SATYAM TRIPATHI
202151141

LAB 10

CODE

1. Min priority queue using min heap

```
import java.util.*;

public class PriorityQueue {
    static int []arr = new int[50];
    static int size = -1;


    static int parent(int i)
    {
      return (i - 1) / 2;
    }
    static int leftChild(int i)
    {
      return ((2 * i) + 1);
    }

    static int rightChild(int i)
    {
      return ((2 * i) + 2);
    }


    static void shiftUp(int i)
    {
      while (i > 0 &&arr[parent(i)] > arr[i])
      {

        swap(parent(i), i);


        i = parent(i);
      }
```

```
}


static void shiftDown(int i)
{
  int maxIndex = i;


  int l = leftChild(i);

  if (l <= size &&arr[l] < arr[maxIndex])
  {
    maxIndex = l;
  }


  int r = rightChild(i);

  if (r <= size &&arr[r] < arr[maxIndex])
  {
    maxIndex = r;
  }


  if (i != maxIndex)
  {
    swap(i, maxIndex);
    shiftDown(maxIndex);
  }
}


static void insert(int p)
{
  size = size + 1;
  arr[size] = p;


  shiftUp(size);
```

```
    }


    static int extractMax()
    {
      int result = arr[0];


      arr[0] = arr[size];
      size = size - 1;

      shiftDown(0);
      return result;
    }

    static void changePriority(int i, int p)
    {
      int oldp = arr[i];
      arr[i] = p;

      if (p < oldp)
      {
        shiftUp(i);
      }
      else
      {
        shiftDown(i);
      }
    }


    static int getMax()
    {
      return arr[0];
    }


    static void remove(int i)
    {
```

```
    arr[i] = getMax() + 1;


    shiftUp(i);


    extractMax();
  }

  static void swap(int i, int j)
  {
    int temp= arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }

  // Driver Code
  public static void main(String[] args)
  {


    insert(45);
    insert(20);
    insert(14);
    insert(12);
    insert(31);
    insert(7);
    insert(11);
    insert(13);
    insert(7);

    int i = 0;

    System.out.print("Priority Queue : ");
    while (i <= size)
    {
      System.out.print(arr[i] + " ");
      i++;
    }
```

SATYAM TRIPATHI
202151141

```
    System.out.print("\n");


    System.out.print("Node with maximum priority : " + extractMax() +
"\n");


    System.out.print("Priority queue after " + "extracting minimum : ");
    int j = 0;
    while (j <= size)
    {
      System.out.print(arr[j] + " ");
      j++;
    }

    System.out.print("\n");

    changePriority(2, 49);
    System.out.print("Priority queue after " +"priority change : ");
    int k = 0;
    while (k <= size)
    {
      System.out.print(arr[k] + " ");
      k++;
    }

    System.out.print("\n");


    remove(3);
    System.out.print("Priority queue after " + "removing the element :
");
    int l = 0;
    while (l <= size)
    {
      System.out.print(arr[l] + " ");
      l++;
    }
```

```
    }
}
```

OUTPUT

```
Priority Queue : 7 7 11 13 31 20 12 45 14
Node with maximum priority : 7
Priority queue after extracting minimum : 7 13 11 14 31 20 12 45
Priority queue after priority change : 7 13 12 14 31 20 49 45
Priority queue after removing the element : 8 13 12 45 31 20 49
```

2. Huffman coding algorithm

```
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Comparator;

class HuffmanNode {

    int data;
    char c;

    HuffmanNode left;
    HuffmanNode right;
}
class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y)
    {

        return x.data - y.data;
    }
}

public class Huffman {
    public static void printCode(HuffmanNode root, String s)
    {
        if (root.left == null && root.right == null &&
Character.isLetter(root.c))
```

SATYAM TRIPATHI
202151141

```
        {
            System.out.println(root.c + ":" + s);

            return;
        }
        printCode(root.left, s + "0");
        printCode(root.right, s + "1");
    }
    public static void main(String[] args)
    {

        Scanner s = new Scanner(System.in);
        int n = 6;
        char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
        int[] charfreq = { 5, 10, 15, 20, 25, 30 };
        PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>(n,
new MyComparator());

        for (int i = 0; i < n; i++) {
            HuffmanNode hn = new HuffmanNode();

            hn.c = charArray[i];
            hn.data = charfreq[i];

            hn.left = null;
            hn.right = null;
            q.add(hn);
        }
        HuffmanNode root = null;
        while (q.size() > 1) {
            HuffmanNode x = q.peek();
            q.poll();
            HuffmanNode y = q.peek();
            q.poll();
            HuffmanNode f = new HuffmanNode();
            f.data = x.data + y.data;
            f.c = '-';
            f.left = x;
            f.right = y;
```

```
            root = f;
            q.add(f);
        }
        printCode(root, "");
    }
}
```

OUTPUT

```
d:00
e:01
c:100
a:1010
b:1011
f:11
```

3. Binary Search Tree

```
public class BST_Implementation {

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

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

      public Node root;

      public BST_Implementation(){
          root = null;
      }
```

SATYAM TRIPATHI
202151141

```
//insertion
public void insert(int data) {
    Node newNode = new Node(data);
    if(root == null){
        root = newNode;
        return;
      }
    else {
        Node current = root, parent = null;

        while(true) {
            parent = current;
            if(data < current.data) {
                current = current.left;
                if(current == null) {
                    parent.left = newNode;
                    return;
                }
            }
            else {
                current = current.right;
                if(current == null) {
                    parent.right = newNode;
                    return;
                }
            }
        }
    }
}

//minNode
public Node minNode(Node root) {
    if (root.left != null)
        return minNode(root.left);
    else
        return root;
}

//deleteNode
```

SATYAM TRIPATHI
202151141

```
public Node deleteNode(Node node, int value) {
    if(node == null){
        return null;
     }
    else {
        if(value < node.data)
            node.left = deleteNode(node.left, value);
        else if(value > node.data)
            node.right = deleteNode(node.right, value);
        else {
            if(node.left == null && node.right == null)
                node = null;
            else if(node.left == null) {
                node = node.right;
            }
            else if(node.right == null) {
                node = node.left;
            }
            else {
                Node temp = minNode(node.right);
                node.data = temp.data;
                node.right = deleteNode(node.right, temp.data);
            }
        }
        return node;
    }
}

//inorder
public void inorderTraversal(Node node) {

    if(root == null){
        System.out.println("Tree is empty");
        return;
     }
    else {

        if(node.left!= null)
            inorderTraversal(node.left);
```

SATYAM TRIPATHI
202151141

```
            System.out.print(node.data + " ");
            if(node.right!= null)
                inorderTraversal(node.right);

        }
    }

    public static void main(String[] args) {

      BST_Implementation bt = new BST_Implementation();
        //Add nodes to the binary tree
        bt.insert(50);
        bt.insert(30);
        bt.insert(70);
        bt.insert(60);
        bt.insert(10);
        bt.insert(90);

        System.out.println("Binary search tree after insertion:");
        //Displays the binary tree
        bt.inorderTraversal(bt.root);

        Node deletedNode = null;
        //Deletes node 90 which has no child
        deletedNode = bt.deleteNode(bt.root, 90);
        System.out.println("\nBinary search tree after deleting node
90:");
        bt.inorderTraversal(bt.root);

        //Deletes node 30 which has one child
        deletedNode = bt.deleteNode(bt.root, 30);
        System.out.println("\nBinary search tree after deleting node
30:");
        bt.inorderTraversal(bt.root);

        //Deletes node 50 which has two children
        deletedNode = bt.deleteNode(bt.root, 50);
        System.out.println("\nBinary search tree after deleting node
50:");
```

```
          bt.inorderTraversal(bt.root);
      }
}
```

OUTPUT

```
Binary search tree after insertion:
10 30 50 60 70 90
Binary search tree after deleting node 90:
10 30 50 60 70
Binary search tree after deleting node 30:
10 50 60 70
Binary search tree after deleting node 50:
10 60 70
```