External Lab

1.Write a program to implement diet plan performance

```
import java.util.*;
class DietPlanPerformance
{
   public int dietPlanPerformance(int[] calories, int k, int lower, int upper)
   {
      int points = 0;
      int sum = 0;
      //logic
      for(int i=0;i<k;i++)
      {
         sum=sum+calories[i];
      }
      if(sum>upper)
      {
         points++;
      }
      else if(sum < lower)
      {
         points--;
      }
      int length=calories.length;
      for(int i=k;i<length;i++){
         sum=sum+calories[i]-calories[i-k];
         if(sum>upper)
         {
            points++;
         }
         else if(sum < lower)
         {
            points--;
         }

      }
      return points;
   }
   public static void main(String args[])
   {
      Scanner sc=new Scanner(System.in);
      System.out.println("enter calories size");
      int n=sc.nextInt();
      int calories[]=new int[n];
      System.out.println("enter the calroties");
      //read
      for(int i=0;i<n;i++)
```

```
        {
           calories[i]=sc.nextInt();
        }
        System.out.println("enter the days");
        //read
        int k=sc.nextInt();
        System.out.println("enter the Lower value");
        //read
        int l=sc.nextInt();
        System.out.println("enter the Upper value");
        //read
        int u=sc.nextInt();
        System.out.println(new
DietPlanPerformance().dietPlanPerformance(calories,k,l,u));
    }
}
```

2. Design a java program to remove all 1's with row and column flips in a binary matrix using bit manipulation

```
import java.util.*;
class  WithFlipsRemoveAllOnesBits
{
   public static boolean removeOnes(int grid[][])
   {
      //logic
      int r=grid.length,c=grid[0].length;
      int fr[]=grid[0],rr[]=reverse(fr);
      for(int i=1;i<r;i++)
      {
         for(int j=0;j<c;j++)
         {
            if(grid[0][0]==grid[i][0])
            {
               if(fr[j]!=grid[i][j])
               return false;
            }
            else
            {
               if(rr[j]!=grid[i][j])
               return false;
            }
         }
      }
      return true;
   }
```

```
    public static int[] reverse(int[] row)
    {
        //logic
        int rr[]=new int[row.length];
        for(int i=0;i<row.length;i++)
        rr[i]=row[i]^1;
        return rr;
    }
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter square matrix order");
        int n=s.nextInt();
        int grid[][]=new int[n][n];
        System.out.println("Enter the binary matrix");
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                grid[i][j]=s.nextInt();
            }
        }
        System.out.println(removeOnes(grid));
    }
}
```

3. Write a program to implement trie

```
import java.util.*;
class Trie
{
    class TrieNode
    {
        boolean isEnd;
        TrieNode arr[];
        TrieNode()
        {
            arr=new TrieNode[26];
            isEnd=false;
        }
    }
    TrieNode root;
    public Trie()
    {
        root = new TrieNode();
    }
```

```
    public void insert(String word)
    {
        //logic
        TrieNode node=root;
        for(int i=0;i<word.length();i++)
        {
            char ch=word.charAt(i);
            if(node.arr[ch-'a']==null)
            {
                TrieNode cur=new TrieNode();
                node.arr[ch-'a']=cur;
            }
            node=node.arr[ch-'a'];
        }
        node.isEnd=true;
    }

    public boolean search(String word)
    {
        //logic
        TrieNode node=root;
        for(int i=0;i<word.length();i++)
        {
            char ch=word.charAt(i);
            if(node.arr[ch-'a']==null) return false;
            node=node.arr[ch-'a'];
        }
        return node.isEnd;
    }

    public boolean startsWith(String prefix)
    {
        //logic
        TrieNode node=root;
        for(int i=0;i<prefix.length();i++)
        {
            char ch=prefix.charAt(i);
            if(node.arr[ch-'a']==null) return false;
            node=node.arr[ch-'a'];
        }
        return true;
    }
}
class  TrieDemo
{
    public static void main(String[] args)
```

```
    {
        Scanner s=new Scanner(System.in);
        String str[]=new String[10];
        Trie t=new Trie();
        for(int i=0;i<10;i++)
        {
            str[i]=s.next();
            t.insert(str[i]);
        }
        System.out.println("Enter string to search");
        String s1=s.next();
        System.out.println(t.search(s1));
        System.out.println("Enter prefix");
        String s2=s.next();
        System.out.println(t.startsWith(s2));
    }
}
```

4. Write a program to find the kth smallest sub array sum

```
import java.util.*;
class SumofKthSmallestSubArray
{
    public static int kthSmallestSubarraySum(int[] nums, int k)
    {
        int min = Integer.MAX_VALUE, sum = 0;

        //find the min sum and max sum
        for(int i=0;i<nums.length;i++)
        {
            if(nums[i]<min)
            min=nums[i];
            sum+=nums[i];
        }
        int low = min, high = sum;
        //logic
        while(low<high)
        {
            int mid=(low+high)/2;
            int c=countSubarrays(nums,mid);
            if(c<k)
            low=mid+1;
            else
            high=mid;
        }
        return low;
    }
```

```
    public static int countSubarrays(int[] nums, int threshold)
    {
        int count = 0;
        int sum = 0;
        int length = nums.length;
        int left = 0, right = 0;
        //logic
        while(right<length)
        {
            sum+=nums[right];
            while(sum>threshold)
            {
                sum-=nums[left];
                left++;
            }
            count+=(right-left)+1;
            right++;
        }
        return count;
    }
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);

        System.out.println("Enter the size of the array");
        //declare n and array a[]
        int n=s.nextInt(),a[]=new int[n];
        System.out.println("Enter array elements");
        //read array elements
        for(int i=0;i<n;i++)
        a[i]=s.nextInt();
        System.out.println("Enter the required position");
        int k=s.nextInt();
        int ct=kthSmallestSubarraySum(a,k);

        System.out.println(ct);
    }
}
```

5. Write a program to find the number of connected components in a graph using union-find algorithm

```
import java.util.*;

class ConnectedComponentsUF
{
```

```
int[] parent;
int[] size;
public int countComponents(int n, int[][] edges)
{
   parent = new int[n];
   size = new int[n];
   //initialize parent and size
   Arrays.fill(parent,-1);
   Arrays.fill(size,1);

   int components = n;
   for (int[] e : edges)
   {
      //find parents of e[0] and e[1]
      int p1=find(e[0]),p2=find(e[1]);
      //if parents are not equal perform union based on size
      if(p1!=p2)
      {
         if(size[p1]<size[p2])
         {
            parent[p1]=p2;
            size[p2]+=size[p1];
         }
         else
         {
            parent[p2]=p1;
            size[p1]+=size[p2];
         }
         components--;
      }
      //decrement components
   }
   return components;
}
private int find(int i)
{
   //logic
   while(parent[i]>=0) i=parent[i];
   return i;
}

public static void main(String args[])
{
   Scanner sc= new Scanner(System.in);
   int n=sc.nextInt();
   int e=sc.nextInt();
```

```
        int edges[][]=new int[e][2];
        for(int i=0;i<e;i++)
            for(int j=0;j<2;j++)
                edges[i][j]=sc.nextInt();
        System.out.println(new
ConnectedComponentsUF().countComponents(n,edges));
    }
}
```

6. Write a program to implement parallel courses using topological sort.

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

public class ParallelCourses
{
    public int minimumSemesters(int numCourses, int[][] prerequisites,int
maxCourses)
    {
        // create an adjacency list to represent the graph
        int graph[][]=new int[numCourses][numCourses];
        int[] indegree = new int[numCourses];

        // populate the adjacency list using the prerequisites array
        for(int pre[]:prerequisites)
        {
            int u=pre[0];
            int v=pre[1];
            graph[u][v]=1;
            indegree[v]++;
        }


        // Perform a topological sort to find the order in which the courses should
be taken
        Queue<Integer> queue = new LinkedList<>();
        for(int i=0;i<numCourses;i++)
        {
            if(indegree[i]==0) queue.offer(i);
        }

        int semesters = 0;
        int coursesTaken = 0;
        //logic
        while(!queue.isEmpty())
        {
```

```
            int cts=Math.min(queue.size(),maxCourses);
            for(int i=0;i<cts;i++)
            {
               int u=queue.poll();
               coursesTaken++;
               for(int v=0;v<numCourses;v++)
               {
                  if(graph[u][v]==1 && --indegree[v]==0) queue.offer(v);
               }
            }
            semesters++;
         }

         if (coursesTaken != numCourses) {
            return -1; // cannot complete all courses
         }

         return semesters;
      }

      public static void main(String[] args)
      {
         Scanner s=new Scanner(System.in);
         int numCourses=s.nextInt();
         int c=s.nextInt();
         int prerequisites[][]=new int[c][2];
         for(int i=0;i<c;i++)
         {
            for(int j=0;j<2;j++)
            {
               prerequisites[i][j]=s.nextInt();
            }
         }
         int maxCourses=s.nextInt();

         ParallelCourses p=new ParallelCourses();

         System.out.println(p.minimumSemesters(numCourses,prerequisites,maxC
ourses));
      }
}
```

7.Write a program to find the distinct numbers in each sub array

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

```
public static int[] distinct(int a[],int k)
{
   int n=a.length;
   int ans[]=new int[n-k+1];
   //logic
   int x=0;
   HashMap<Integer,Integer>m=new HashMap<>();
   for(int i=0;i<k;i++)
   {
      m.put(a[i],m.getOrDefault(a[i],0)+1);
   }
   ans[x++]=m.size();
   for(int i=k;i<n;i++)
   {
      int cur=a[i],prev=a[i-k];
      m.put(prev,m.getOrDefault(prev,0)-1);
      if(m.get(prev)==0)
      m.remove(prev);
      m.put(cur,m.getOrDefault(cur,0)+1);
      ans[x++]=m.size();
   }
   return ans;
}
public static void main(String[] args)
{
   Scanner s=new Scanner(System.in);

   System.out.println("Enter the size of the array");
   int n=s.nextInt();

   int a[]=new int[n];

   System.out.println("Enter array elements");
   //read array
   for(int i=0;i<n;i++)
   a[i]=s.nextInt();

   System.out.println("Enter the window size");
   //read size k
   int k=s.nextInt();


   int[] ans=distinct(a,k);

   for(int i=0;i<ans.length;i++)
      System.out.print(ans[i]+" ");
```

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

8. Write a program to check whether an abbreviation is valid for a given word or not using 2 pointer approach.

```
import java.util.*;

class ValidAbbreviation
{
    public static boolean validity(String word, String abbr)
    {
        int i=0,j=0;
        if(word==null||abbr==null)
            return false;
        //logic
        while(i<word.length() && j<abbr.length())
        {
            if(Character.isDigit(abbr.charAt(j)))
            {
                if(abbr.charAt(j)=='0')
                return false;
                else
                {
                    int sum=0;
                    while(j<abbr.length() && Character.isDigit(abbr.charAt(j)))
                    {
                        sum=sum*10+(abbr.charAt(j)-'0');
                        j++;
                    }
                    i+=sum;
                }
            }
            else
            {
                if(word.charAt(i)!=abbr.charAt(j))
                return false;
                i++;j++;
            }
        }
        return i==word.length()&&j==abbr.length();
    }

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

```
        System.out.println("Enter word");
        String word=sc.next();
        System.out.println("Enter abbreviation");
        String abbr=sc.next();
        System.out.println(validity(word,abbr));
    }
}
```

9. Write a java program to count the number of 1's in a bit representation of a given number

```
import java.util.*;
class CountingBitsWithBitwiseOperators
{
    public static int[] countBits(int n)
    {
        //create an array r of size n+1
        int r[]=new int[n+1];
        r[0]=0;
        for(int i=1;i<=n;i++)
        {
            int ct=0,x=i;
            while(x>0)
            {
                ct=ct+(x&1);
                x=x>>1;
            }
            r[i]=ct;
        }
        //logic
        return r;
    }
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        int r[]=new int[n+1];
        r=countBits(n);
        for(int i=0;i<=n;i++)
        System.out.println(" "+r[i]);
    }
}
```

10. Write a program to find the longest string in words array such that every prefix of it is also in words using a trie data structure.

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

```
class TrieNode
{
   TrieNode arr[] = new TrieNode[26];
   boolean isEnd;
}

class LongestWord
{
   TrieNode root = new TrieNode();
   String res = "";
   public String longestWord(String[] words)
   {
      for (String word : words) addWord(word);
      for (String word : words) searchPrefix(word);
      return res;
   }

   private void searchPrefix(String word)
   {
      //logic
      TrieNode cur=root;
      for(int i=0;i<word.length();i++)
      {
         char ch=word.charAt(i);
         cur=cur.arr[ch-'a'];
         if(cur.isEnd==false) return;
      }
      if((res.length()<word.length())||(res.length()==word.length()&&res.compa
reTo(word)>0)) res=word;
   }

   private void addWord(String word)
   {
      //logic
      TrieNode node=root;
      for(int i=0;i<word.length();i++)
      {
         char ch=word.charAt(i);
         if(node.arr[ch-'a']==null)
         {
            TrieNode cur=new TrieNode();
            node.arr[ch-'a']=cur;
         }
         node=node.arr[ch-'a'];
      }
      node.isEnd=true;
```

```
    }
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        String dict[]=sc.nextLine().split(" ");
        System.out.println(new LongestWord().longestWord(dict));
    }
}
```

11.Write a program to find the kth largest element in an array using treap

```
import java.util.*;
class TreapNode
{
    int data;
    int priority;
    TreapNode left;
    TreapNode right;
    TreapNode(int data)
    {
        this.data = data;
        this.priority = new Random().nextInt(1000);
        this.left = this.right = null;
    }
}
class KthLargest
{
    static int k;
    public static TreapNode rotateLeft(TreapNode root)
    {
        //logic
        TreapNode R=root.right;
        TreapNode X=root.right.left;
        R.left=root;
        root.right=X;
        return R;
    }
    public static TreapNode rotateRight(TreapNode root)
    {
        //logic
        TreapNode L=root.left;
        TreapNode Y=root.left.right;
        L.right=root;
        root.left=Y;
        return L;
```

```
}
public static TreapNode insertNode(TreapNode root, int data)
{
   //logic
   if(root==null)
   {
      return new TreapNode(data);
   }
   if(data<root.data)
   {
      root.left=insertNode(root.left,data);
      if(root.left!=null && root.left.priority>root.priority)
      root=rotateRight(root);
   }
   else
   {
      root.right=insertNode(root.right,data);
      if(root.right!=null && root.right.priority>root.priority)
      root=rotateLeft(root);
   }
   return root;
}
 static void inorder(TreapNode root)
{
   //logic
   if(root !=null){
      inorder(root.left);
      k--;
      if(k==0){
         System.out.print(" "+root.data);
         return;
      }
      inorder(root.right);
   }
}
public static void main(String[] args)
{
   Scanner sc = new Scanner(System.in);
   System.out.println("Enter number of elements");
   int n = sc.nextInt();
   System.out.println("Enter the position");
   int p = sc.nextInt();
   k=n-p+1;
   int arr[] = new int[n];
   for(int i=0;i<n;i++)
   {
```

```
            arr[i] = sc.nextInt();
        }
        TreapNode root = null;
        for(int a:arr)
        {
            root = insertNode(root,a);
        }
        inorder(root);
    }
}
```

12. Write a program to find the articulation points in a graph using depth first search.

```
import java.util.*;

class Graph
{
    private int V; // number of vertices
    private int[][] adj; // adjacency matrix
    private int time; // time used in DFS
    private static final int NIL = -1; // constant value for uninitialized variables

    public Graph(int V)
    {
        //logic
        adj=new int[V][V];
        this.V=V;
    }

    // add an edge to the graph
    public void addEdge(int v, int w)
    {
        //logic
        adj[v][w]=1;
        adj[w][v]=1;
    }

    // utility function to find articulation points in the graph
    private void APUtil(int u, boolean[] visited, int[] disc, int[] low, int[] parent,
boolean[] ap)
    {
        // count children in DFS tree
        int children = 0;

        // mark current node as visited and initialize discovery time and low value
       visited[u]=true;
```

```
    disc[u]=low[u]=++time;

    // loop through all vertices adjacent to this vertex
    for(int w=0;w<V;w++)
    {
       if(adj[u][w]==1)
       {
          if(!visited[w])
          {
             children++;
             parent[w]=u;
             APUtil(w,visited,disc,low,parent,ap);
             low[u]=Math.min(low[u],low[w]);
             if(parent[u]==NIL && children>1) ap[u]=true;
             if(parent[u]!=NIL && low[w]>=disc[u]) ap[u]=true;
          }
          else if(w!=parent[u])
          low[u]=Math.min(low[u],disc[w]);
       }
    }

          // if v is not visited yet, then make it a child of u in DFS tree and
recur for it

             // check if the subtree rooted with v has a connection to one of the
ancestors of u


             // u is an articulation point in the following cases:
             // 1. u is the root of DFS tree and has two or more children


             // 2. if u is not the root and low value of one of its child is more
than discovery value of u

          // update low value of u for parent function calls

   }

   // main function to find articulation points in the graph
   public void AP()
   {
      boolean[] visited = new boolean[V];
      int[] disc = new int[V];
      int[] low = new int[V];
      int[] parent = new int[V];
```

```
        boolean[] ap = new boolean[V];

        // initialize parent and visited, and ap arrays
        for(int i=0;i<V;i++)
        {
            ap[i]=false;
            visited[i]=false;
            parent[i]=NIL;
        }


        // call the recursive helper function to find articulation points
        for(int i=0;i<V;i++)
        {
            if(!visited[i])
            APUtil(i,visited,disc,low,parent,ap);
        }

        // print articulation points
        for (int i = 0; i < V; i++) {
            if (ap[i]) {
                System.out.print(i + " ");
            }
        }
    }
}

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

        int vertices = s.nextInt();
        int edges = s.nextInt();

        Graph g = new Graph(vertices);

        for (int i = 1; i <= edges; i++)
        {
            int start = s.nextInt();
            int end = s.nextInt();
            g.addEdge(start, end);
        }
        g.AP();
    }
```

```
}
```

13. Write a program to check whether a linked list is palindrome or not

```
import java.util.*;
class node
{
      int data;
      node next;
}
class LList
{
//  Stack<Integer> s=new Stack<>();
   node head=new node();

   LList(){
      head=null;
   }
      void create(int x){
      node t=head,nn=new node();
      nn.data=x;
     // s.push(x);
      nn.next=null;
      if(head==null)
      {
         head=nn;
         return;
      }
      while(t.next!=null)
      {t=t.next;
      }
      t.next=nn;
      }

   void display()
   {
      node temp=head;
      while(temp!=null)
      {
         System.out.print(temp.data+" ");
         temp=temp.next;
      }
      System.out.println();
   }
   boolean isPalindrome()
   {
```

```
        java.util.Stack<Integer>s=new java.util.Stack<>();
        node fp=head,sp=head;
        while(sp!=null)
        {
           s.push(sp.data);
           sp=sp.next;
        }
        while(fp!=null)
        {
           if(fp.data!=s.pop())
           return false;
           fp=fp.next;
        }
        return true;
    }
}

class  PalindromeListNR
{
    public static void main(String[] args)
    {
        LList l=new LList();
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the number of values");
        int n=s.nextInt();
        System.out.println("Enter  values");
        for(int i=1;i<=n;i++)
        {
           int x=s.nextInt();
           l.create(x);
        }
        System.out.println("List is");
        l.display();

        System.out.println("Is Palindrome "+l.isPalindrome());
    }
}
```

14. Write a program to find the maximum flow in a graph from source to sink using breadth first search.

```
import java.util.*;

public class MaxFlow
{
    static int V; // number of vertices in the graph
```

```
   // method to find the maximum flow in a flow network using the Edmonds-Karp algorithm
   static int findMaxFlow(int[][] graph, int source, int sink)
   {
      int[][] residualGraph = new int[V][V];
      //copy graph into residual graph
      for(int i=0;i<V;i++)
      {
         for(int j=0;j<V;j++)
         {
            residualGraph[i][j]=graph[i][j];
         }
      }

      int[] parent = new int[V];
      int maxFlow = 0;

      while (bfs(residualGraph,source,sink,parent))
      {
         int pathFlow = Integer.MAX_VALUE;
         //find min path flow
         for(int v=sink;v!=source;v=parent[v])
         {
            int u=parent[v];
            pathFlow=Math.min(pathFlow,residualGraph[u][v]);
         }

         //update residual graph and maxFlow
         for(int v=sink;v!=source;v=parent[v])
         {
            int u=parent[v];
            residualGraph[u][v]-=pathFlow;
            residualGraph[v][u]+=pathFlow;
         }
         maxFlow+=pathFlow;

      }
      return maxFlow;
   }

   // helper method to find the shortest augmenting path in the residual graph using BFS
   static boolean bfs(int[][] residualGraph, int source, int sink, int[] parent)
   {
      boolean[] visited = new boolean[V];
```

```
        //update visited with false

        Queue<Integer> queue = new java.util.LinkedList<>();
        //add source to queue
        queue.add(source);
        visited[source] = true;
        parent[source] = -1;
        while(!queue.isEmpty())
        {
            int u=queue.poll();
            for(int v=0;v<V;v++)
            {
                if(visited[v]==false && residualGraph[u][v]>0)
                {
                    queue.add(v);
                    visited[v]=true;
                    parent[v]=u;
                }
            }
        }
        return visited[sink];
    }

    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter number of vertices");
        V=s.nextInt();
        int[][] graph = new int[V][V];
        System.out.println("Enter the adjacency matrix of the directed graph");
        for(int i=0;i<V;i++)
            for(int j=0;j<V;j++)
            graph[i][j]=s.nextInt();
        System.out.println("Enter source and sink");
        int source = s.nextInt();
        int sink = s.nextInt();
        int maxFlow = findMaxFlow(graph, source, sink);
        System.out.println(maxFlow);
    }
}
```

15. Write a program to find the lexicographically smallest equivalent string using union and find.

```
import java.util.*;
class LexSmallestEquivalentString
```

```
{
   static int p[];
   public static String smallestEquivalentString(String A, String B, String S)
   {
      p=new int[26];
      //assign value to parent
      Arrays.fill(p,-1);

      for(int i = 0; i < A.length(); i++)
      {
         //get the value of every character
         int a=A.charAt(i)-'a',b=B.charAt(i)-'a';
         //find its parent
         int p1=find(a),p2=find(b);
         //perform union by making smallest charater as parent
         if(p1<p2) p[p2]=p1;
         else if(p2<p1) p[p1]=p2;
      }
      StringBuilder sb = new StringBuilder();
      //read every character from the required string and add its root to sb
      for(int i=0;i<S.length();i++)
      {
         int c=S.charAt(i)-'a';
         sb.append((char)('a'+find(c)));
      }
      return sb.toString();
   }

   static int find(int i)
   {
      //logic
      while(p[i]>=0) i=p[i];
      return i;
   }

   public static void main(String args[])
   {
      Scanner sc=new Scanner(System.in);
      String A=sc.next();
      String B=sc.next();
      String T=sc.next();
      System.out.println(smallestEquivalentString(A,B,T));
   }
}
```