NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(NAAC Accredited)

(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

Pampady, Thiruvilwamala (PO), Thrissur (DT), Kerala 680 588

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

08 CS 6081 (P) ADVANCED DATA STRUCTURE LAB

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT THE DEPARTMENT

- Established in: 2002
- Course offered: B.Tech. in Computer Science and Engineering

  M. Tech. in Computer Science and Engineering

  M. Tech. in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Certified by ISO 9001-2015
- Affiliated to A P J Abdul Kalam Technological University, Kerala.

DEPARTMENT VISSION

To develop professionally ethical and socially responsible Mechatronics engineers to serve the humanity through quality professional education.

DEPARTMENT MISSION

1) The department is committed to impart the right blend of knowledge and quality education to create professionally ethical and socially responsible graduates.

2) The department is committed to impart the awareness to meet the current challenges in technology.

3) Establish state-of-the-art laboratories to promote practical knowledge of mechatronics to meet the needs of the society

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1:Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyze, design and development of novel Software Packages,

Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills,

Teamwork and leadership qualities.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

PREPARATI ON FOR THE LABORATORY SESSION

GENERAL INSTRUCTIONS TO STUDENTS

1. Read carefully and understand the description of the experiment in the lab manual. You may go to the lab at an earlier date to look at the experimental facility and understand it better. Consult the appropriate references to be completely familiar with the concepts and hardware.

2. Make sure that your observation for previous week experiment is evaluated by the faculty member and you have transferred all the contents to your record before entering to the lab/workshop.

3. At the beginning of the class, if the faculty or the instructor finds that a student is not adequately prepared, they will be marked as absent and not be allowed to perform the experiment.

4. Bring necessary material needed (writing materials, graphs, calculators, etc.) to perform the required preliminary analysis. It is a good idea to do sample calculations and as much of the analysis as possible during the session. Faculty help will be available. Errors in the procedure may thus be easily detected and rectified.

5. Please actively participate in class and don’t hesitate to ask questions. Please utilize the teaching assistants fully. To encourage you to be prepared and to read the lab manual before coming to the laboratory, unannounced questions may be asked at any time during the lab.

6. Carelessness in personal conduct or in handling equipment may result in serious injury to the individual or the equipment. Do not run near moving machinery/equipment. Always be on the alert for strange sounds. Guard against entangling clothes in moving parts of machinery.

7. Students must follow the proper dress code inside the laboratory. To protect clothing from dirt, wear a lab coat. Long hair should be tied back. Shoes covering the whole foot will have to be worn.

8. In performing the experiments, please proceed carefully to minimize any water spills, especially on the electric circuits and wire.

9. Maintain silence, order and discipline inside the lab. Don't use cell phones inside the laboratory.

10. Any injury no matter how small must be reported to the instructor immediately.

11. Check with faculty members one week before the experiment to make sure that you have the handout for that experiment and all the apparatus.

AFTER THE LABORATORY SESSION

1. Clean up your work area.

2. Check with the technician before you leave.

3. Make sure you understand what kind of report is to be prepared and due submission of record is next lab class.

4. Do sample calculations and some preliminary work to verify that the experiment was successful

MAKE-UPS AND LATE WORK

Students must participate in all laboratory exercises as scheduled. They must obtain permission from the faculty member for absence, which would be granted only under justifiable circumstances. In such an event, a student must make arrangements for a make-up laboratory, which will be scheduled when the time is available after completing one cycle. Late submission will be awarded less mark for record and internals and zero in worst cases.

LABORATORY POLICIES

1. Food, beverages & mobile phones are not allowed in the laboratory at any time.

2. Do not sit or place anything on instrument benches.

3. Organizing laboratory experiments requires the help of laboratory technicians and staff. Be punctual.

SYLLABUS

Course No.	Course Name	L-T-P	Credits	Year of Introduction
08CS 6081(P)	Advanced Data Structure Lab	0-0-2	2	2015

Course Objectives

To give the Student:-

- The ability to design and implement algorithms for various operations on advanced data structures.

Course Outcome:

A student who completes this course will get hands on experience of working with advanced data structures.

Kerala Technological University
Master of Technology – Curriculum, Syllabus & Course Plan

08 CS 6081(P) – EXPERIMENTS

Experiment No	Description
I	AVL Trees
II	Red Black Trees
III	Splay Trees
IV	Treap
V	Min-Max Heap
VI	Binomial heap
VII	Dijkstra's algorithm using Fibonacci heap
VIII	Skewed heap
IX	K-d trees

INDEX

EXP NO	EXPERIMENT NAME	PAGE NO	MARK	SIGN
1	AVL TREE	9		
2	RED BLACK TREES	19		
3	SPLAY TREES	28		
4	TREAP	39		
5	MIN-MAX HEAP	46		
6	BINOMIAL HEAP	54		
7	DIJKSTRA’S ALGORITHM USING FIBNOACCI HEAP	65		
8	SKEWED HEAP	70		
9	K-D TREES	76		

FINAL VERIFICATION BY THE FACULTY

TOTAL MARKS:

INTERNAL EXAMINER **EXTERNAL EXAMINER**

EXPERIMENT 1

AVL TREE

OBJECTIVE

To perform AVL Tree operations using java

ALGORITHM

Step 1: Start

Step 2: First, insert a new element into the tree using BST's (Binary Search Tree) insertion logic.

Step 3: After inserting the elements you have to check the Balance Factor of each node.

Step 4: When the Balance Factor of every node will be found like 0 or 1 or -1 then the algorithm will proceed for the next operation.

Step 5: When the balance factor of any node comes other than the above three values then the tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced and then the algorithm will proceed for the next operation.

Step 6: For deletion, Firstly, find that node where k is stored

Step 7: Secondly delete those contents of the node (Suppose the node is x)

Step 8: Deleting a node in an AVL tree can be reduced by deleting a leaf. When x has no children then, delete x When x has one child, let x' becomes the child of x.

a. When x has two children,
b. Then find x's successor z which has no left child.
c. then replace x's contents with z's contents, and delete z

Step 9: Search the element with the help of BST property

Step 10: Stop the program.

PROGRAM

```
import java.util.Scanner;
 class AVLNode
 {
     AVLNode left, right;
     int data;
     int height;
   public AVLNode()
{
left =null;

  right = null;
         data = 0;
         height = 0;
     }
     public AVLNode(int n)
     {
         left = null;
         right = null;
         data = n;
         height = 0;
     }}
 class AVLTree
 {
     private AVLNode root;
     public AVLTree()
     {
         root = null;
     }
     public boolean isEmpty()
     {
         return root == null;
     }
     public void makeEmpty()
     {
         root = null;
     }
      public void insert(int data)
     {
         root = insert(data, root);
     }
     private int height(AVLNode t )
     {
         return t == null ? -1 : t.height;
     }
     private int max(int lhs, int rhs)
{
         return lhs > rhs ? lhs : rhs;
     }
     private AVLNode insert(int x, AVLNode t)
{
         if (t == null)
             t = new AVLNode(x);
         else if (x < t.data)
         {
             t.left = insert( x, t.left );
             if( height( t.left ) - height( t.right ) == 2 )
                 if( x < t.left.data )
                     t = rotateWithLeftChild( t );
                 else
                     t = doubleWithLeftChild( t );
}
         else if( x > t.data )
         {
             t.right = insert( x, t.right );
             if( height( t.right ) - height( t.left ) == 2 )
                 if( x > t.right.data)
                     t = rotateWithRightChild( t );
                 else
                     t = doubleWithRightChild( t );
         }
         else
           height = max( height( t.left ), height( t.right ) ) + 1;
         return t;
     }
     private AVLNode rotateWithLeftChild(AVLNode k2)
     {
         AVLNode k1 = k2.left;
         k2.left = k1.right;
         k1.right = k2;
         k2.height = max( height( k2.left ), height( k2.right ) ) + 1;
         k1.height = max( height( k1.left ), k2.height ) + 1;
         return k1;
     }
      private AVLNode rotateWithRightChild(AVLNode k1)
     {
         AVLNode k2 = k1.right;
```

```
         k1.right = k2.left;
         k2.left = k1;
         k1.height = max( height( k1.left ), height( k1.right ) ) + 1;
         k2.height = max( height( k2.right ), k1.height ) + 1;
return k2;
     }
     private AVLNode doubleWithLeftChild(AVLNode k3)
     {
         k3.left = rotateWithRightChild( k3.left );
         return rotateWithLeftChild( k3 );
     }
     private AVLNode doubleWithRightChild(AVLNode k1)
     {
  k1.right = rotateWithLeftChild( k1.right );
         return rotateWithRightChild( k1 );
     }
     public int countNodes()
     {
         return countNodes(root);
  }
     private int countNodes(AVLNode r)
     {
         if (r == null)
             return 0;

   else
         {
             int l = 1;
             l += countNodes(r.left);
             l += countNodes(r.right);
```

```
            return l;
        } }
    public boolean search(int val)
    {
        return search(root, val);
    }
    private boolean search(AVLNode r, int val)
    {
        boolean found = false;
        while ((r != null) && !found)
        {
            int rval = r.data;
            if (val < rval)
                r = r.left;
            else if (val > rval)
                r = r.right;
            else
            {
                found = true;
                break;
  }
            found = search(r, val);
        }
        return found;
    }
    public void inorder()
    {
        inorder(root);
    }
    private void inorder(AVLNode r)
    {
       if (r != null)
{
           inorder(r.left);
System.out.print(r.data +" ");
           inorder(r.right);
       }}
     public void preorder()
    {
       preorder(root);
    }
    private void preorder(AVLNode r)
    {
       if (r != null)
       {

     System.out.print(r.data +" ");
           preorder(r.left);
           preorder(r.right);
       }}

    public void postorder()
    {
       postorder(root);
    }
    private void postorder(AVLNode r)
    {
       if (r != null)
       {
           postorder(r.left);
```

```
            postorder(r.right);
            System.out.print(r.data +" ");
        }}}

public class AVLTreeTest
{
 public static void main(String[] args)
   {
       Scanner scan = new Scanner(System.in);
       AVLTree avlt = new AVLTree();
       System.out.println("AVLTree Tree Test\n");
       char ch;
 do
       {
           System.out.println("\nAVLTree Operations\n");
           System.out.println("1. insert ");
           System.out.println("2. search");
           System.out.println("3. count nodes");
           System.out.println("4. check empty");
           System.out.println("5. clear tree");

           int choice = scan.nextInt();
           switch (choice)
           {
           case 1 :
          System.out.println("Enter integer element to insert");
               avlt.insert( scan.nextInt() );
               break;
           case 2 :
               System.out.println("Enter integer element to search");
               System.out.println("Search result : "+ avlt.search( scan.nextInt() ));
               break;
           case 3 :
               System.out.println("Nodes = "+ avlt.countNodes());
               break;
           case 4 :
```

AVL Tree

```
       System.out.println("Empty status = "+ avlt.isEmpty());
            break;
        case 5 :
            System.out.println("\nTree Cleared");
            avlt.makeEmpty();
            break;
        default :
            System.out.println("Wrong Entry \n ");
            break;
        }
        System.out.print("\nPost order : ");
        avlt.postorder();
        System.out.print("\nPre order : ");
        avlt.preorder();
        System.out.print("\nIn order : ");
        avlt.inorder();
        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
    }
  while (ch == 'Y'|| ch == 'y');

    }}}
```

RESULT

The program to perform AVL Tree operations using java has been completed successfully and output was verified.

OUTPUT

1. **Create**
2. **Insert**
3. **Delete**
4. **Print**
5. **Quit**

   Enter your choice

   1

   Enter number of elements 8

   Enter the tree data 12

   8

   4

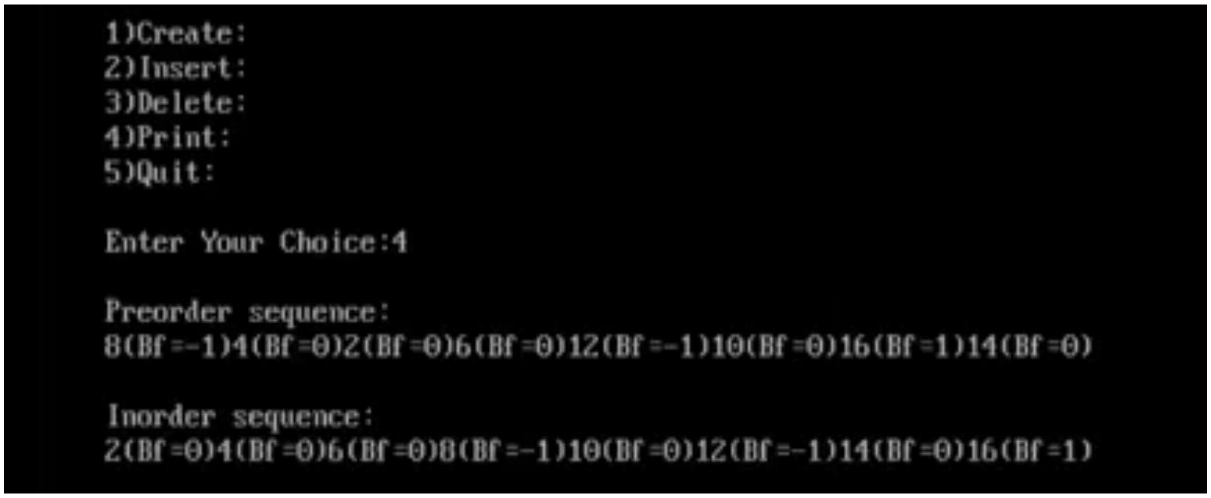
   2

   6

   10

   16

   14



```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

Preorder sequence:
8(Bf=-1)4(Bf=0)2(Bf=0)6(Bf=0)12(Bf=-1)10(Bf=0)16(Bf=1)14(Bf=0)

Inorder sequence:
2(Bf=0)4(Bf=0)6(Bf=0)8(Bf=-1)10(Bf=0)12(Bf=-1)14(Bf=0)16(Bf=1)
```

EXPERIMENT – 2

RED BLACK TREES

AIM
To perform Red Black Trees in java

ALGORITHM

Step 1: Start

Step 2: Import the necessary packages.

Step 3: Check whether tree is Empty.

Step 4: If tree is Empty then insert the newNode as Root node with color Black and exit from the operation.

step 5: If tree is not Empty then insert the newNode as a leaf node with Red color.

Step 6: If the parent of newNode is Black then exit from the operation.

Step 7: If the parent of newNode is Red then check the color of parent node's sibling of newNode.

Step 8: If it is Black or NULL node then make a suitable Rotation and Recolor it.

Step 9: If it is Red colored node then perform Recolor and Recheck it. Repeat the same until tree becomes Red Black Tree.

Step 10: The deletion operation is similar to deletion operation in BST. But after every deletion operation we need to check with the Red Black Tree properties.

PROGRAM

```
import java.util.Scanner;
class RedBlackNode
{
RedBlackNode left, right;
int element;
int color;
public RedBlackNode(int theElement)
{
this( theElement, null, null );
}
public RedBlackNode(int theElement, RedBlackNode lt, RedBlackNode rt)
{
left = lt;
right = rt;
element = theElement;
color = 1;
}}
class RBTree
{
private RedBlackNode current;
private RedBlackNode parent;
private RedBlackNode grand;
private RedBlackNode great;
private RedBlackNode header;
private static RedBlackNode nullNode;
static
{
nullNode = new RedBlackNode(0);
nullNode.left = nullNode;
nullNode.right = nullNode;
}
static final int BLACK = 1;
static final int RED   = 0;
public RBTree(int negInf)
```

```
{
header = new RedBlackNode(negInf);
header.left = nullNode;
header.right = nullNode;
}
public boolean isEmpty()
{
return header.right == nullNode;
}
public void makeEmpty()
{
header.right = nullNode;
}
public void insert(int item )
{
current = parent = grand = header;
nullNode.element = item;
while (current.element != item)
{
great = grand;
grand = parent;
parent = current;
current = item < current.element ? current.left : current.right;

if (current.left.color == RED && current.right.color == RED)
handleReorient( item );
}
return;
current = new RedBlackNode(item, nullNode, nullNode);
if (item < parent.element)
parent.left = current;
else
parent.right = current;
handleReorient( item );
}
```

```
private void handleReorient(int item)
{
if (parent.color == RED)
{
if (item < grand.element != item < parent.element) parent = rotate( item, grand );
current = rotate(item, great ); current.color = BLACK;
}}
private RedBlackNode rotate(int item, RedBlackNode parent)
{
if(item < parent.element)
return parent.left = item < parent.left.element ? rotateWithLeftChild(parent.left) :
rotateWithRightChild(parent.left) ;
else
return parent.right = item < parent.right.element ? rotateWithLeftChild(parent.right)
rotateWithRightChild(parent.right);
}
private RedBlackNode rotateWithLeftChild(RedBlackNode k2)
{
RedBlackNode k1 = k2.left;
k2.left = k1.right;
k1.right = k2;
return k1;
}
private RedBlackNode rotateWithRightChild(RedBlackNode k1)
{
RedBlackNode k2 = k1.right;
k1.right = k2.left;
k2.left = k1;
return k2;
}
public int countNodes()
{
return countNodes(header.right);
}
private int countNodes(RedBlackNode r)
```

```
{
if (r == nullNode)
return 0;
else
{
int l = 1;
l += countNodes(r.left);
l += countNodes(r.right);
return l;
}
}

public boolean search(int val)
{
return search(header.right, val);
}
private boolean search(RedBlackNode r, int val)
{
boolean found = false;
while ((r != nullNode) && !found)
{
int rval = r.element;
if (val < rval)
r = r.left;
else if (val > rval)
r = r.right;
else
{
found = true;
break;
}
found = search(r, val);
}
return found;
}
```

```
public void inorder()
{
inorder(header.right);
}
private void inorder(RedBlackNode r)
{
if (r != nullNode)
{
inorder(r.left);
char c = 'B';
if (r.color == 0)
c = 'R';
System.out.print(r.element +""+c+" ");
inorder(r.right);
}
}
public void preorder()
{
preorder(header.right);
}
private void preorder(RedBlackNode r)
{
if (r != nullNode)
{
char c = 'B';
if (r.color == 0)
c = 'R';
System.out.print(r.element +""+c+" ");
preorder(r.left);
preorder(r.right);
}
}
public void postorder()
{
postorder(header.right);
```

```
}
private void postorder(RedBlackNode r)
{
if (r != nullNode)
{
postorder(r.left);
postorder(r.right);
char c = 'B';
if (r.color == 0)
c = 'R';
System.out.print(r.element +""+c+" ");
}}}
public class RedBlackTreeTest
{
public static void main(String[] args)
{Scanner scan = new Scanner(System.in);
RBTree rbt = new RBTree(Integer.MIN_VALUE); System.out.println("Red Black Tree Test\n"); char ch;
do
{
System.out.println("\nRed Black Tree Operations\n");
System.out.println("1. insert ");
System.out.println("2. Search
System.out.println("3. count nodes");
System.out.println("4. check empty");
 System.out.println("5. clear tree");
int choice = scan.nextInt();
switch (choice)
{case 1 :
System.out.println("Enter integer element to insert");
rbt.insert( scan.nextInt() );
break;
case 2 :
System.out.println("Enter integer element to search"); System.out.println("Search result : "+ rbt.search(
scan.nextInt() )); break;
case 3 :
```

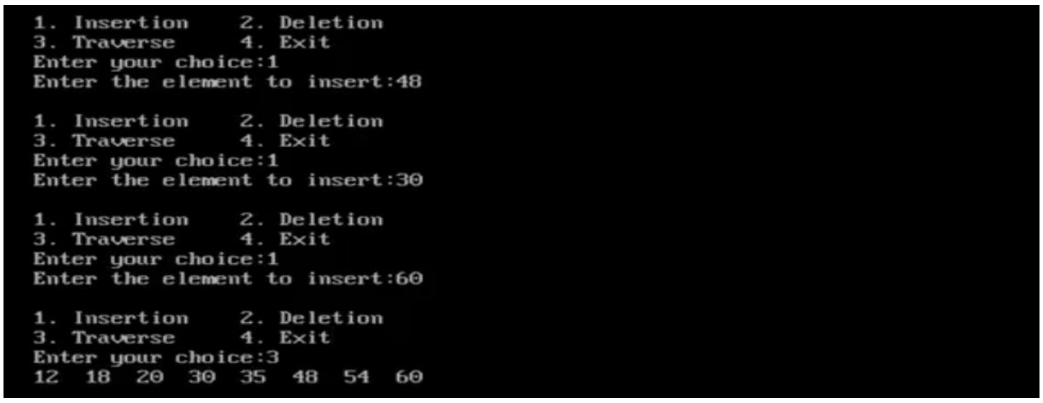
```
System.out.println("Nodes = "+ rbt.countNodes()); break;
case 4 :
System.out.println("Empty status = "+ rbt.isEmpty()); break;
case 5 :
System.out.println("\nTree Cleared");



rbt.makeEmpty();
break;
default :
System.out.println("Wrong Entry \n ");
break;
}
System.out.print("\nPost order : ");
rbt.postorder();
System.out.print("\nPre order : ");
rbt.preorder();
System.out.print("\nIn order : ");
rbt.inorder();
System.out.println("\nDo you want to continue (Type y or n) \n"); ch = scan.next().charAt(0);
} while (ch == 'Y'|| ch == 'y');
}}
```

RESULT

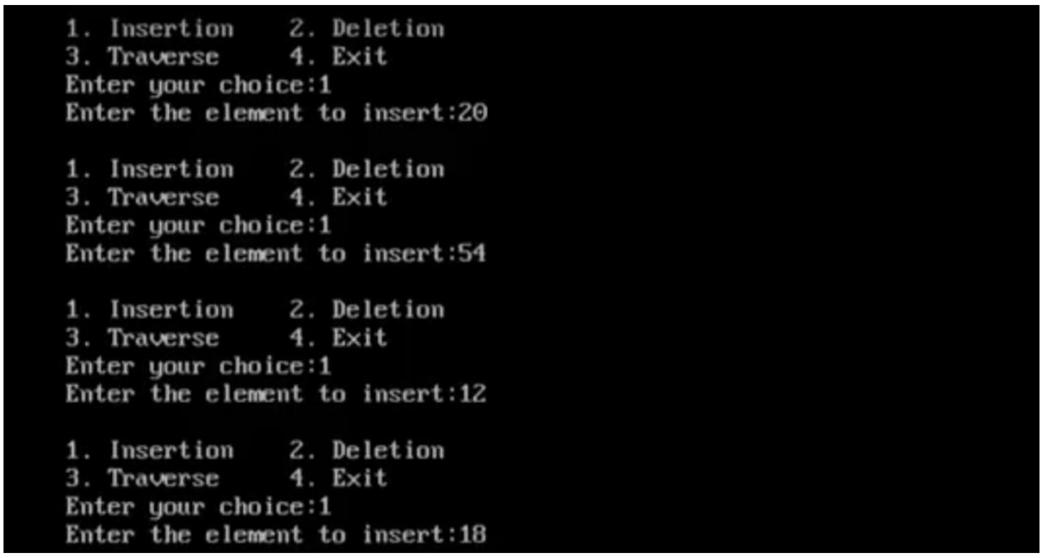
The program to perform Red Black Tees using java has been completed successfully and output was verified.

OUTPUT

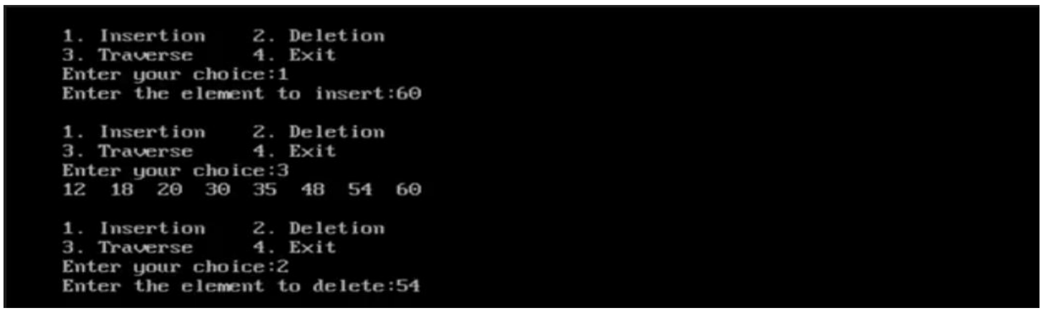
OUTPUT



```
1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:48

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:30

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:60

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:3
12  18  20  30  35  48  54  60
```



```
1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:20

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:54

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:12

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:18
```



```
1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:60

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:3
12  18  20  30  35  48  54  60

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:2
Enter the element to delete:54
```

EXPERIMENT – 3

SPLAY TREE

AIM

To implement Splay Tree using java.

ALGORITHM

Step1: Start

Step2: Import necessary packages

Step3: Insert x as with a normal binary search tree.

Step4: When an item is inserted, a splay is performed.

Step5: As a result, the newly inserted node x becomes the root of the tree.

Step6: If Root is NULL: We simply return the root.

Step 7: Else Splay the given key k. If k is present, then it becomes the new root. If not

present,then last accessed leaf node becomes the new root.

Step 8: If new root's key is not same as k, then return the root as k is not present.

Step 9: Else the key k is present.

Step 10: Split the tree into two trees Tree1 = root's left subtree and Tree2 = root's

right subtree and delete the root node.

Step 11: Let the root's of Tree1 and Tree2 be Root1 and Root2respectively.

Step 12: If Root1 is NULL: Return Root2.

Else, Splay the maximum node (node having the maximum value of Tree1.

Step 13: After the Splay procedure, make Root2 as the right child of Root1 and

return Root1.

Step 14: Stop

PROGRAM

```
import java.util.Scanner;
 class SplayNode{
    SplayNode left, right, parent;
    int element;

    public SplayNode()
    {
        this(0, null, null, null);
    }
    public SplayNode(int ele)
    {
        this(ele, null, null, null)
     }
     public SplayNode(int ele, SplayNode left, SplayNode right, SplayNode parent)
    {
        this.left = left;
        this.right = right;
        this.parent = parent;
        this.element = ele;
    }
 }

  class SplayTree
 {
    private SplayNode root;
    private int count = 0;

    public SplayTree()
    {
        root = null;
    }

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

    public void clear()
    {
        root = null;
        count = 0;
    }

    public void insert(int ele)
    {
        SplayNode z = root;
        SplayNode p = null;
        while (z != null)
        {
            p = z;
      if (ele > p.element)
                z = z.right;
            else
                z = z.left;
        }
        z = new SplayNode();
        z.element = ele;
        z.parent = p;
        if (p == null)
            root = z;
        else if (ele > p.element)
            p.right = z;
        else
            p.left = z;
        Splay(z);
        count++;
    }
public void makeLeftChildParent(SplayNode c, SplayNode p)
    {
        if ((c == null) || (p == null) || (p.left != c) || (c.parent != p))
```

```
            throw new RuntimeException("WRONG");

        if (p.parent != null)
        {
            if (p == p.parent.left)
                p.parent.left = c;
            else
                p.parent.right = c;
        }
        if (c.right != null)
            c.right.parent = p;

        c.parent = p.parent;
        p.parent = c;
        p.left = c.right;
        c.right = p;
    }

    public void makeRightChildParent(SplayNode c, SplayNode p)
    {
        if ((c == null) || (p == null) || (p.right != c) || (c.parent != p))
            throw new RuntimeException("WRONG");
        if (p.parent != null)
        {
        if (p == p.parent.left)
                p.parent.left = c;
            else
                p.parent.right = c;
        }
        if (c.left != null)
            c.left.parent = p;
        c.parent = p.parent;
        p.parent = c;
        p.right = c.left;
        c.left = p;
```

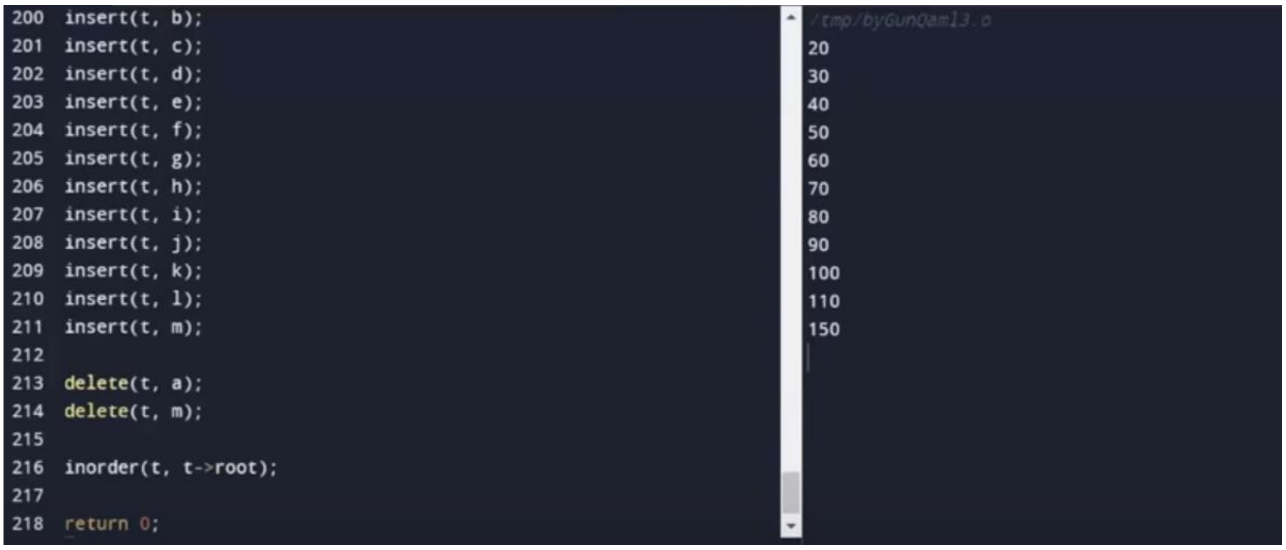
```
}

private void Splay(SplayNode x)
{
    while (x.parent != null)
    {
        SplayNode Parent = x.parent;
        SplayNode GrandParent = Parent.parent;
        if (GrandParent == null)
        {
            if (x == Parent.left)
                makeLeftChildParent(x, Parent);
            else
                makeRightChildParent(x, Parent);
        }
        else
        {
            if (x == Parent.left)
            {
                if (Parent == GrandParent.left)
                {
                    makeLeftChildParent(Parent, GrandParent);
                    makeLeftChildParent(x, Parent);
                }
                else
                {
                    makeLeftChildParent(x, x.parent);
                    makeRightChildParent(x, x.parent);
                }
            }
            else
            {
                if (Parent == GrandParent.left)
            {
                    makeRightChildParent(x, x.parent);
                    makeLeftChildParent(x, x.parent);
                }
                else
                {
                    makeRightChildParent(Parent, GrandParent);
                    makeRightChildParent(x, Parent);
                }}}
            root = x;
    }
    public void remove(int ele)
    {
        SplayNode node = findNode(ele);
       remove(node);
    }

    private void remove(SplayNode node)
    {
        if (node == null)
            return;

        Splay(node);
        if( (node.left != null) && (node.right !=null))
        {
            SplayNode min = node.left;
            while(min.right!=null)
                min = min.right;

            min.right = node.right;
            node.right.parent = min;
            node.left.parent = null;
            root = node.left;
        }
        else if (node.right != null)
        {
            node.right.parent = null;
            root = node.right;
        }
        else if( node.left !=null)
        {
            node.left.parent = null;
            root = node.left;
        }
       else
        {
            root = null;
        }
        node.parent = null;
        node.left = null;
        node.right = null;
        node = null;
        count--;
    }
     public int countNodes()
    {return count;
    }
     public boolean search(int val)
    {  return findNode(val) != null;
    }
    private SplayNode findNode(int ele)
    { SplayNode PrevNode = null;
        SplayNode z = root;
        while (z != null)
        {
            PrevNode = z;
            if (ele > z.element)
                z = z.right;
            else if (ele < z.element)
                z = z.left;
            else if(ele == z.element) {
                Splay(z);
            return z;
          }}
       if(PrevNode != null)
       { Splay(PrevNode);
          return null;
       }
       return null;     }
     public void inorder()
    {
       inorder(root);
    }
    private void inorder(SplayNode r)
    {
     if (r != null)
       {
       inorder(r.left);
      System.out.print(r.element +" ");
          inorder(r.right);
       }}
      public void preorder()
    {
   preorder(root);
    }
    private void preorder(SplayNode r)
    {
       if (r != null)
       {
          System.out.print(r.element +" ");
          preorder(r.left);
          preorder(r.right);
       }}
     public void postorder()
    {
       postorder(root);
    }
```

```
     private void postorder(SplayNode r)
     {
         if (r != null)
         {
             postorder(r.left);
             postorder(r.right);
             System.out.print(r.element +" ");
         }}}
    public class SplayTreeTest
 {
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
       SplayTree spt = new SplayTree();
        System.out.println("Splay Tree Test\n");
        char ch;
        do
        {
            System.out.println("\nSplay Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove ");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");
            System.out.println("6. clear tree");

          int choice = scan.nextInt();
            switch (choice)
           case 1:
                System.out.println("Enter integer element to insert");
                spt.insert( scan.nextInt() );
                break;
            case 2 :
                System.out.println("Enter integer element to remove");
                spt.remove( scan.nextInt() );
                break;
            case 3 :
                System.out.println("Enter integer element to search");
                System.out.println("Search result : "+ spt.search( scan.nextInt() ));
                break;
            case 4 :
                System.out.println("Nodes = "+ spt.countNodes());
                break;
            case 5 :
                System.out.println("Empty status = "+ spt.isEmpty());
                break;
            case 6 :
                System.out.println("\nTree Cleared");
                spt.clear();
                break;
            default :
            System.out.println("Wrong Entry \n ");
                break;
            }
            System.out.print("\nPost order : ");
            spt.postorder();
            System.out.print("\nPre order : ");
            spt.preorder();
            System.out.print("\nIn order : ");
            spt.inorder();
           System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
  }}}
```

RESULT

Thus the java program to implement Splay Tree has been run successfully and output has been verified.

OUTPUT



```
insert(t, b);
insert(t, c);
insert(t, d);
insert(t, e);
insert(t, f);
insert(t, g);
insert(t, h);
insert(t, i);
insert(t, j);
insert(t, k);
insert(t, l);
insert(t, m);

delete(t, a);
delete(t, m);

inorder(t, t->root);

return 0;
```

```
/tmp/byGunQam13.o
20
30
40
50
60
70
80
90
100
110
150
```

EXPERIMENT – 4

TREAP

AIM

To implement a treap using java program.

ALGORITHM

Step 1: Start

Step 2: Import necessary header packages.

Step 3: Create a new node with key equals to x and value equals to a random value

Step 4: Perform standard Binary search tree insertion.

Step 5: Print the inorder, pre order, and post order of the tree

Step 6: count the total number of node by using the function isEmpty()

Step7: Perform deletion operation if the node is a leaf.

Step 8: if the node has one child NULL and other as non-NULL, find max of left and
right children

Step 9: Stop.

PROGRAM

```
import java.util.Scanner;
 import java.util.Random;
{
TreapNode left, right; int priority, element;
{
this.element = 0;
this.left = this;
this.right = this;
this.priority = Integer.MAX_VALUE;
}
public TreapNode(int ele)
{
this(ele, null, null);
}
public TreapNode(int ele, TreapNode left, TreapNode right)
{
this.element = ele;
this.left = left;
this.right = right;

this.priority = new Random().nextInt( );
}}
class TreapTree
{
private TreapNode root;
private static TreapNode nil = new TreapNode();
public TreapTree()
{
root = nil;
}
public boolean isEmpty()
{
return root == nil;
}
public void makeEmpty()
{ root = nil; }
public void insert(int X)
{
root = insert(X, root);
}
private TreapNode insert(int X, TreapNode T)
```

```
{
if (T == nil)
return new TreapNode(X, nil, nil);
else if (X < T.element)
{
T.left = insert(X, T.left);
if (T.left.priority < T.priority)
{
TreapNode L = T.left;
T.left = L.right;
L.right = T;
return L;
}
}
else if (X > T.element)
{
T.right = insert(X, T.right);
if (T.right.priority < T.priority)
{
TreapNode R = T.right;
T.right = R.left;
R.left = T;
return R;
}
}
return T;
}}
public int countNodes()
{return countNodes(root);
private int countNodes(TreapNode r)
{
if (r == nil)
return 0;
else
{
```

```
int l = 1;
l += countNodes(r.left);
l += countNodes(r.right);
return l;
}
}
public boolean search(int val)
{
return search(root, val);
}
private boolean search(TreapNode r, int val)
{
boolean found = false;
while ((r != nil) && !found)
{
int rval = r.element;
if (val < rval)
r = r.left;
else if (val > rval)
r = r.right;
else
{
found = true;
break;
}
found = search(r, val);
}
return found;
}
public void inorder()
{
inorder(root);
}
private void inorder(TreapNode r)
{
```

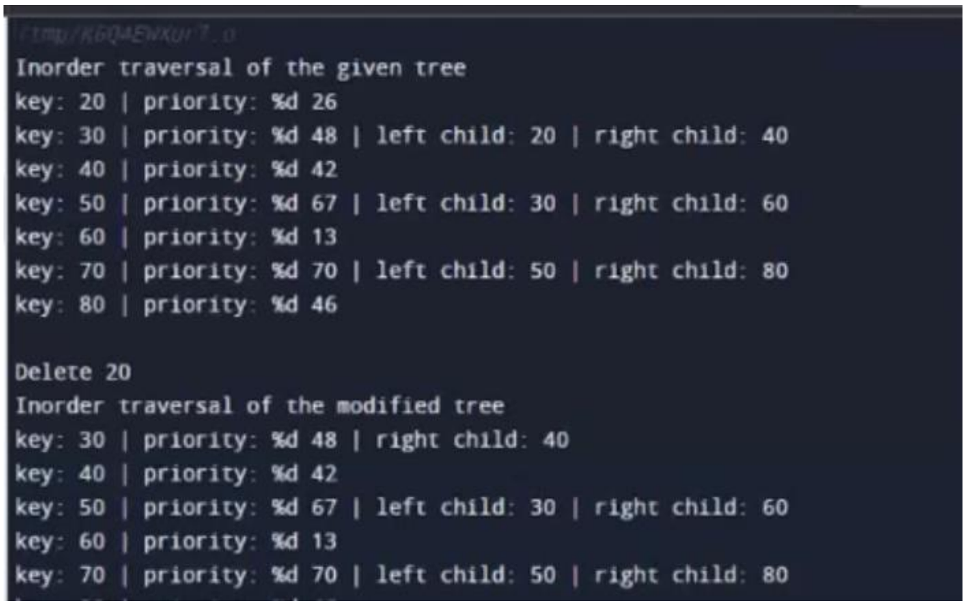
```
if (r != nil)
{
inorder(r.left);
System.out.print(r.element +" ");
inorder(r.right);
}}
public void preorder()
{
preorder(root);}
private void preorder(TreapNode r)
{
if (r != nil)
{
System.out.print(r.element +" ");
preorder(r.left);
preorder(r.right);
}}
public void postorder()
{
postorder(root);
}
private void postorder(TreapNode r)
{
if (r != nil)
{
postorder(r.left);
postorder(r.right);
System.out.print(r.element +" ");
} } }
public class TreapTest
{
public static void main(String[] args)
{
Scanner scan = new Scanner(System.in);
TreapTree trpt = new TreapTree();
```

```
system.out.println("Treap Test\n");
char ch;
do
{
System.out.println("\nTreap Operations\n");
System.out.println("1. insert ");
S.out.println("2. search");
System.out.println("3. count nodes");
System.out.println("4. check empty");
System.out.println("5. clear");
int choice = scan.nextInt();
switch (choice){
case 1 :
System.out.println("Enter integer element to insert"); trpt.insert( scan.nextInt() ); break;
case 2 :
System.out.println("Enter integer element to search"); System.out.println("Search result : "+ trpt.search(
scan.nextInt() )); break;
case 3 :

System.out.println("Nodes = "+ trpt.countNodes()); break;
case 4 :
System.out.println("Empty status = "+ trpt.isEmpty()); break;
case 5 :

System.out.println("\nTreap Cleared");
trpt.makeEmpty();
break;
default :
System.out.println("Wrong Entry \n ");
break;
}
System.out.print("\nPost order : ");
trpt.postorder();
System.out.print("\nPre order : ");
trpt.preorder();
System.out.print("\nIn order : ");
trpt.inorder();
System.out.println("\nDo you want to continue (Type y or n) \n"); ch = scan.next().charAt(0);
} while (ch == 'Y'‖ ch == 'y');
}}
```

RESULT

Thus the program to implement treap has been performed successfully and output was verified.

OUTPUT



```
/tmp/K6Q4EWxur7.o
Inorder traversal of the given tree
key: 20 | priority: %d 26
key: 30 | priority: %d 48 | left child: 20 | right child: 40
key: 40 | priority: %d 42
key: 50 | priority: %d 67 | left child: 30 | right child: 60
key: 60 | priority: %d 13
key: 70 | priority: %d 70 | left child: 50 | right child: 80
key: 80 | priority: %d 46

Delete 20
Inorder traversal of the modified tree
key: 30 | priority: %d 48 | right child: 40
key: 40 | priority: %d 42
key: 50 | priority: %d 67 | left child: 30 | right child: 60
key: 60 | priority: %d 13
key: 70 | priority: %d 70 | left child: 50 | right child: 80
```

EXPERIMENT – 5

MIN-MAX HEAP

AIM

To implement Min-Max Heap using java.

ALGORITHM

Step1: Start

Step2: Import necessary header packages

Step 3: Create a new node at the end of heap.

Step 4: Assign new value to the node.

Step5: The nodes in the left subtree of the root will have datafields that are less than the datafield of the root

Step6: The nodes in the right subtree of the root will have datafields that are greater than the datafield of the root

Step 7: The priority value of the root is less than the priority values of its children.Then it is min heap

Step 8: The priority value of the root is greater than the priority values of its children.Then it is max heap

Step 9: If value of parent is less than child, then swap them.

Step 10: Repeat step 5& 6until Heap property holds.

Step11: Stop

PROGRAM

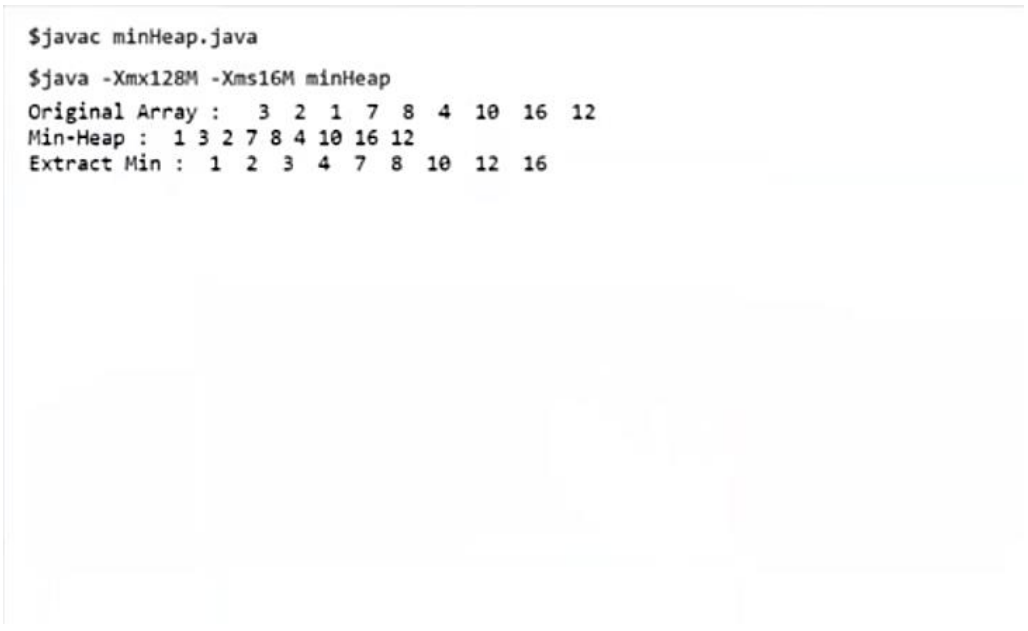
```
public class MinHeap
{
 private int[] Heap;
 private int size;
 private int maxsize;
     private static final int FRONT = 1;
  public MinHeap(int maxsize)
    {  this.maxsize = maxsize;
      this.size = 0;
        Heap = new int[this.maxsize + 1];
        Heap[0] = Integer.MIN_VALUE;
    }
   private int parent(int pos)
    {
     return pos / 2;
    }
     private int leftChild(int pos)
    {
        return (2 * pos);
    }
    private int rightChild(int pos)
    {
        return (2 * pos) + 1;
    }
    private boolean isLeaf(int pos)
    {
        if (pos >=  (size / 2)  &&  pos <= size)
         return true;
        }
        return false;
    }
   private void swap(int fpos, int spos)
    {
        int tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }
    private void minHeapify(int pos)
    {
        if (!isLeaf(pos))
        {
            if ( Heap[pos] > Heap[leftChild(pos)]  || Heap[pos] > Heap[rightChild(pos)])
            {
                if (Heap[leftChild(pos)] < Heap[rightChild(pos)])
                {
                    swap(pos, leftChild(pos));
                    minHeapify(leftChild(pos));
                }else
                {swap(pos, rightChild(pos));
                    minHeapify(rightChild(pos));
                }}}}
     public void insert(int element)
    {
        Heap[++size] = element;
         int current = size;
         while (Heap[current] < Heap[parent(current)])

  {
            swap(current,parent(current));
            current = parent(current);
        }}
    public void print()
    {
        for (int i = 1; i <= size / 2; i++ )
        {
            System.out.print(" PARENT : " + Heap[i] + " LEFT CHILD : " + Heap[2*i]
                + " RIGHT CHILD :" + Heap[2 * i  + 1]);
            System.out.println();
        } }
        public void minHeap()
    {
        for (int pos = (size / 2); pos >= 1 ; pos--)
        {
            minHeapify(pos);
    }}
     public int remove()
    {
        int popped = Heap[FRONT];
        Heap[FRONT] = Heap[size--];
        minHeapify(FRONT);
        return popped;
    }
     public static void main(String...arg)
    {
        System.out.println("The Min Heap is ");
        MinHeap minHeap = new MinHeap(15);
        minHeap.insert(5);
        minHeap.insert(3);
        minHeap.insert(17);
        minHeap.insert(10);
        minHeap.insert(84);
        minHeap.insert(19);
        minHeap.insert(6);
        minHeap.insert(22);
        minHeap.insert(9);
        minHeap.minHeap();
         minHeap.print();
        System.out.println("The Min val is " + minHeap.remove());
    }}
public class MaxHeap
{
    private int[] Heap;
    private int size;
    private int maxsize;
     private static final int FRONT = 1;
Public MaxHeap(int maxsize)
    {

      this.maxsize = maxsize;
        this.size = 0;
        Heap = new int[this.maxsize + 1];
        Heap[0] = Integer.MAX_VALUE;
    }
     private int parent(int pos)
    {
        return pos / 2;
    }
     private int leftChild(int pos)
   {
        return (2 * pos);
    }
     private int rightChild(int pos)
    {
        return (2 * pos) + 1;
    }
  private boolean isLeaf(int pos)
    {
        if (pos >=  (size / 2)  &&  pos <= size)
       {
            return true;
        }
       return false;
    }
     private void swap(int fpos,int spos)
    {
        int tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }

    private void maxHeapify(int pos)
    {
        if (!isLeaf(pos))
        {
            if ( Heap[pos] < Heap[leftChild(pos)]  || Heap[pos] < Heap[rightChild(pos)])
            {
                if (Heap[leftChild(pos)] > Heap[rightChild(pos)])
                {
                    swap(pos, leftChild(pos));
                    maxHeapify(leftChild(pos));
                }else
                {
                    swap(pos, rightChild(pos));
                    maxHeapify(rightChild(pos));
                }}}}

   public void insert(int element)
    {
  Heap[++size] = element;
        int current = size;

        while(Heap[current] > Heap[parent(current)])
        { swap(current,parent(current));
            current = parent(current);
        }  }
public void print()
    { for (int i = 1; i <= size / 2; i++ )
        {
 System.out.print(" PARENT : " + Heap[i] + " LEFT CHILD : " + Heap[2*i]
                  + " RIGHT CHILD :" + Heap[2 * i  + 1]);
            System.out.println();
        }}
```

```
public void maxHeap()
{
    for (int pos = (size / 2); pos >= 1; pos--)
    {
        maxHeapify(pos);
    }
     public int remove()
{    int popped = Heap[FRONT];
    Heap[FRONT] = Heap[size--];
    maxHeapify(FRONT);
    return popped;
}
 public static void main(String...arg)
{
    System.out.println("The Max Heap is ");
    MaxHeap maxHeap = new MaxHeap(15);
    maxHeap.insert(5);
    maxHeap.insert(3);
    maxHeap.insert(17);
    maxHeap.insert(10);
    maxHeap.insert(84);
    maxHeap.insert(19);
    maxHeap.insert(6);
    maxHeap.insert(22);
    maxHeap.insert(9);
    maxHeap.maxHeap();
     maxHeap.print();
    System.out.println("The max val is " + maxHeap.remove());
}}
```

RESULT

Thus the program for implementation of Min-Max Heap has been completed successfully and output has been verified.

OUTPUT



```
$javac minHeap.java

$java -Xmx128M -Xms16M minHeap
Original Array :   3  2  1  7  8  4  10  16  12
Min-Heap :  1 3 2 7 8 4 10 16 12
Extract Min :  1  2  3  4  7  8  10  12  16
```

EXPERIMENT – 6

BINOMIAL HEAP

AIM

To write a program for binary heap in java.

ALGORITHM

Step 1: Start

Step 2: Import necessary header files and Create a Class Binomial Heap Node.

Step 3: Create the constructor k in the binary heap node and reversed.

Step 4: Create a function to min node and node with k value in the binomial heap.

Step 5: Create a function to get size of the binomial heap.

Step 6: Create a class Binomial Heap and check if heap is empty ,function to get size and clear heap.

Step 7: Create the functions to get insert, to unite two binary heap and for union of nodes.

Step 8: Create the functions to return minimum key,delete particular elements,decrease key with a given value,extract the node with the mimimum key and display heap.

Step 9: To make objects and perform binary heap operations and display in the class Binary Heap test.

Step 10: Stop.

PROGRAM

```
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;
}
else
{
} } } }
if (temp1 == null)
```

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

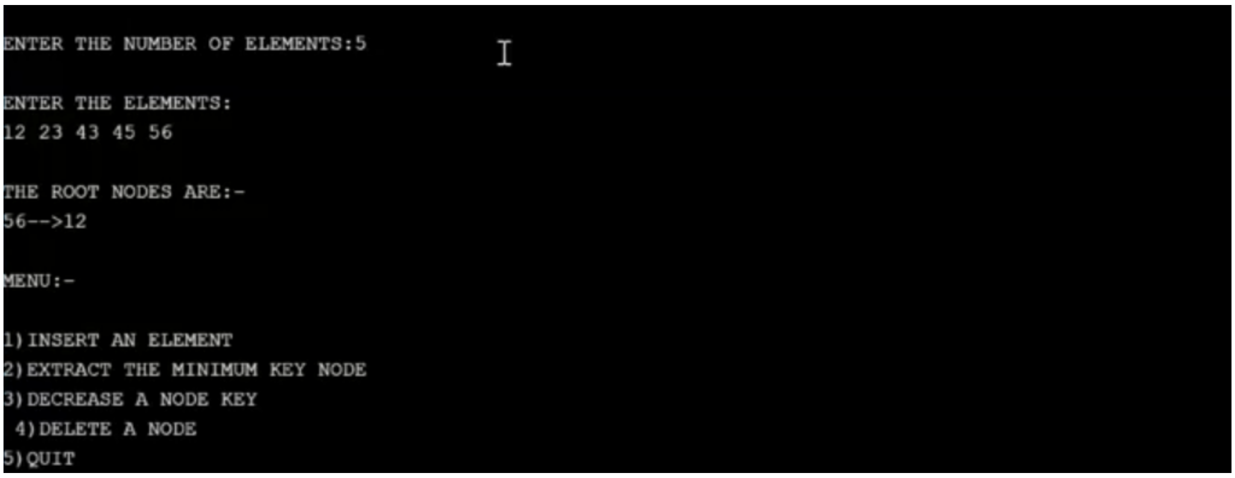
```
size = Nodes.getSize();}
else
{
if ((Nodes != null) && (fakeNode == null))
{size = Nodes.getSize();}
else
{
unionNodes(fakeNode.reverse(null));
size = Nodes.getSize();
}}}
return minNode.key;
}
public void displayHeap()
{System.out.print("\nHeap : ");
displayHeap(Nodes);
System.out.println("\n");
}private void displayHeap(BinomialHeapNode r)
{if (r != null)
{displayHeap(r.child);
System.out.print(r.key +" ");
displayHeap(r.sibling);
}}}
public class BinomialHeapTest
{public static void main(String[] args)
{Scanner scan = new Scanner(System.in);
System.out.println("Binomial Heap Test\n\n");
BinomialHeap bh = new BinomialHeap( );
char ch;
do
{System.out.println("\nBinomialHeap Operations\n");
 System.out.println("1. insert "); System.out.println("2. delete "); System.out.println("3. size");
 System.out.println("4. check empty");
 System.out.println("5. clear");
int choice = scan.nextInt();
switch (choice)
```

```
{case 1 :
System.out.println("Enter integer element to insert");
bh.insert( scan.nextInt() );
break;
case 2 :
System.out.println("Enter element to delete ");
bh.delete( scan.nextInt() );
break;
case 3 :
System.out.println("Size = "+ bh.getSize());
break;
case 4 :
System.out.println("Empty status = "+ bh.isEmpty()); break;
case 5 :
bh.makeEmpty();
System.out.println("Heap Cleared\n");
break;
default :
System.out.println("Wrong Entry \n ");
break;}
bh.displayHeap();
System.out.println("\nDo you want to continue (Type y or n) \n");
 ch = scan.next().charAt(0);
}while (ch == 'Y'|| ch == 'y');}}
```

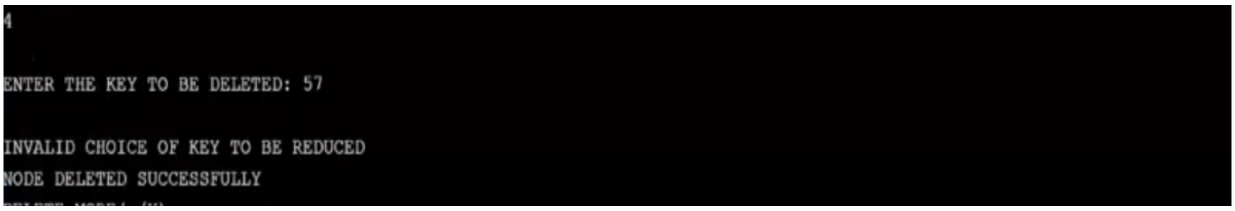
RESULT

The program to perform Binomial Heap has been completed successfully and output was verified using java.

OUTPUT



```
ENTER THE NUMBER OF ELEMENTS:5

ENTER THE ELEMENTS:
12 23 43 45 56

THE ROOT NODES ARE:-
56-->12

MENU:-

1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
 4)DELETE A NODE
5)QUIT
```



```
4

ENTER THE KEY TO BE DELETED: 57

INVALID CHOICE OF KEY TO BE REDUCED
NODE DELETED SUCCESSFULLY
```

EXPERIMENT – 7

DIJKSTRA'S ALGORITHM USING FIBNOACCI HEAP

OBJECTIVE

To perform Dijkstra's algorithm using Fibonacci heap

ALGORITHM

Step 1: Start

Step 2: Import necessary packages.

Step 3: First, find the minimum node in the heap using adjacency list representation in the graph.

Step 4: Then iterate with the neighbour nodes.

Step 5: After the iteration, update the distances of the neighbour nodes.

Step 6: The current distances of the neighbour nodes (which is stored in each node in the heap) is found from that particular node from heap.

Step 7: Using the Dijkstra's algorithm using fibonacci heap can check the shortest path from the graph.

Step 8: Stop the program.

PROGRAM

```
class FibonacciHeap
{private FibonacciHeapNode root;
private int count;
```

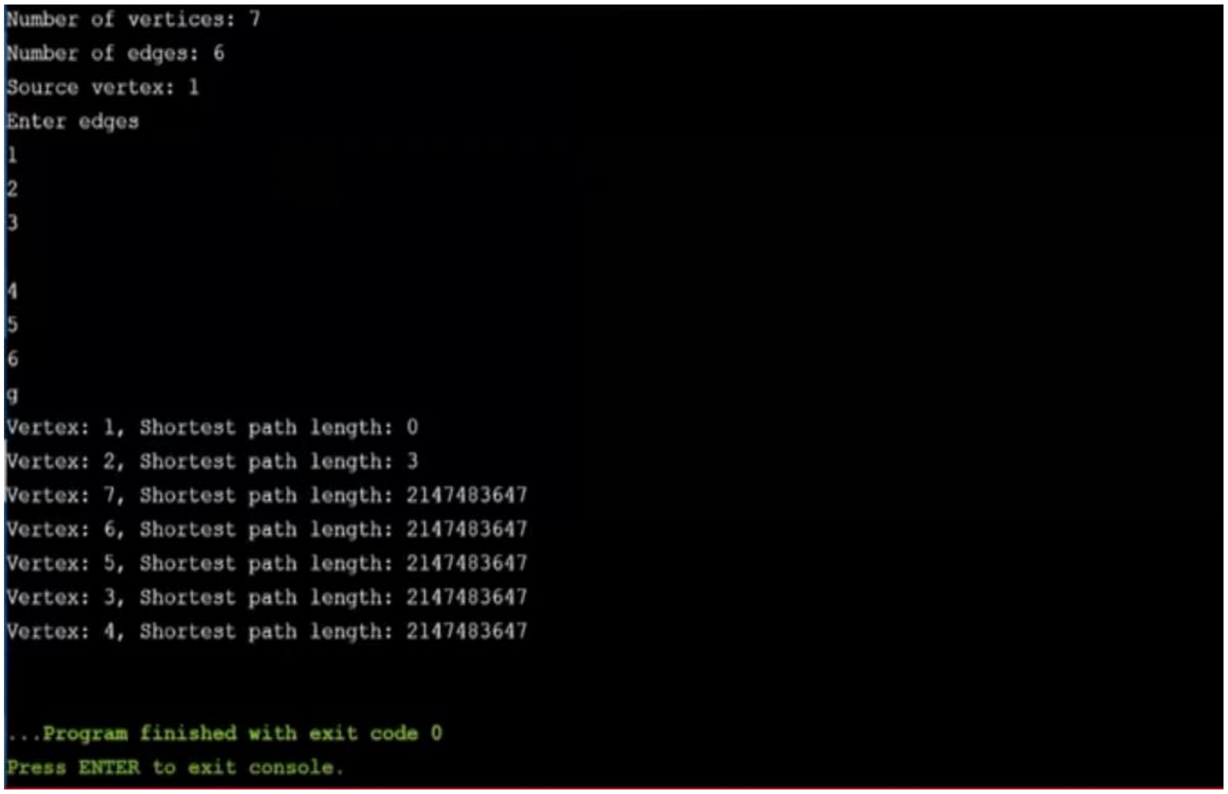
```
public FibonacciHeap()
{
root = null;
count = 0;
}
public boolean isEmpty()
{return root == null;
}
public void clear()
{
root = null;

count = 0;}
public void insert(int element)
{

FibonacciHeapNode node = new FibonacciHeapNode(element);
 node.element = element;
if (root != null)
{
node.left = root;
node.right = root.right;
root.right = node;
node.right.left = node;
if (element < root.element)
root = node;
}
else
root = node;
count++;
}
public void display()
```

```
{
System.out.print("\nHeap = ");
FibonacciHeapNode ptr = root;
if (ptr == null)
{
System.out.print("Empty\n");
return;
}
do
{
System.out.print(ptr.element +" ");
ptr = ptr.right;

} while (ptr != root && ptr.right !=
null); System.out.println();

}
}public class FibonacciHeapTest

{
public static void main(String[] args)
{
Scanner scan = new Scanner(System.in);
System.out.println("FibonacciHeap Test\n\n");
FibonacciHeap fh = new FibonacciHeap();
char ch;
do
{

System.out.println("\nFibonacciHeap Operations\n");
System.out.println("1. insert element ");
System.out.println("2. check empty");
System.out.println("3. clear");
int choice = scan.nextInt();
switch (choice)
{
case 1 :
System.out.println("Enter element");
fh.insert( scan.nextInt() );
break;
case 2 :

System.out.println("Empty status = "+ fh.isEmpty());
break;
case 3 :
fh.clear();
break;
default :
System.out.println("Wrong Entry \n ");
break;
fh.display();

System.out.println("\nDo you want to continue (Type y or n) \n");
ch = scan.next().charAt(0);

}
while (ch == 'Y'|| ch == 'y');
}
}
```

RESULT

The program to perform Dijkstra's algorithm using Fibonacci heap using java has been completed successfully and output was verified.

OUTPUT



```
Number of vertices: 7
Number of edges: 6
Source vertex: 1
Enter edges
1
2
3

4
5
6
9
Vertex: 1, Shortest path length: 0
Vertex: 2, Shortest path length: 3
Vertex: 7, Shortest path length: 2147483647
Vertex: 6, Shortest path length: 2147483647
Vertex: 5, Shortest path length: 2147483647
Vertex: 3, Shortest path length: 2147483647
Vertex: 4, Shortest path length: 2147483647


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT – 8

SKEWED HEAP

AIM

To implement Skewed heap using java

ALGORITHM

Step 1: Start

Step 2: Import necessary packages

Step 3: Create a class SkewNode and a constructor

Step 4: Main operation in Skew Heaps is Merge. We can implement other operations insert,delete, check empty, size etc.

Step 5: Let h1 and h2 be the two min skew heaps to be merged. Let h1's root be smaller than h2's root swap h1->left and h1->right, h1->left = merge(h2, h1>left).

Step 6: Initially the heap is empty and size will be null

Step 7: Insert elements into the skew heap.

Step 8: Check whether the heap is empty or not, if empty return true else false

Step 9: Delete a node from the array

Replace the deletion node with the "farthest right node" on the lowest level

Step 10: using SkewHeap operations can check the status of the size of the heap

Step 11: Stop the program.

PROGRAM

```
    import java.util.*;
   class SkewNode
    {
    int element;
    SkewNode left, right;
    public SkewNode(int val)
    {
   this.element = val;
   this.left = null;
        this.right = null;
    }}
class SkewHeap
{
    private SkewNode root;

    private int size;
    public SkewHeap()
    {
        root = null;
        size = 0;
    }
    public boolean isEmpty()
    {
        return root == null;}
    public void clear()
    {
        root = null;
        size = 0;
    }
    public int getSize()

    {
        return size;
    }
public void insert(int val)
}
public void remove()
    {
        if (root == null)

            throw new NoSuchElementException("Element not found");
        root = merge(root.left, root.right); size--;
    }
    private SkewNode merge(SkewNode x, SkewNode y)

    {
        if (x == null)
            return y;
        if (y == null)
            return x;
if (x.element < y.element)
        {
SkewNode temp = x.left;
            x.left = merge(x.right, y);
            x.right = temp;
            return x;
        }
        else
        {
        SkewNode temp = y.right;
```

```
            y.right = merge(y.left, x);
            y.left = temp;
public void displayHeap()

    {
        System.out.print("\nHeap : ");
        displayHeap(root);
        System.out.println("\n");
    }
    private void displayHeap(SkewNode r)
    {
        if (r != null)
        {
displayHeap(r.left);
            System.out.print(r.element +" ");
            displayHeap(r.right);
        }}}
public class SkewHeapTest
{
public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);

        System.out.println("Skew Heap Test\n\n");
        SkewHeap sh = new SkewHeap( );
       char ch;
        do
        {

            System.out.println("\nSkewHeap
            Operations\n"); System.out.println("1. insert ");
            System.out.println("2. delete ");
```

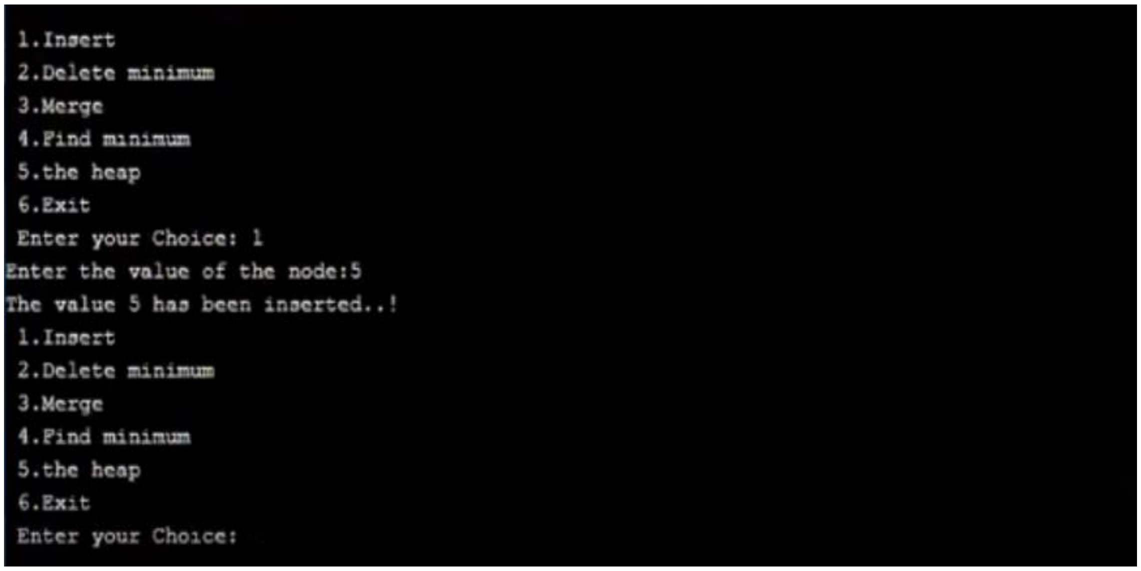
```
 System.out.println("3. size");
 System.out.println("4. check empty");
 System.out.println("5. clear");
int choice = scan.nextInt();
 switch (choice)
 {
 case 1 :
    System.out.println("Enter integer element to insert");
    sh.insert( scan.nextInt() );
    break;
 case 2 :

    sh.remove();
    break;
 case 3 :
    System.out.println("Size = "+ sh.getSize());
    break;
 case 4 :

    System.out.println("Empty status = "+
    sh.isEmpty()); break;

 case 5:
    sh.clear();
    System.out.println("Heap Cleared\n");
    break;
 default :
    System.out.println("Wrong Entry \n ");
    break;
 }
 sh.displayHeap();
```

```
            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);

        } while (ch == 'Y'|| ch == 'y');

    }
}
```

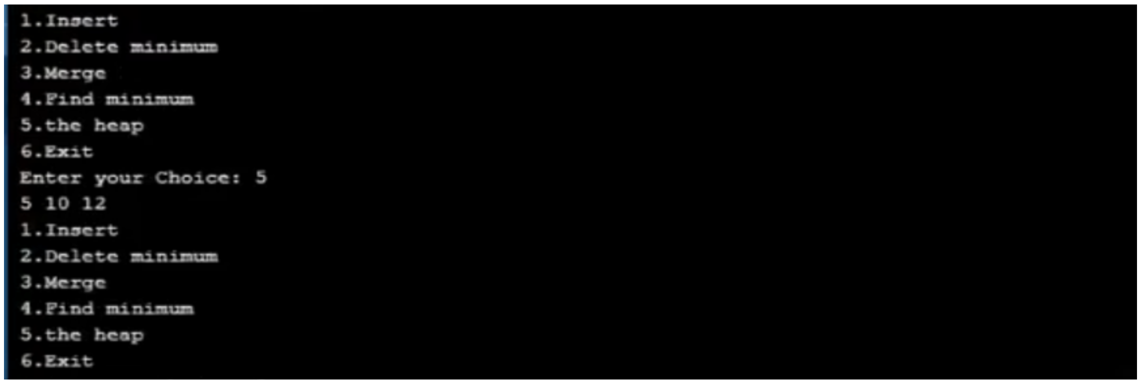
RESULT

The program to implement skewed Heap using java has been completed successfully and the output was verified.

OUTPUT



```
 1.Insert
 2.Delete minimum
 3.Merge
 4.Find minimum
 5.the heap
 6.Exit
 Enter your Choice: 1
Enter the value of the node:5
The value 5 has been inserted..!
 1.Insert
 2.Delete minimum
 3.Merge
 4.Find minimum
 5.the heap
 6.Exit
 Enter your Choice:
```



```
1.Insert
2.Delete minimum
3.Merge
4.Find minimum
5.the heap
6.Exit
Enter your Choice: 5
5 10 12
1.Insert
2.Delete minimum
3.Merge
4.Find minimum
5.the heap
6.Exit
```

EXPERIMENT – 9

K-D TREES

AIM

To implement K-D Tree using java.

ALGORITHM

Step 1: Start

Step 2: Import the necessary packages.

Step 3: Create Constructor for a balanced tree.

Step 4: Create nodes from the list of xyz points.

Step 5: Add values to the node.

Step 6: Check the value contains in the tree.

Step 7: Perform find parent operation and return parent.

Step 8: Insert necessary elements.

Step 9: Find the nearest node and search parent node using search_parent operation.

Step 10: Stop

PROGRAM

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
class KDNode
{
int axis;
double[] x;
int id;
boolean checked;
boolean orientation;
KDNode Parent;
KDNode Left;
KDNode Right;
public KDNode(double[] x0, int axis0)
{
x = new double[2];
axis = axis0;
for (int k = 0; k < 2; k++)
x[k] = x0[k];
Left = Right = Parent = null;
checked = false;
id = 0;
}
public KDNode FindParent(double[] x0)
{
KDNode parent = null;
KDNode next = this;
int split;
while (next != null)
{
split = next.axis;
parent = next;
if (x0[split] > next.x[split])
next = next.Right;
else
next = next.Left;
}
return parent;
}
public KDNode Insert(double[] p)
{
KDNode parent = FindParent(p);
if (equal(p, parent.x, 2) == true)
return null;
KDNode newNode = new KDNode(p, parent.axis + 1 < 2 ? parent.axis + 1:0);
newNode.Parent = parent;
if (p[parent.axis] > parent.x[parent.axis])
 {
parent.Right = newNode;
} else
{
parent.Left = newNode; newNode.orientation = false;
}
return newNode;
}
boolean equal(double[] x1, double[] x2, int dim)
{
for (int k = 0; k < dim; k++)
{
if (x1[k] != x2[k])
return false;
}
return true;}
double distance2(double[] x1, double[] x2, int dim)
{
```

```
double S = 0;
for (int k = 0; k < dim; k++)
S += (x1[k] - x2[k]) * (x1[k] - x2[k]);
return S;
}}
class KDTree
{
KDNode Root;
int TimeStart, TimeFinish;
int CounterFreq;
double d_min;
KDNode nearest_neighbour;
int KD_id;
int nList;
KDNode CheckedNodes[];
int checked_nodes;
KDNode List[];
double x_min[], x_max[];
boolean max_boundary[], min_boundary[];
int n_boundary;
public KDTree(int i)
{
Root = null;
KD_id = 1;
nList = 0;
List = new KDNode[i];
CheckedNodes = new KDNode[i];
max_boundary = new boolean[2];
min_boundary = new boolean[2];
x_min = new double[2];
x_max = new double[2];
}
```

```
public boolean add(double[] x)
{
if (nList >= 2000000 - 1)
return false; if (Root == null)
{
Root = new KDNode(x, 0);
Root.id = KD_id++;
List[nList++] = Root;
} else
{
KDNode pNode;
if ((pNode = Root.Insert(x)) != null)
{
pNode.id = KD_id++;
List[nList++] = pNode;
}}
return true;}
public KDNode find_nearest(double[] x)
{
if (Root == null)
return null;
checked_nodes = 0;
KDNode parent = Root.FindParent(x);
nearest_neighbour = parent;
d_min = Root.distance2(x, parent.x, 2);
if (parent.equal(x, parent.x, 2) == true)
return nearest_neighbour;
search_parent(parent, x);
uncheck();
return nearest_neighbour;
}
public void check_subtree(KDNode node, double[] x)
```

```
{if ((node == null) || node.checked)
return;
CheckedNodes[checked_nodes++] = node;
node.checked = true;
set_bounding_cube(node, x);
int dim = node.axis;
double d = node.x[dim] - x[dim];
if (d * d > d_min){
if (node.x[dim] > x[dim])
check_subtree(node.Left, x);
else
check_subtree(node.Right, x);
} else
{
check_subtree(node.Left, x); check_subtree(node.Right, x);
}}
public void set_bounding_cube(KDNode node, double[] x)
{
if (node == null)
return;
int d = 0;
double dx;
for (int k = 0; k < 2; k++)
{dx = node.x[k] - x[k];
if (dx > 0)
{
dx *= dx;
if (!max_boundary[k]){
if (dx > x_max[k])
x_max[k] = dx;
if (x_max[k] > d_min){
max_boundary[k] = true;
n_boundary++;
}}} else
{
dx *= dx;
if (!min_boundary[k])
{
x_min[k] = dx;
if (x_min[k] > d_min)
{
min_boundary[k] = true;
n_boundary++;
}}}
d += dx;
if (d > d_min)
return;
}
if (d < d_min)
{
d_min = d;
nearest_neighbour = node;
}}
public KDNode search_parent(KDNode parent, double[] x)
{
for (int k = 0; k < 2; k++)
{
x_min[k] = x_max[k] = 0;
max_boundary[k] = min_boundary[k] = false; //
}
n_boundary = 0;
KDNode search_root = parent;
while (parent != null && (n_boundary != 2 * 2))
{check_subtree(parent, x);
```

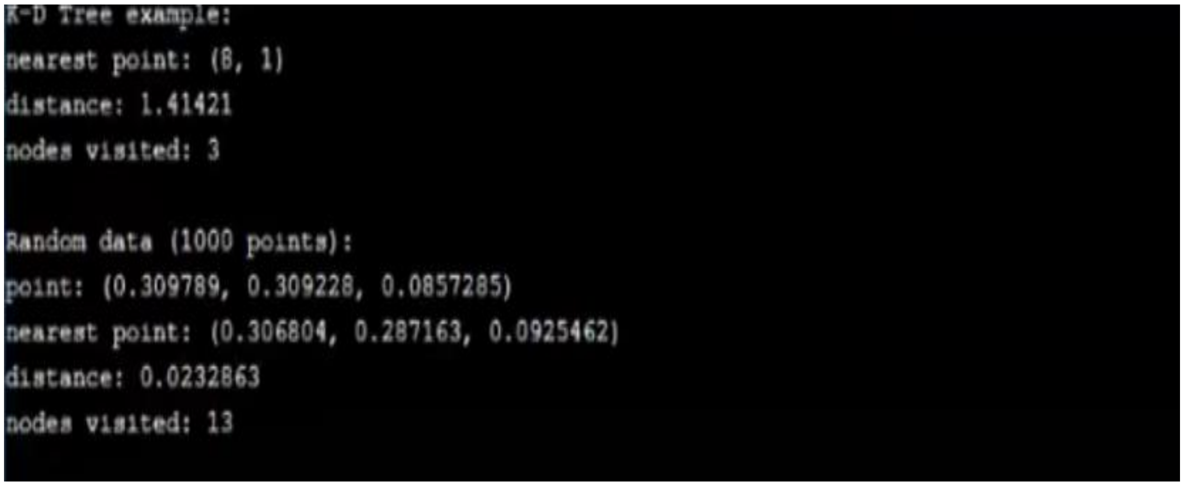
```
search_root = parent;
parent = parent.Parent;
}
return search_root;
}
public void uncheck(){
for (int n = 0; n < checked_nodes; n++)
CheckedNodes[n].checked = false;
}}
public class KDTNearest
{
public static void main(String args[]) throws IOException {
BufferedReader in = new BufferedReader(new FileReader("input.txt")); int numpoints = 5;
KDTree kdt = new KDTree(numpoints);
double x[] = new double[2];
x[0] = 2.1;
x[1] = 4.3;
kdt.add(x);
x[0] = 3.3;
[1] = 1.5;
kdt.add(x);
x[0] = 4.7;
x[1] = 11.1;

kdt.add(x);
x[0] = 5.0;
x[1] = 12.3;
kdt.add(x);
x[0] = 5.1;
x[1] = 1.2;
kdt.add(x);
System.out.println("Enter the co-ordinates of the point: (one after the other)");
```

```
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(reader); double sx =
Double.parseDouble(br.readLine());
double sy = Double.parseDouble(br.readLine());
double s[] = { sx, sy };
KDNode kdn = kdt.find_nearest(s); System.out.println("The nearest neighbor is: ");
System.out.println("(" + kdn.x[0] + " , " + kdn.x[1] + ")"); in.close();
}}
```

RESULT

Thus the program to implement K-D Tree using java has been completed successfully and the output was verified.

OUTPUT



```
K-D Tree example:
nearest point: (8, 1)
distance: 1.41421
nodes visited: 3

Random data (1000 points):
point: (0.309789, 0.309228, 0.0857285)
nearest point: (0.306804, 0.287163, 0.0925462)
distance: 0.0232863
nodes visited: 13
```