
{
  root = null;
}

node get_root()
{
  return root;
}

void insert( node r, node n)
{
  if(r == null)
    root=n;

  else
  {
     if(n.data<r.data)
     {
        if(r.left == null)
           r.left = n;
        else
           insert(r.left , n);
     }
     else
     {
        if(r.right == null)
           r.right = n;
        else
           insert(r.right , n);
     }
  }
}

void inorder(node r)
{
  if(r!= null)
  {
     inorder(r.left);  // L

     System.out.println(r.data+"\t"); // V

     inorder(r.right);  // R
  }
}

void preorder(node r)
{
  if(r!= null)
  {
     System.out.println(r.data+"\t"); // V
```


```
      preorder(r.left);  // L

      preorder(r.right);  // R
    }
  }

  void postorder(node r)
  {
    if(r!= null)
    {
      postorder(r.left);  // L

      postorder(r.right);  // R

      System.out.println(r.data+"\t"); // V
    }
  }

// count no. of elements in a tree

int count(node r , int counter)
{
    if(r!= null)
    {
      count(r.left, counter);

      counter++;

      count(r.right , counter);

    }
    return counter;
}

// function to count no. of leaf nodes

int Lcount(node r , int counter)
{
    if(r!= null)
    {
      Lcount(r.left , counter);

      if(r.left == null && r.right == null)
         counter++;

      Lcount(r.right , counter);

    }
```


```
   return counter;
}

void search(int key)
{
  if(root == null)
    System.out.println("Empty Tree");
  else
  {
    node t = root;

    while(t.data!= key && t!= null)
    {
       if(key< t. data)
          t= t.left;
       else
          t = t.right;
    }

    if(t!= null)
      System.out.println("Found");
    else
      System.out.println("Not Found");
  }
}

node deletion(int key)
{
  node p , r , t , c=null;

  if(root == null)
  {
    System.out.println("Empty Tree");
    return null;
  }
  else
  {
    r = root;
    p = r;

    while(r!= null && r.data!= key)
    {
      p = r;

      if(key< r.data)
         r = r.left;
      else
         r = r.right;
    }
```


```
if(r == null)
{
  System.out.println("Not Found");
  return null;
}
else
{
  // found
  // case 1

  if(r.left == null && r.right == null)
  {
    if(root == r)
    {
      root = null;
      return r;
    }
    else
    {
      if( p.left == r)
         p.left =null;
      else
         p.right = null;

      return r;
    }
  }

  // case 2

  if(r.left == null && r.right!= null)
  {
    if(r == root)
    {
       root = root.right;
       return r;
    }
    else
    {
       if(p.right == r)
          p.right = r.right;
       else
          p.left = r.right;

       return r;
    }
  }

  if(r.left!= null && r.right == null)
  {
```


```
  if(r == root)
  {
     root = root.left;
     return r;
  }
  else
  {
     if(p.left == r)
        p.left = r.left;
     else
        p.right = r.left;

     return r;
  }
}

// case 3

if( r.left!= null && r.right!= null)
{
   if(r == root)
   {
      c = root;
      r = r.right;
      t = r;

      while(r.left!= null)
      {
         t =r;
         r= r.left;
      }


      t.left = r.right;

      r.left = root.left;
      r.right = root.right;
      root = r;

   }
   else
   {
     c = r;
     r = r.right;
     t =r;

     while(r.left!= null)
     {
       t = r;
       r = r.right;
```


```
            }
            t.left = r.right;

            r.left = c.left;
            r.right = c.right;

            if(p.left == c)
               p.left = r;
            else
               p.right = r;

          } // else
        } // if (case 3 finished)
    } // else  (found finished)
  } // else   (function else finished)

  return c;

 }// function
} // class

class BST
{
  public static void main(String args[])
  {
     Scanner in = new Scanner(System.in);

     tree t =new tree();

     node x , r, n;

     int ch , e;

     do
     {
        System.out.println("\n1.INSERT \n2.INORDER \n3.PREORDER
\n4.POSTORDER \n5.SEARCH \n6.DELETE \n0.EXIT:");
        ch = in.nextInt();
        r = t.get_root();

        switch(ch)
        {

          case 1: System.out.println("Enter number:");
                  e = in.nextInt();
                  n = new node(e);
                  t.insert(r,n);
                  break;

          case 2: t.inorder(r);
```


```
            break;

        case 3: t.preorder(r);
            break;

        case 4: t.postorder(r);
            break;

        case 5: System.out.println("Enter Element To Search:");
            e = in.nextInt();
            t.search(e);
            break;

        case 6: System.out.println("Enter Element To Delete:");
            e = in.nextInt();
            x = t.deletion(e);

            if(x!= null)
              System.out.println("Deleted:"+x.data);
            break;

        case 0: System.out.println("Exiting");
            break;

        default: System.out.println("Wrong Input");
             break;
      } // switch
    }while(ch!= 0);
  }
}
```


//Postfix Based Expression Tree

```
import java.util.*;

class node
{
  char data;
  node left , right;

  node(char e)
  {
    data = e;
    left = right = null;
  }

}

class Stack
{
  int MaxSize , tos;
  node s[];

  Stack(int size)
  {
    MaxSize = size;
    tos = -1;
    s = new node[MaxSize];
  }

  void push(node e)
  {
    tos++;
    s[tos] = e;
  }

  node pop()
  {
    if(tos == -1)
    {
      System.out.println("Error in input\n");
      return null;
    }
    else
    {
      node t = s[tos];
      tos--;
      return t;
    }
  }
} // stack
```


```
public class Postfixtree
{
  public static void main(String args[])
  {
    node n , r , l , root=null;

    Scanner in = new Scanner(System.in);

    String postfix;

    Stack s = new Stack(50);

    System.out.println("Enter Postfix:");

    postfix = in.nextLine();



    for(int i = 0; i < postfix.length(); i++)
    {
      char c = postfix.charAt(i);
      n = new node(c);

      if( (n.data>='a' && n.data<='z') || (n.data>='A' && n.data<='Z') )
        s.push(n);
      else
        {
           r = s.pop();
           l = s.pop();
           n.left = l;
           n.right = r;
           s.push(n);
        }
    }

    if(s.tos!= 0)
      System.out.println("Error in Input");
    else
      root = s.pop();

    System.out.println("Postfix:"+ postfix);

    System.out.println("Prefix:");

    preorder(root);

  } // main

  public static void preorder(node r)
  {
```


```
        if( r!= null)
        {
            System.out.print(r.data+"\t");   // V
            preorder(r.left);    // L
            preorder(r.right);   // R
        }
    }
}       // class postfix tree
```


//Prefix Based Expression Tree

```
import java.util.*;

class node
{
  char data;
  node left , right;

  node(char e)
  {
    data = e;
    left = right = null;
  }

}

class Stack
{
  int MaxSize , tos;
  node s[];

  Stack(int size)
  {
    MaxSize = size;
    tos = -1;
    s = new node[MaxSize];
  }

  void push(node e)
  {
    tos++;
    s[tos] = e;
  }

  node pop()
  {
    if(tos == -1)
    {
      System.out.println("Error in input\n");
      return null;
    }
    else
    {
      node t = s[tos];
      tos--;
      return t;
    }
  }
} // stack
```


```
public class Prefixtree
{
  public static void main(String args[])
  {
    node n=null , r=null , l=null , root=null;

    Scanner in = new Scanner(System.in);

    String prefix;

    Stack s = new Stack(50);

    System.out.println("Enter Prefix:");

    prefix = in.nextLine();



    for(int i = prefix.length()-1; i >=0 ; i--)
    {
      char c = prefix.charAt(i);
      n = new node(c);

      if( (n.data>='a' && n.data<='z') || (n.data>='A' && n.data<='Z') )
        s.push(n);
      else
        {
           l = s.pop();
           r = s.pop();
           n.left = l;
           n.right = r;
           s.push(n);
        }
    }

    if(s.tos!= 0)
      System.out.println("Error in Input");
    else
      root = s.pop();

    System.out.println("Prefix:"+ prefix);

    System.out.println("Postfix:");

    postorder(root);

  } // main

 public static void postorder(node r)
```


```
  {
    if(r!= null)
    {
       postorder(r.left);  // L

       postorder(r.right);  // R

       System.out.print(r.data+" "); // V
    }
  }
}      // class prefix tree
```


//bubble sort

```
import java.util.*;
public class bubble_sort
{
   public static void main(String args[])
{
int a[],size;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
bubble(a,size);
}


static void bubble(int x[],int n)
   {
       for(int i=0;i<n;i++)
       {
           for(int j=0;j<n-1;j++)
           {
               if(x[j]>x[j+1])
               {
                   int t=x[j];
                   x[j]=x[j+1];
                   x[j+1]=t;
               }
           }
       }
   display(x,n);
}
    static void display(int x[],int n)
    {
      System.out.println("Sorted array is\n");
        for(int i=0;i<n;i++)
       {
        System.out.println(x[i]);
       }
   }
}
```

//heap sort

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


```
    public static void main(String args[])
{
int a[],size;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
heap(a,size);


}

static void heap(int x[],int n)
    {
    int pc,done;
        for(int i=n-1;i>=0;i--)
        {
          for(int j=0;j<=i;j++)
           {
             pc=j;
             done=0;
             while(pc>=0 && pc/2>=0 && done!=1)
                {
                 if(x[pc] > x[pc/2])
                 {
                    int t=x[pc];
                    x[pc]=x[pc/2];
                    x[pc/2]=t;
                    pc=pc/2;

                 }
                 else
                     done=1;

                }
           }
            int t=x[0];
            x[0]=x[i];
            x[i]=t;
       }

    display(x,n);
}
     static void display(int x[],int n)
     {
```


```
        System.out.println("Sorted array is\n");
          for(int i=0;i<n;i++)
         {
          System.out.println(x[i]);
         }
    }
}
```


//insertion sort

```
import java.util.*;
public class insertion_sort {
    public static void main(String args[])
{
int a[],size;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
insertion(a,size);
}


static void insertion(int x[],int n)
   {
    int newno,i,j;
    for(i=0;i<n-1;i++)
       {


        newno=x[i+1];
        for(j=i+1;j>0 && x[j-1]>newno;j--)
           {

            x[j]=x[j-1];
           }
        x[j]=newno;
       }
   display(x,n);
}

    static void display(int x[],int n)
    {
      System.out.println("Sorted array is\n");
        for(int i=0;i<n;i++)
       {
        System.out.println(x[i]);
       }
   }

}
```

//quick sort

```
import java.util.*;
```


```
class quick_sort {
    public static void main(String args[])
{
int a[],size;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
quick(a,0,size-1);


}


static void quick(int x[],int low,int high)
    {
    int i,j,pivot;
    pivot=(low+high)/2;
    i=low;
    j=high;
    while(i<j)
    {
        while(x[i]<x[pivot])
            i++;
        while(x[j]>x[pivot])
            j--;
        if(i<=j)
        {
            int t=x[i];
            x[i]=x[j];
            x[j]=t;
        }
     }
    if(low<j)
      quick(x,low,j-1);
    if(i<high)
      quick(x,i+1,high);

    display(x);
}

     static void display(int x[])
     {
       System.out.println("Sorted array is\n");
         for(int i=0;i<x.length;i++)
```


```
        {
         System.out.println(x[i]);
        }
    }

}
```


//radix sort

```
import java.util.*;
public class radix_sort {
 public static void main(String args[])
{
int a[],size,nod;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
        System.out.println("enter number of digits :");
        nod=in.nextInt();
radix(a,size,nod);


}


    static void radix(int x[],int n,int nod)
    {
        int b[][]=new int [10][10];
        int count[]=new int[10];
        int i,j,k,e=1,r,c,xindex;
        while(nod>0)
        {
            for(i=0;i<10;i++)
                count[i]=-1;
            for(i=0;i<n;i++)
            {
                r=(x[i]/e)%10;
                count[r]++;
                c=count[r];
                b[r][c]=x[i];

            }
        xindex=0;
        for(i=0;i<10;i++)
        {
            if(count[i]>-1)
            {
                for(k=0;k<=count[i];k++)
                    x[xindex++]=b[i][k];
            }
        }
        nod--;
```


```
        e=e*10;
        }
     display(x,n);
    }
static void display(int x[],int n)
    {
      System.out.println("Sorted array is\n");
        for(int i=0;i<n;i++)
        {
         System.out.println(x[i]);
        }
    }
}
```

//selection sort

```
import java.util.*;
public class selection_sort
{
    public static void main(String args[])
{
int a[],size;
Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
selection(a,size);


}

static void selection(int x[],int n)
    {
     int min,pos;
     for(int i=0;i<n;i++)
        {
            min=x[i];
            pos=i;
            for(int j=i+1;j<n;j++)
            {
                if(x[j]<min)
                {
                    min=x[j];
                    pos=j;

                }
```


```
            }
            x[pos]=x[i];
            x[i]=min;
        }
    display(x,n);
}
     static void display(int x[],int n)
     {
       System.out.println("Sorted array is\n");
         for(int i=0;i<n;i++)
        {
         System.out.println(x[i]);
        }
    }
}
```


//sequential search

```
import java.util.*;
public class sequential_search
{
 public static void main(String args[])
  {
        int a[],size,key;
        Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
        System.out.println("enter element to search :");
        key=in.nextInt();

int ans=sequential(a,size,key);
if(ans!=-1)
    System.out.println("Element found at:"+ans);
else
    System.out.println("Element not found");
}

 static int sequential(int x[],int n,int key)
 {
     for(int i=0;i<n;i++)
     {
         if(key==x[i])
             return(i);
     }

     return(-1);
 }

}
```


//binary search

```
import java.util.*;
public class binary_search
{
public static void main(String args[])
  {
        int a[],size,key;
        Scanner in=new Scanner(System.in);
        System.out.println("enter number of elements :");
        size=in.nextInt();
        a=new int[size];
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter element "+(i+1)+":");
            a[i]=in.nextInt();
        }
        System.out.println("enter element to search :");
        key=in.nextInt();

int ans=bin(a,0,size-1,key);
if(ans!=-1)
    System.out.println("Element found at:"+ans);
else
    System.out.println("Element not found");
}

 static int bin(int x[],int low,int high,int key)
 {
     int mid;
     if(low<=high)
     {
         mid=(low+high)/2;
         if(key==x[mid])
             return(mid);
         else
         {
             if(key<x[mid])
                 bin(x,low,mid-1,key);
             else
                 bin(x,mid+1,high,key);

         }
     }
     return(-1);
 }}
```