VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

ADVANCED DATA STRUCTURES (22CS5PEADS)

Submitted by

G SANJANA HEBBAR (1BM21CS062)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
March -June 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **“ADVANCED DATA STRUCTURES”** carried out by G SANJANA HEBBAR **(1BM21CS062)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Advanced Data structures Lab - **(22CS5PEADS)** work prescribed for the said degree.

Prof. Namratha M	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	2. getMin(H): A simple way to getMin() is to traverse the list of root of Binomial Trees and return the minimum key. 3. extractMin(H): This operation also uses union(). We first call getMin() to find the minimum key Binomial Tree, then we remove the node and create a new Binomial Heap by connecting all subtrees of the removed minimum node. Finally we call union() on H and the newly created Binomial Heap.	
10	Write a program to implement the following functions on a Binomial heap: 1. delete(H): Like Binary Heap, delete operation first reduces the key to minus infinite, then calls extractMin(). 2. decreaseKey(H): decreaseKey() is also similar to Binary Heap. We compare the decreases key with it parent and if parent's key is more, we swap keys and recur for parent. We stop when we either reach a node whose parent has smaller key or we hit the root node.	51

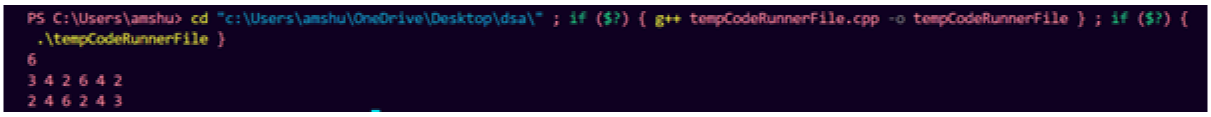
Lab Program 1:

Q)Write a program to implement the following list: An ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory efficient version of Doubly LinkedList can be created using only one space for address field with every node. This memory efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bitwise XOR operation to save space for one address. In the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.

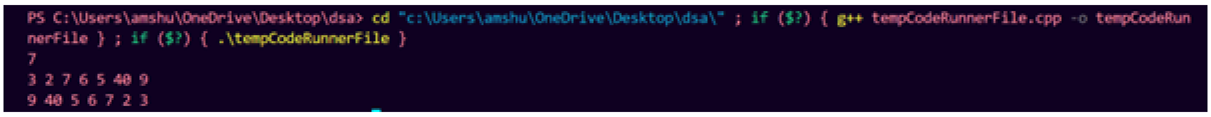
Code:

```
import java.util.Scanner;

class Node {
    int data;
    Node npx;


    public Node(int data) {
        this.data = data;
        this.npx = null;
    }
}


public class Main {
    static Node XOR(Node a, Node b) {
        return new Node(System.identityHashCode(a) ^ System.identityHashCode(b));
    }


    static Node insert(Node head_ref, int data) {
        Node new_node = new Node(data);
        new_node.npx = head_ref;


        if (head_ref != null) {
            new_node.npx = XOR(new_node, head_ref.npx);
        }
        head_ref = new_node;
        return head_ref;
    }

    static void printList(Node head) {
        System.out.println("Following are the nodes of Linked List:");

        Node curr = head;
        Node prev = null;
        Node next;

        while (curr != null) {
            System.out.println(curr.data);
            next = XOR(prev, curr.npx);
            prev = curr;
            curr = next;
        }
    }

    public static void main(String[] args) {
        Scanner = new Scanner(System.in);
        Node head = null;

        System.out.print("Enter the number of elements to be inserted: ");
        int n = scanner.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter element " + (i + 1) + ": ");
            int num = scanner.nextInt();
            head = insert(head, num);
        }

        printList(head);
        scanner.close();
    }
}
```

Output:



```
PS C:\Users\amshu> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
 .\tempCodeRunnerFile }
6
3 4 2 6 4 2
2 4 6 2 4 3
```



```
PS C:\Users\amshu\OneDrive\Desktop\dsa> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRun
nerFile } ; if ($?) { .\tempCodeRunnerFile }
7
3 2 7 6 5 40 9
9 40 5 6 7 2 3
```

Lab Program 2:

Q) Write a program to perform insertion, deletion and searching operations on a skip list.

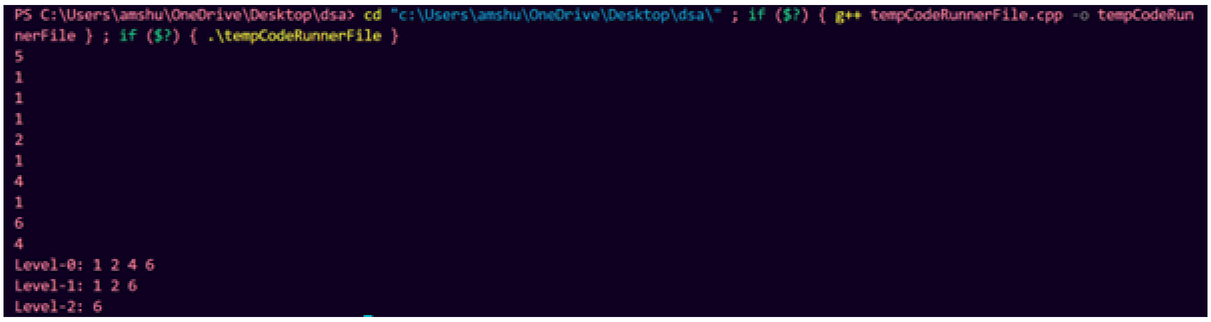
Code:

```
import java.util.Scanner;

class Node {
    int key;
    Node forward[];

    Node(int key, int level) {
        this.key = key;
        forward = new Node[level + 1];
    }
}

class SkipList {
    int MAXLVL;
    float P;
    int level;
    Node header;

    SkipList(int MAXLVL, float P) {
        this.MAXLVL = MAXLVL;
        this.P = P;
        level = 0;
        header = new Node(-1, MAXLVL);
    }
```

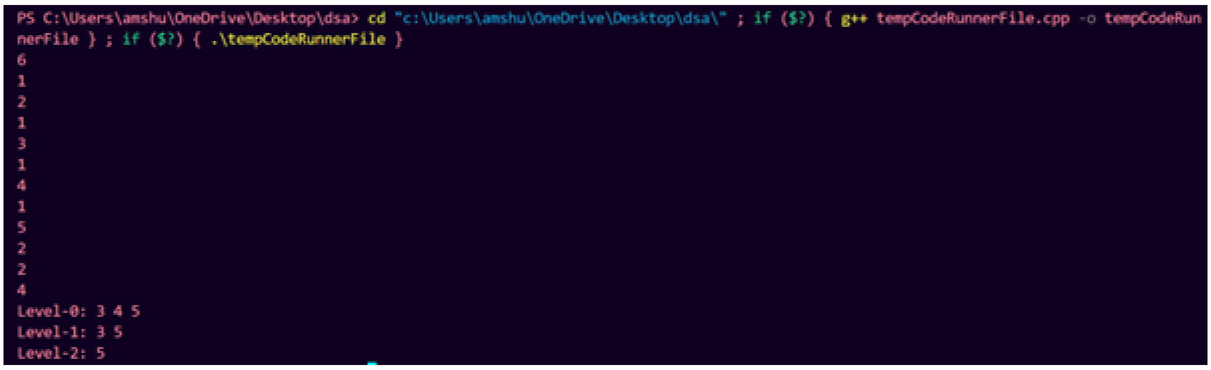
```
int randomLevel() {
    float r = (float)Math.random();
    int lvl = 0;
    while (r < P && lvl < MAXLVL) {
        lvl++;
        r = (float)Math.random();
    }
    return lvl;
}

Node createNode(int key, int level) {
    Node n = new Node(key, level);
    return n;
}

void insertElement(int key) {
    Node current = header;
    Node update[] = new Node[MAXLVL + 1];
    for (int i = level; i >= 0; i--) {
        while (current.forward[i] != null && current.forward[i].key < key)
            current = current.forward[i];
        update[i] = current;
    }
    current = current.forward[0];
    if (current == null || current.key != key) {
        int rlevel = randomLevel();
        if (rlevel > level) {
            for (int i = level + 1; i < rlevel + 1; i++)
```

```
                update[i] = header;
            level = rlevel;
        }
        Node n = createNode(key, rlevel);
        for (int i = 0; i <= rlevel; i++) {
            n.forward[i] = update[i].forward[i];
            update[i].forward[i] = n;
        }
        System.out.println("Successfully Inserted key " + key);
    }
}

void displayList() {
    System.out.println("\n*****Skip List*****");
    for (int i = 0; i <= level; i++) {
        Node = header.forward[i];
        System.out.print("Level " + i + ": ");
        while (node != null) {
            System.out.print(node.key + " ");
            node = node.forward[i];
        }
        System.out.println();
    }
}

void deleteElement(int key) {
    Node current = header;
    Node[] update = new Node[MAXLVL + 1];
    for (int i = level; i >= 0; i--) {
            while (current.forward[i] != null && current.forward[i].key < key)
                current = current.forward[i];
            update[i] = current;
        }
        current = current.forward[0];
        if (current != null && current.key == key) {
            for (int i = 0; i <= level; i++) {
                if (update[i].forward[i] != current)
                    break;
                update[i].forward[i] = current.forward[i];
            }
            while (level > 0 && header.forward[level] == null) {
                level--;
            }
            System.out.println("Successfully deleted key " + key);
        }
    }

    void searchElement(int key) {
        Node current = header;
        for (int i = level; i >= 0; i--) {
            while (current.forward[i] != null && current.forward[i].key < key) {
                current = current.forward[i];
            }
        }
        current = current.forward[0];
        if (current != null && current.key == key) {
            System.out.println("Key found");
        } else {
            System.out.println("Key not found");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        SkipList lst = new SkipList(3, 0.5f);
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nMENU:");
            System.out.println("1. Insert an element");
            System.out.println("2. Delete an element");
            System.out.println("3. Search for an element");
            System.out.println("4. Display list");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter the element to be inserted: ");
                    int insertKey = scanner.nextInt();
                    lst.insertElement(insertKey);
                    break;
                case 2:
                    System.out.print("Enter the element to be deleted: ");
                    int deleteKey = scanner.nextInt();
                lst.deleteElement(deleteKey);
                break;
            case 3:
                System.out.print("Enter the element to be searched: ");
                int searchKey = scanner.nextInt();
                lst.searchElement(searchKey);
                break;
            case 4:
                lst.displayList();
                break;
            case 5:
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid Choice.");
            }
        }
    }
}
```

Output:



```
PS C:\Users\amshu\OneDrive\Desktop\dsa> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRun
nerFile } ; if ($?) { .\tempCodeRunnerFile }
5
1
1
1
2
1
4
1
6
4
Level-0: 1 2 4 6
Level-1: 1 2 6
Level-2: 6
```



```
PS C:\Users\amshu\OneDrive\Desktop\dsa> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRun
nerFile } ; if ($?) { .\tempCodeRunnerFile }
6
1
2
1
3
1
4
1
5
2
2
4
Level-0: 3 4 5
Level-1: 3 5
Level-2: 5
```

Lab Program 3:

Q) Given a boolean 2D matrix, find the number of islands.

A group of connected 1s forms an island. For example, the below matrix contains 5

islands

{1, 1, 0, 0, 0},

{0, 1, 0, 0, 1},

{1, 0, 0, 1, 1},

{0, 0, 0, 0, 0},

{1, 0, 1, 0, 1}

A cell in the 2D matrix can be connected to 8 neighbours.Use disjoint sets to implement the above scenario.

Code:

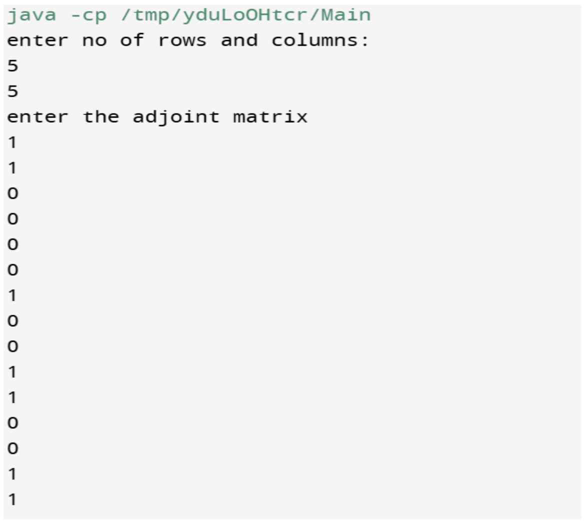
```
import java.io.*;

import java.util.*;


public class Main

{

    public static void main(String[] args)throws IOException

    {

        Scanner ss=new Scanner(System.in);

        System.out.println("enter no of rows and columns:");

        int r=ss.nextInt();

        int q=ss.nextInt();

        int[][] a = new int[r][q];

        System.out.println("enter the adjoint matrix");

        for(int i=0;i<r;i++){

            for(int j=0;j<q;j++){

                a[i][j]=ss.nextInt();

            }

        }
```

```
  System.out.println("Number of Islands is: " +
               countIslands(a));
               ss.close();
}



static int countIslands(int a[][])
{
  int n = a.length;
  int m = a[0].length;

  DisjointUnionSets dus = new DisjointUnionSets(n*m);



  for (int j=0; j<n; j++)
  {
     for (int k=0; k<m; k++)
     {

        if (a[j][k] == 0)
           continue;



        if (j+1 < n && a[j+1][k]==1)
           dus.union(j*(m)+k, (j+1)*(m)+k);
        if (j-1 >= 0 && a[j-1][k]==1)
           dus.union(j*(m)+k, (j-1)*(m)+k);
        if (k+1 < m && a[j][k+1]==1)
```

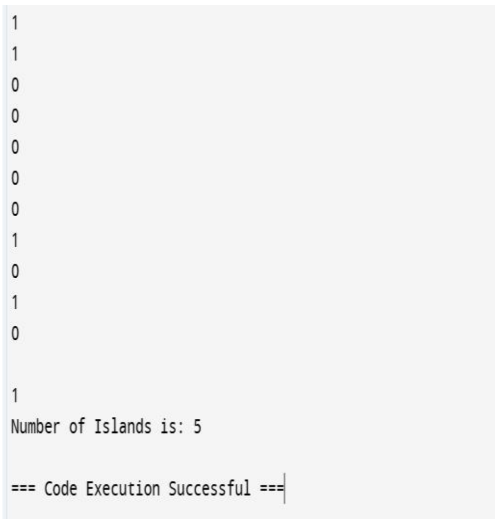
```
            dus.union(j*(m)+k, (j)*(m)+k+1);
        if (k-1 >= 0 && a[j][k-1]==1)
            dus.union(j*(m)+k, (j)*(m)+k-1);
        if (j+1<n && k+1<m && a[j+1][k+1]==1)
            dus.union(j*(m)+k, (j+1)*(m)+k+1);
        if (j+1<n && k-1>=0 && a[j+1][k-1]==1)
            dus.union(j*m+k, (j+1)*(m)+k-1);
        if (j-1>=0 && k+1<m && a[j-1][k+1]==1)
            dus.union(j*m+k, (j-1)*m+k+1);
        if (j-1>=0 && k-1>=0 && a[j-1][k-1]==1)
            dus.union(j*m+k, (j-1)*m+k-1);
    }
}



int[] c = new int[n*m];
int numberOfIslands = 0;
for (int j=0; j<n; j++)
{
    for (int k=0; k<m; k++)
    {
        if (a[j][k]==1)
        {

            int x = dus.find(j*m+k);



            if (c[x]==0)
            {
                    numberOfIslands++;
                    c[x]++;
                }

                else
                    c[x]++;
            }
        }
    }
    return numberOfIslands;
  }
}


class DisjointUnionSets
{
    int[] rank, parent;
    int n;

    public DisjointUnionSets(int n)
    {
        rank = new int[n];
        parent = new int[n];
        this.n = n;
        makeSet();
    }

    void makeSet()
    {
```

```
    for (int i=0; i<n; i++)
        parent[i] = i;
}



int find(int x)
{
    if (parent[x] != x)
    {

        parent[x]=find(parent[x]);
    }

    return parent[x];
}



void union(int x, int y)
{

    int xRoot = find(x);
    int yRoot = find(y);



    if (xRoot == yRoot)
        return;
        if (rank[xRoot] < rank[yRoot])

            parent[xRoot] = yRoot;



        else if(rank[yRoot]<rank[xRoot])

            parent[yRoot] = xRoot;


        else

        {


            parent[yRoot] = xRoot;



            rank[xRoot] = rank[xRoot] + 1;

        }

    }

}
```

Output:



```
java -cp /tmp/yduLoOHtcr/Main
enter no of rows and columns:
5
5
enter the adjoint matrix
1
1
0
0
0
0
1
0
0
1
1
0
0
1
1
```



```
1
1
0
0
0
0
0
1
0
1
0


1
Number of Islands is: 5

=== Code Execution Successful ===
```

Lab Program 4:

Q) Write a program to perform insertion and deletion operations on AVL trees.

Code:

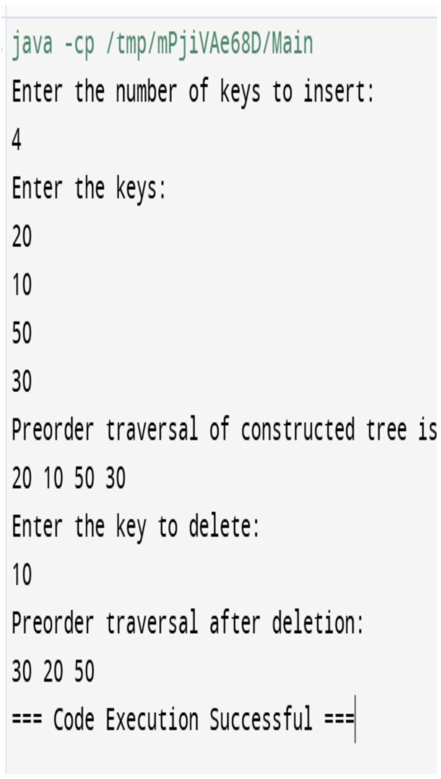
```
import java.util.Scanner;


class Node {
    int key, height;
    Node left, right;


    Node(int d) {
        key = d;
        height = 1;
    }
}


class AVLTree {
    Node root;


    int height(Node N) {
        if (N == null)
            return 0;
        return N.height;
    }


    int max(int a, int b) {
        return (a > b) ? a : b;
    }


    Node rightRotate(Node y) {
```

```
    Node x = y.left;
    Node T2 = x.right;

    x.right = y;
    y.left = T2;

    y.height = max(height(y.left), height(y.right)) + 1;
    x.height = max(height(x.left), height(x.right)) + 1;

    return x;
}

Node leftRotate(Node x) {
    Node y = x.right;
    Node T2 = y.left;

    y.left = x;
    x.right = T2;

    x.height = max(height(x.left), height(x.right)) + 1;
    y.height = max(height(y.left), height(y.right)) + 1;

    return y;
}

int getBalance(Node N) {
    if (N == null)
        return 0;
    return height(N.left) - height(N.right);
```

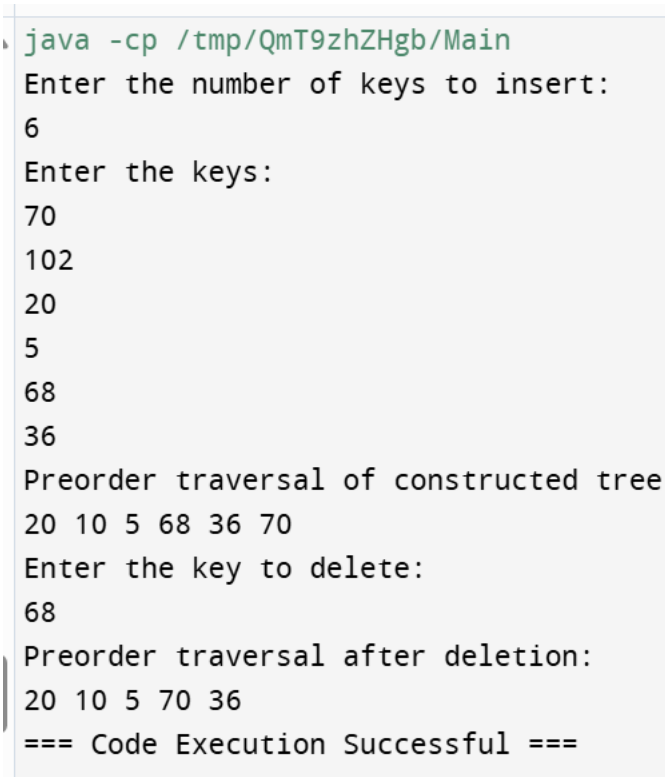
```
}

Node minValueNode(Node node) {
    Node current = node;

    while (current.left != null)
        current = current.left;

    return current;
}

Node insert(Node, int key) {
    if (node == null)
        return new Node(key);

    if (key < node.key)
        node.left = insert(node.left, key);
    else if (key > node.key)
        node.right = insert(node.right, key);
    else
        return node;

    node.height = 1 + max(height(node.left), height(node.right));

    int balance = getBalance(node);

    if (balance > 1 && key < node.left.key)
        return rightRotate(node);
    if (balance < -1 && key > node.right.key)
        return leftRotate(node);

    if (balance > 1 && key > node.left.key) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node.right.key) {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }

    return node;
}

Node deleteNode(Node root, int key) {
    if (root == null)
        return root;

    if (key < root.key)
        root.left = deleteNode(root.left, key);
    else if (key > root.key)
        root.right = deleteNode(root.right, key);
    else {
        if ((root.left == null) || (root.right == null)) {
            Node temp = null;
            if (temp == root.left)
                temp = root.right;
            else
                temp = root.left;

            if (temp == null) {
                temp = root;
                root = null;
            } else
                root = temp;
        } else {
            Node temp = minValueNode(root.right);
            root.key = temp.key;
            root.right = deleteNode(root.right, temp.key);
        }
    }

    if (root == null)
        return root;

    root.height = max(height(root.left), height(root.right)) + 1;

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root.left) >= 0)
        return rightRotate(root);

    if (balance > 1 && getBalance(root.left) < 0) {
        root.left = leftRotate(root.left);
        return rightRotate(root);
    }
```

```
        if (balance < -1 && getBalance(root.right) <= 0)
            return leftRotate(root);

        if (balance < -1 && getBalance(root.right) > 0) {
            root.right = rightRotate(root.right);
            return leftRotate(root);
        }

        return root;
    }

    void preOrder(Node node) {
        if (node != null) {
            System.out.print(node.key + " ");
            preOrder(node.left);
            preOrder(node.right);
        }
    }

    public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of keys to insert:");
        int numKeys = scanner.nextInt();

        System.out.println("Enter the keys:");
        for (int i = 0; i < numKeys; i++) {
            int key = scanner.nextInt();

            tree.root = tree.insert(tree.root, key);

        }


        System.out.println("Preorder traversal of constructed tree is:");

        tree.preOrder(tree.root);


        System.out.println("\nEnter the key to delete:");

        int keyToDelete = scanner.nextInt();

        tree.root = tree.deleteNode(tree.root, keyToDelete);


        System.out.println("Preorder traversal after deletion:");

        tree.preOrder(tree.root);

    }

}
```

Output:



```
java -cp /tmp/mPjiVAe68D/Main
Enter the number of keys to insert:
4
Enter the keys:
20
10
50
30
Preorder traversal of constructed tree is
20 10 50 30
Enter the key to delete:
10
Preorder traversal after deletion:
30 20 50
=== Code Execution Successful ===
```



```
java -cp /tmp/QmT9zhZHgb/Main
Enter the number of keys to insert:
6
Enter the keys:
70
102
20
5
68
36
Preorder traversal of constructed tree
20 10 5 68 36 70
Enter the key to delete:
68
Preorder traversal after deletion:
20 10 5 70 36
=== Code Execution Successful ===
```

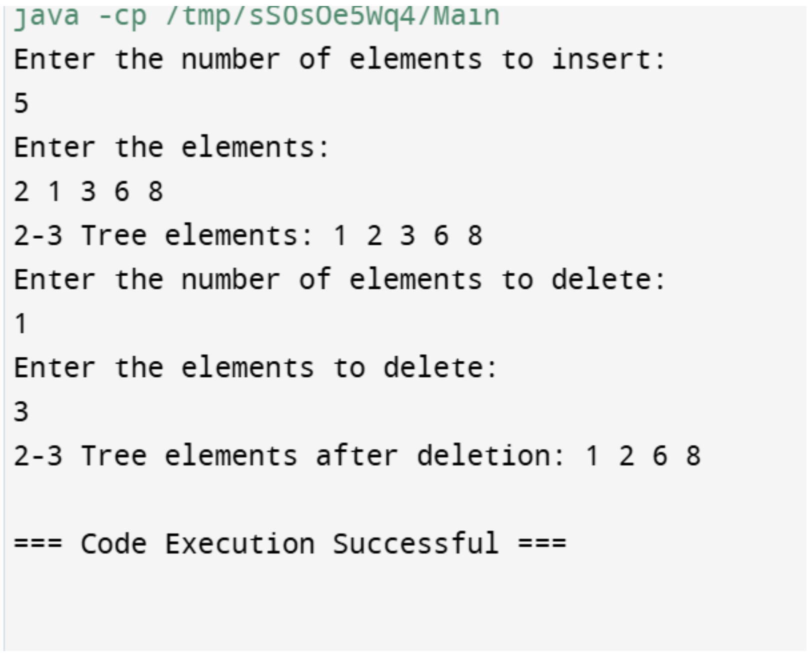
Lab Program 5:

Q) Write a program to perform insertion and deletion operations on 2-3 trees.

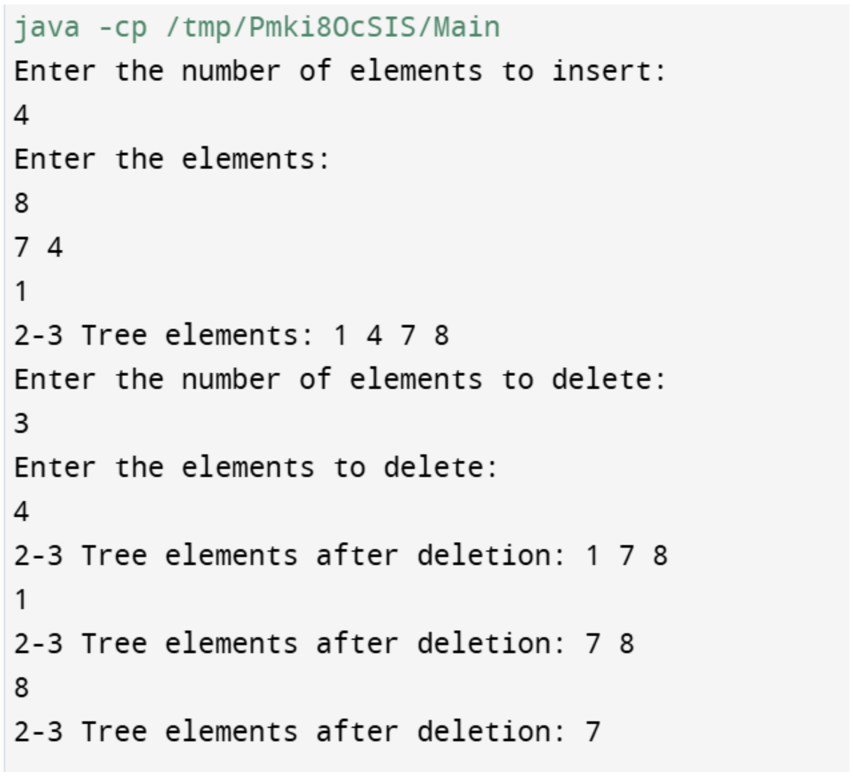
Code:

```
import java.util.*;

class Node {
    public int data1;
    public int data2;
    public Node left;
    public Node mid;
    public Node right;
    public Node parent;

    public Node(int data) {
        data1 = data;
        data2 = -1;
        left = null;
        mid = null;
        right = null;
        parent = null;
    }
}

class TwoThreeTree {
    private Node root;

    private Node insert(Node node, int data) {
        if (node == null) {
            return new Node(data);
        }

        if (node.data2 == -1) {
            if (data < node.data1) {
                node.data2 = node.data1;
                node.data1 = data;
            } else {
                node.data2 = data;
            }
            return node;
        }

        if (data < node.data1) {
            node.left = insert(node.left, data);
        } else if (data > node.data2) {
            node.right = insert(node.right, data);
        } else {
            node.mid = insert(node.mid, data);
        }

        return node;
    }

    private Node remove(Node node, int data) {
        if (node == null) {
            return null;
        }
        if (node.data1 == data && node.data2 == -1) {
            return null;
        }

        if (node.data1 == data && node.data2 != -1) {
            node.data1 = node.data2;
            node.data2 = -1;
            return node;
        }

        if (node.data2 == data && node.mid == null) {
            node.data2 = -1;
            return node;
        }

        if (data < node.data1) {
            node.left = remove(node.left, data);
        } else if ((data < node.data2 && data > node.data1) || (node.data1 == data && node.mid
!= null)) {
            node.mid = remove(node.mid, data);
        } else {
            node.right = remove(node.right, data);
        }

        return node;
    }

    private void traverse(Node node) {
        if (node != null) {
            traverse(node.left);
            System.out.print(node.data1 + " ");
            if (node.data2 != -1) {
                System.out.print(node.data2 + " ");
            }
            traverse(node.mid);
            traverse(node.right);
        }
    }

    public TwoThreeTree() {
        root = null;
    }

    public void insert(int data) {
        root = insert(root, data);
    }

    public void remove(int data) {
        root = remove(root, data);
    }

    public void display() {
        traverse(root);
    }
}

public class Main {
    public static void main(String[] args) {
        TwoThreeTree tree = new TwoThreeTree();
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements to insert:");
        int n = sc.nextInt();

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            tree.insert(sc.nextInt());
        }

        System.out.print("2-3 Tree elements: ");
        tree.display();
        System.out.println();

        System.out.println("Enter the number of elements to delete:");
        int m = sc.nextInt();

        System.out.println("Enter the elements to delete:");
        for (int i = 0; i < m; i++) {
            tree.remove(sc.nextInt());
            System.out.print("2-3 Tree elements after deletion: ");
            tree.display();
            System.out.println();
        }

        sc.close();
    }
}
```

Output:



```
java -cp /tmp/sS0s0e5Wq4/Main
Enter the number of elements to insert:
5
Enter the elements:
2 1 3 6 8
2-3 Tree elements: 1 2 3 6 8
Enter the number of elements to delete:
1
Enter the elements to delete:
3
2-3 Tree elements after deletion: 1 2 6 8

=== Code Execution Successful ===
```



```
java -cp /tmp/Pmki8OcSIS/Main
Enter the number of elements to insert:
4
Enter the elements:
8
7 4
1
2-3 Tree elements: 1 4 7 8
Enter the number of elements to delete:
3
Enter the elements to delete:
4
2-3 Tree elements after deletion: 1 7 8
1
2-3 Tree elements after deletion: 7 8
8
2-3 Tree elements after deletion: 7
```

Lab Program 6:

Q) Write a program to implement insertion operation on a red black tree. During insertion, appropriately show how recolouring or rotation operation is used.

Code:

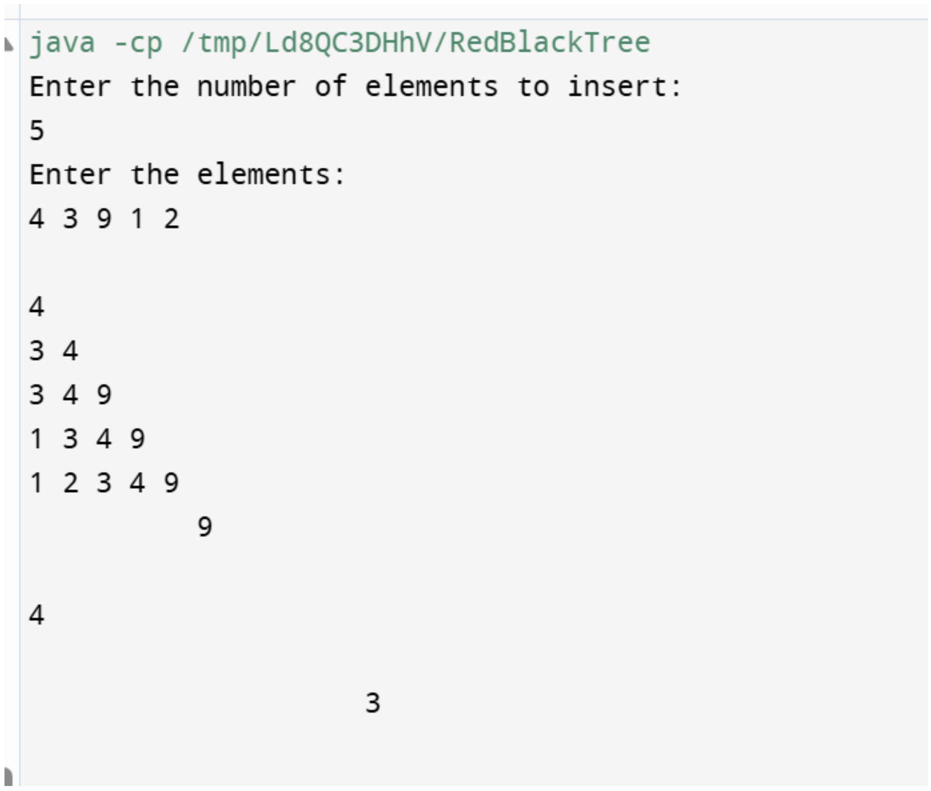
```
import java.io.*;
import java.util.Scanner;

public class RedBlackTree {
    public Node root;

    public RedBlackTree() {
        root = null;
    }

    class Node {
        int data;
        Node left;
        Node right;
        char colour;
        Node parent;

        Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
            this.colour = 'R';
            this.parent = null;
        }
    }
```

```
Node rotateLeft(Node node) {
    Node x = node.right;
    Node y = x.left;
    x.left = node;
    node.right = y;
    node.parent = x;
    if (y != null)
        y.parent = node;
    return x;
}

Node rotateRight(Node node) {
    Node x = node.left;
    Node y = x.right;
    x.right = node;
    node.left = y;
    node.parent = x;
    if (y != null)
        y.parent = node;
    return x;
}

boolean ll = false;
boolean rr = false;
boolean lr = false;
boolean rl = false;

Node insertHelp(Node root, int data) {
```

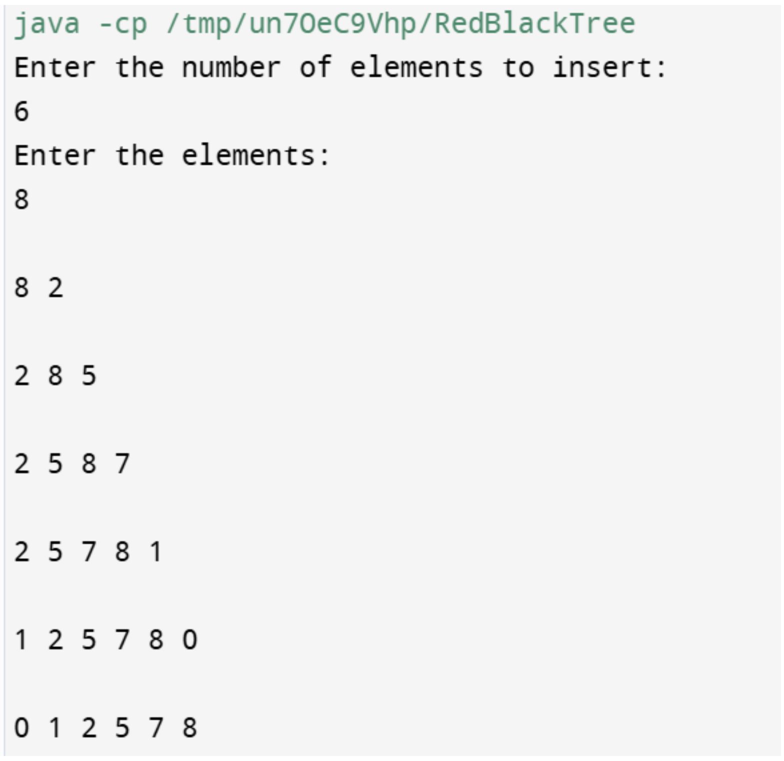
```
boolean f = false;
if (root == null)
    return new Node(data);
else if (data < root.data) {
    root.left = insertHelp(root.left, data);
    root.left.parent = root;
    if (root != this.root) {
        if (root.colour == 'R' && root.left.colour == 'R')
            f = true;
    }
} else {
    root.right = insertHelp(root.right, data);
    root.right.parent = root;
    if (root != this.root) {
        if (root.colour == 'R' && root.right.colour == 'R')
            f = true;
    }
}

if (this.ll) {
    root = rotateLeft(root);
    root.colour = 'B';
    root.left.colour = 'R';
    this.ll = false;
} else if (this.rr) {
    root = rotateRight(root);
    root.colour = 'B';
    root.right.colour = 'R';
    this.rr = false;
```

```
} else if (this.rl) {
    root.right = rotateRight(root.right);
    root.right.parent = root;
    root = rotateLeft(root);
    root.colour = 'B';
    root.left.colour = 'R';
    this.rl = false;
} else if (this.lr) {
    root.left = rotateLeft(root.left);
    root.left.parent = root;
    root = rotateRight(root);
    root.colour = 'B';
    root.right.colour = 'R';
    this.lr = false;
}

if (f) {
    if (root.parent.right == root) {
        if (root.parent.left == null || root.parent.left.colour == 'B') {
            if (root.left != null && root.left.colour == 'R')
                this.rl = true;
            else if (root.right != null && root.right.colour == 'R')
                this.ll = true;
        } else {
            root.parent.left.colour = 'B';
            root.colour = 'B';
            if (root.parent != this.root)
                root.parent.colour = 'R';
        }
        } else {
            if (root.parent.right == null || root.parent.right.colour == 'B') {
                if (root.left != null && root.left.colour == 'R')
                    this.rr = true;
                else if (root.right != null && root.right.colour == 'R')
                    this.lr = true;
            } else {
                root.parent.right.colour = 'B';
                root.colour = 'B';
                if (root.parent != this.root)
                    root.parent.colour = 'R';
            }
        }
        f = false;
    }

    return root;
}

public void insert(int data) {
    if (this.root == null) {
        this.root = new Node(data);
        this.root.colour = 'B';
    } else {
        this.root = insertHelp(this.root, data);
    }
}

void inorderTraversalHelper(Node node) {
    if (node != null) {
        inorderTraversalHelper(node.left);
        System.out.printf("%d ", node.data);
        inorderTraversalHelper(node.right);
    }
}

public void inorderTraversal() {
    inorderTraversalHelper(this.root);
}

void printTreeHelper(Node root, int space) {
    int i;
    if (root != null) {
        space = space + 10;
        printTreeHelper(root.right, space);
        System.out.printf("\n");
        for (i = 10; i < space; i++) {
            System.out.printf(" ");
        }
        System.out.printf("%d", root.data);
        System.out.printf("\n");
        printTreeHelper(root.left, space);
    }
}

public void printTree() {
    printTreeHelper(this.root, 0);
}
public static void main(String[] args) {
    RedBlackTree t = new RedBlackTree();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of elements to insert:");
    int n = sc.nextInt();
    System.out.println("Enter the elements:");
    for (int i = 0; i < n; i++) {
        t.insert(sc.nextInt());
        System.out.println();
        t.inorderTraversal();
    }
    t.printTree();
    sc.close();
}
}
```

Output:



```
java -cp /tmp/Ld8QC3DHhV/RedBlackTree
Enter the number of elements to insert:
5
Enter the elements:
4 3 9 1 2

4
3 4
3 4 9
1 3 4 9
1 2 3 4 9
          9


4


                    3

```



```
java -cp /tmp/un7OeC9Vhp/RedBlackTree
Enter the number of elements to insert:
6
Enter the elements:
8

8 2

2 8 5

2 5 8 7

2 5 7 8 1

1 2 5 7 8 0

0 1 2 5 7 8
```

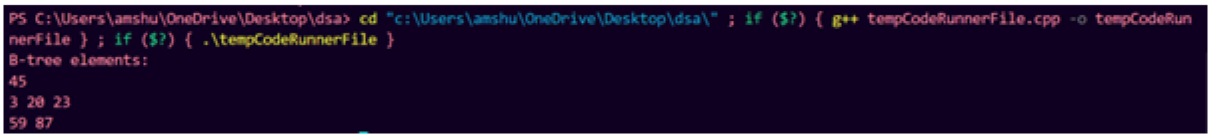
Lab Program 7:

Q) Write a program to implement insertion operation on a B-tree.

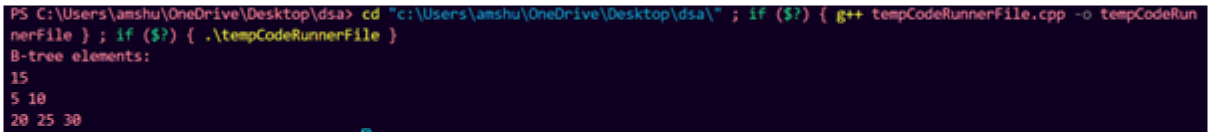
Code:

```
import java.util.Scanner;

class BTreeNode {
    int[] keys;
    int t;
    BTreeNode[] C;
    int n;
    boolean leaf;

    public BTreeNode(int t, boolean leaf) {
        this.keys = new int[2 * t - 1];
        this.t = t;
        this.C = new BTreeNode[2 * t];
        this.n = 0;
        this.leaf = leaf;
    }

    void insertNonFull(int k) {
        int i = n - 1;
        if (leaf) {
            while (i >= 0 && keys[i] > k) {
                keys[i + 1] = keys[i];
                i--;
            }
            keys[i + 1] = k;
            n++;
        } else {
            while (i >= 0 && keys[i] > k) {
                i--;
            }
            if (C[i + 1].n == 2 * t - 1) {
                splitChild(i + 1, C[i + 1]);
                if (keys[i + 1] < k) {
                    i++;
                }
            }
            C[i + 1].insertNonFull(k);
        }
    }

    void splitChild(int i, BTreeNode y) {
        BTreeNode z = new BTreeNode(y.t, y.leaf);
        z.n = t - 1;
        for (int j = 0; j < t - 1; j++) {
            z.keys[j] = y.keys[j + t];
        }
        if (!y.leaf) {
            for (int j = 0; j < t; j++) {
                z.C[j] = y.C[j + t];
            }
        }
        y.n = t - 1;
        for (int j = n; j >= i + 1; j--) {
            C[j + 1] = C[j];
        }
        C[i + 1] = z;
        for (int j = n - 1; j >= i; j--) {
            keys[j + 1] = keys[j];
        }
        keys[i] = y.keys[t - 1];
        n++;
    }

    void traverse() {
        int i;
        for (i = 0; i < n; i++) {
            if (!leaf) {
                C[i].traverse();
            }
            System.out.print(" " + keys[i]);
        }
        if (!leaf) {
            C[i].traverse();
        }
    }

    BTreeNode search(int k) {
        int i = 0;
        while (i < n && k > keys[i]) {
            i++;
        }
        if (i < n && k == keys[i]) {
            return this;
        }
        if (leaf) {
            return null;
        }
        return C[i].search(k);
    }
}

class BTree {
    BTreeNode root;
    int t;

    public BTree(int t) {
        this.root = null;
        this.t = t;
    }

    void traverse() {
        if (root != null) {
            root.traverse();
        }
    }

    BTreeNode search(int k) {
        return root == null ? null : root.search(k);
    }

    void insert(int k) {
        if (root == null) {
            root = new BTreeNode(t, true);
            root.keys[0] = k;
            root.n = 1;
        } else {
            if (root.n == 2 * t - 1) {
                BTreeNode s = new BTreeNode(t, false);
                s.C[0] = root;
                s.splitChild(0, root);
                int i = 0;
                if (s.keys[0] < k) {
                    i++;
                }
                s.C[i].insertNonFull(k);
                root = s;
            } else {
                root.insertNonFull(k);
            }
        }
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter minimum degree: ");
        int degree = scanner.nextInt();

        BTree t = new BTree(degree);

        System.out.print("Enter number of keys to insert: ");
        int numKeys = scanner.nextInt();

        System.out.println("Enter the keys:");
        for (int i = 0; i < numKeys; i++) {
            int key = scanner.nextInt();
            t.insert(key);
        }

        System.out.print("Traversal of the constructed tree is ");
        t.traverse();
        System.out.println();

        System.out.print("Enter key to search: ");
        int key = scanner.nextInt();
        if (t.search(key) != null) {
            System.out.println(" | Present");
        } else {
            System.out.println(" | Not Present");
        }
    }
}
```

Output:



```
PS C:\Users\amshu\OneDrive\Desktop\dsa> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRun
nerFile } ; if ($?) { .\tempCodeRunnerFile }
B-tree elements:
45
3 20 23
59 87
```



```
PS C:\Users\amshu\OneDrive\Desktop\dsa> cd "c:\Users\amshu\OneDrive\Desktop\dsa\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRun
nerFile } ; if ($?) { .\tempCodeRunnerFile }
B-tree elements:
15
5 10
20 25 30
```

Lab Program 8:

Q) Write a program to to implement functions of Dictionary using Hashing.

Code:

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class Dictionary {
    private static final int MAX = 10;
    Node[] root;
    Node[] temp;

    public Dictionary() {
        root = new Node[MAX];
        temp = new Node[MAX];
        for (int i = 0; i < MAX; i++) {
            root[i] = null;
            temp[i] = null;
        }
    }
```

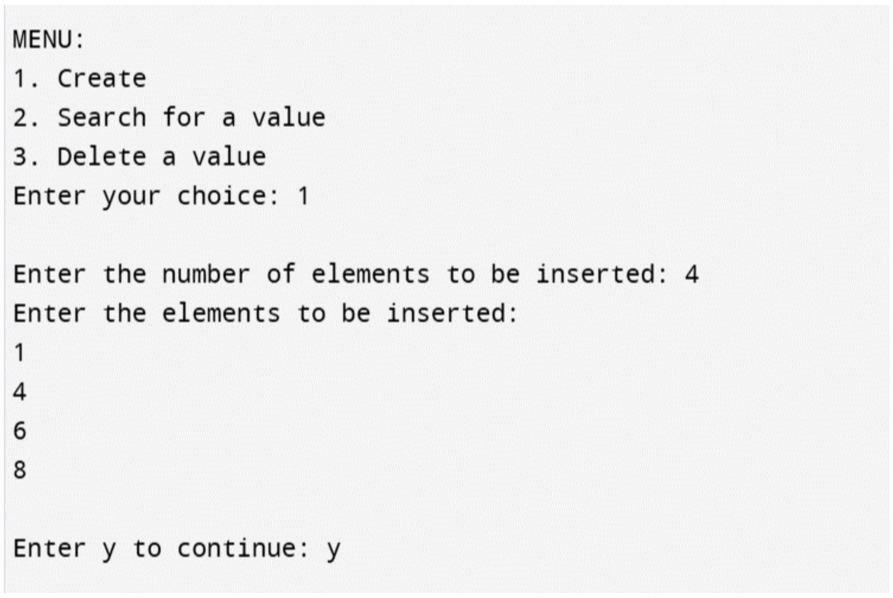
```
public void insert(int key) {
    int index = key % MAX;
    Node newNode = new Node(key);

    if (root[index] == null) {
        root[index] = newNode;
    } else {
        Node tempNode = root[index];
        while (tempNode.next != null) {
            tempNode = tempNode.next;
        }
        tempNode.next = newNode;
    }
}

public void search(int key) {
    int index = key % MAX;
    Node tempNode = root[index];
    boolean flag = false;

    while (tempNode != null) {
        if (tempNode.data == key) {
            System.out.println("\nSearch key is found!!");
            flag = true;
            break;
        } else {
            tempNode = tempNode.next;
        }
    }
```

```
        if (!flag) {
            System.out.println("\nSearch key not found.......");
        }
    }

    public void delete(int key) {
        int index = key % MAX;
        Node tempNode = root[index];
        Node prevNode = null;

        while (tempNode != null && tempNode.data != key) {
            prevNode = tempNode;
            tempNode = tempNode.next;
        }

        if (tempNode == null) {
            System.out.println("\nKey not found.");
            return;
        }

        if (prevNode == null) {
            root[index] = tempNode.next;
        } else {
            prevNode.next = tempNode.next;
        }

        System.out.println("\n" + key + " has been deleted.");
    }
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Dictionary dictionary = new Dictionary();
    char c;

    do {
        System.out.println("\nMENU:\n1. Create");
        System.out.println("2. Search for a value\n3. Delete a value");
        System.out.print("Enter your choice: ");
        int ch = scanner.nextInt();

        switch (ch) {
            case 1:
                System.out.print("\nEnter the number of elements to be inserted: ");
                int n = scanner.nextInt();
                System.out.println("Enter the elements to be inserted:");
                for (int i = 0; i < n; i++) {
                    int num = scanner.nextInt();
                    dictionary.insert(num);
                }
                break;
            case 2:
                System.out.print("\nEnter the element to be searched: ");
                int searchKey = scanner.nextInt();
                dictionary.search(searchKey);
                break;
            case 3:
                System.out.print("\nEnter the element to be deleted: ");
```

```
                    int deleteKey = scanner.nextInt();
                    dictionary.delete(deleteKey);
                    break;
                default:
                    System.out.println("\nInvalid Choice.");
                    break;
            }

            System.out.print("\nEnter y to continue: ");
            c = scanner.next().charAt(0);
        } while (c == 'y' || c == 'Y');
    }
}
```

Output:



```
MENU:
1. Create
2. Search for a value
3. Delete a value
Enter your choice: 1

Enter the number of elements to be inserted: 4
Enter the elements to be inserted:
1
4
6
8

Enter y to continue: y
```

```
MENU:
1. Create
2. Search for a value
3. Delete a value
Enter your choice: 2

Enter the element to be searched: 7

Search key not found.......

Enter y to continue: y
```

Lab Program 9&10:

Q) Write a program to implement the following functions on a Binomial heap:

1. insert(H, k): Inserts a key ‘k’ to Binomial Heap ‘H’. This operation first creates a Binomial Heap with single key ‘k’, then calls union on H and the new Binomial heap.
2. getMin(H): A simple way to getMin() is to traverse the list of root of Binomial Trees and return the minimum key.
3. extractMin(H): This operation also uses union(). We first call getMin() to find the minimum key Binomial Tree, then we remove the node and create a new Binomial Heap by connecting all subtrees of the removed minimum node. Finally we call union() on H and the newly created Binomial Heap.
4. delete(H): Like Binary Heap, delete operation first reduces the key to minus infinite, then calls extractMin().
5. decreaseKey(H): decreaseKey() is also similar to Binary Heap. We compare the decreases key with it parent and if parent’s key is more, we swap keys and recur for parent. We stop when we either reach a node whose parent has smaller key or we hit the root node.

Code:

```
import java.util.Scanner;

class BinomialHeapNode {
    int key, degree;
    BinomialHeapNode parent;
    BinomialHeapNode sibling;
    BinomialHeapNode child;

    public BinomialHeapNode(int k) {
        key = k;
        degree = 0;
        parent = null;
        sibling = null;
        child = null;
    }

    public BinomialHeapNode reverse(BinomialHeapNode sibl) {
        BinomialHeapNode ret;
        if (sibling != null)
            ret = sibling.reverse(this);
        else
            ret = this;
        sibling = sibl;
        return ret;
    }

    public BinomialHeapNode findMinNode() {
        BinomialHeapNode x = this, y = this;
        int min = x.key;

        while (x != null) {
            if (x.key < min) {
                y = x;
                min = x.key;
            }
            x = x.sibling;
        }
        return y;
    }
    public BinomialHeapNode findANodeWithKey(int value) {
        BinomialHeapNode temp = this, node = null;

        while (temp != null) {
            if (temp.key == value) {
                node = temp;
                break;
            }
            if (temp.child == null)
                temp = temp.sibling;
            else {
                node = temp.child.findANodeWithKey(value);
                if (node == null)
                    temp = temp.sibling;
                else
                    break;
            }
        }
        return node;
    }

    public int getSize() {
        return (1 + ((child == null) ? 0 : child.getSize()) + ((sibling == null) ? 0 :
sibling.getSize()));
    }
}

class BinomialHeap {
    private BinomialHeapNode Nodes;
    private int size;
```

```
public BinomialHeap() {
    Nodes = null;
    size = 0;
}

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

public int getSize() { return size; }

public void makeEmpty() {
    Nodes = null;
    size = 0;
}

public void insert(int value) {
    if (value > 0) {
        BinomialHeapNode temp = new BinomialHeapNode(value);
        if (Nodes == null) {
            Nodes = temp;
            size = 1;
        } else {
            unionNodes(temp);
            size++;
        }
    }
}

private void merge(BinomialHeapNode binHeap) {
```

```
BinomialHeapNode temp1 = Nodes, temp2 = binHeap;


while ((temp1 != null) && (temp2 != null)) {
    if (temp1.degree == temp2.degree) {
        BinomialHeapNode tmp = temp2;
        temp2 = temp2.sibling;
        tmp.sibling = temp1.sibling;
        temp1.sibling = tmp;
        temp1 = tmp.sibling;
    } else {
        if (temp1.degree < temp2.degree) {
            if ((temp1.sibling == null) || (temp1.sibling.degree > temp2.degree)) {
                BinomialHeapNode tmp = temp2;
                temp2 = temp2.sibling;
                tmp.sibling = temp1.sibling;
                temp1.sibling = tmp;
                temp1 = tmp.sibling;
            } else {
                temp1 = temp1.sibling;
            }
        } else {
            BinomialHeapNode tmp = temp1;
            temp1 = temp2;
            temp2 = temp2.sibling;
            temp1.sibling = tmp;
            if (tmp == Nodes) {
                Nodes = temp1;
            }
        }
```

```
            }
        }

        if (temp1 == null) {
            temp1 = Nodes;
            while (temp1.sibling != null) {
                temp1 = temp1.sibling;
            }
            temp1.sibling = temp2;
        }
    }

    private void unionNodes(BinomialHeapNode binHeap) {
        merge(binHeap);

        BinomialHeapNode prevTemp = null, temp = Nodes, nextTemp = Nodes.sibling;

        while (nextTemp != null) {
            if ((temp.degree != nextTemp.degree) || ((nextTemp.sibling != null) &&
(nextTemp.sibling.degree == temp.degree))) {
                prevTemp = temp;
                temp = nextTemp;
            } else {
                if (temp.key <= nextTemp.key) {
                    temp.sibling = nextTemp.sibling;
                    nextTemp.parent = temp;
                    nextTemp.sibling = temp.child;
                    temp.child = nextTemp;
                    temp.degree++;
                } else {
```

```
                if (prevTemp == null) {
                    Nodes = nextTemp;
                } else {
                    prevTemp.sibling = nextTemp;
                }
                temp.parent = nextTemp;
                temp.sibling = nextTemp.child;
                nextTemp.child = temp;
                nextTemp.degree++;
                temp = nextTemp;
            }
        }
        nextTemp = temp.sibling;
    }
}

public int findMinimum() {
    return Nodes.findMinNode().key;
}

public void delete(int value) {
    if ((Nodes != null) && (Nodes.findANodeWithKey(value) != null)) {
        decreaseKeyValue(value, findMinimum() - 1);
        extractMin();
    }
}

public void decreaseKeyValue(int old_value, int new_value) {
    BinomialHeapNode temp = Nodes.findANodeWithKey(old_value);
```

```
        if (temp == null)
            return;
        temp.key = new_value;
        BinomialHeapNode tempParent = temp.parent;

        while ((tempParent != null) && (temp.key < tempParent.key)) {
            int z = temp.key;
            temp.key = tempParent.key;
            tempParent.key = z;

            temp = tempParent;
            tempParent = tempParent.parent;
        }
    }

    public int extractMin() {
        if (Nodes == null)
            return -1;

        BinomialHeapNode temp = Nodes, prevTemp = null;
        BinomialHeapNode minNode = Nodes.findMinNode();

        while (temp.key != minNode.key) {
            prevTemp = temp;
            temp = temp.sibling;
        }

        if (prevTemp == null) {
            Nodes = temp.sibling;
```

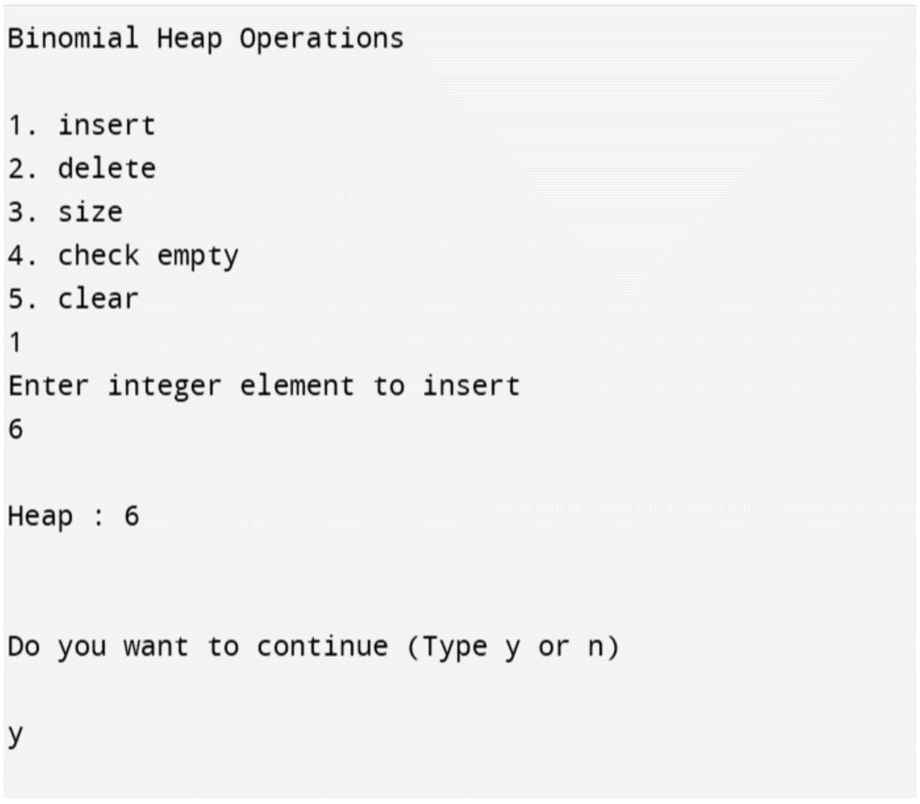
```
} else {
    prevTemp.sibling = temp.sibling;
}

temp = temp.child;
BinomialHeapNode fakeNode = temp;

while (temp != null) {
    temp.parent = null;
    temp = temp.sibling;
}

if ((Nodes == null) && (fakeNode == null)) {
    size = 0;
} else {
    if ((Nodes == null) && (fakeNode != null)) {
        Nodes = fakeNode.reverse(null);
        size = Nodes.getSize();
    } else {
        if ((Nodes != null) && (fakeNode == null)) {
            size = Nodes.getSize();
        } else {
            unionNodes(fakeNode.reverse(null));
            size = Nodes.getSize();
        }
    }
}

return minNode.key;
```

```
    }

    public void displayHeap() {
        System.out.print("\nHeap : ");
        displayHeap(Nodes);
        System.out.println("\n");
    }

    private void displayHeap(BinomialHeapNode r) {
        if (r != null) {
            displayHeap(r.child);
            System.out.print(r.key + " ");
            displayHeap(r.sibling);
        }
    }
}

public class BinomialHeapDemo {
    public static void main(String[] args) {
        BinomialHeap binHeap = new BinomialHeap();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("1. Insert");
            System.out.println("2. Delete");
            System.out.println("3. Find Minimum");
            System.out.println("4. Display Heap");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
```

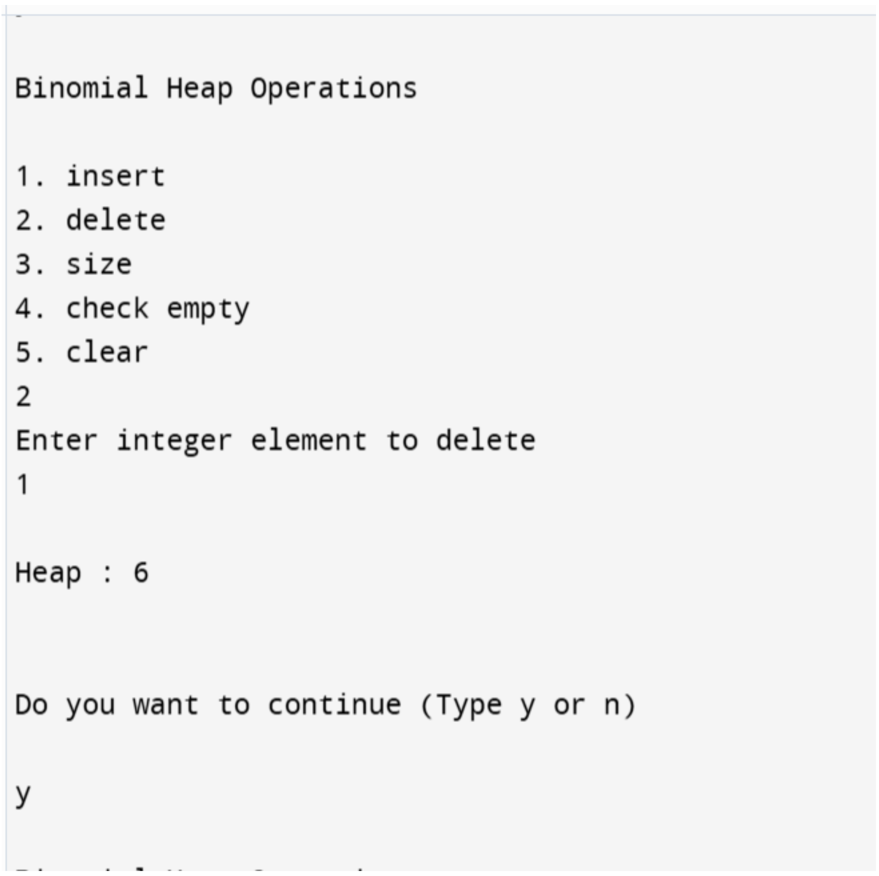
```
int choice = scanner.nextInt();

switch (choice) {
    case 1:
        System.out.print("Enter value to insert: ");
        int valueToInsert = scanner.nextInt();
        binHeap.insert(valueToInsert);
        break;
    case 2:
        System.out.print("Enter value to delete: ");
        int valueToDelete = scanner.nextInt();
        binHeap.delete(valueToDelete);
        break;
    case 3:
        int min = binHeap.findMinimum();
        if (min == -1) {
            System.out.println("Heap is empty");
        } else {
            System.out.println("Minimum value in heap: " + min);
        }
        break;
    case 4:
        binHeap.displayHeap();
        break;
    case 5:
        System.out.println("Exiting...");
        scanner.close();
        System.exit(0);
        break;
```

```
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}
```

Output:



```
Binomial Heap Operations

1. insert
2. delete
3. size
4. check empty
5. clear
1
Enter integer element to insert
6

Heap : 6


Do you want to continue (Type y or n)

y
```


```
Binomial Heap Operations

1. insert
2. delete
3. size
4. check empty
5. clear
2
Enter integer element to delete
1

Heap : 6


Do you want to continue (Type y or n)

y
```