WORKSHEET-B ANSWERS

1) D
2) Program:

```
package com.java.BinaryTree;

import java.util.stream.IntStream;

public class Vowel {
public static void main(String[] args) {
        String s="125";
        boolean hasVowel = IntStream.range(0,s.length())
                        .mapToObj(s::charAt)
                        .anyMatch(c -> "aeiouAEIOU".indexOf(c)>=0);
        System.out.println(hasVowel ? "True" : "False");
}
}
```

Output :

False

3) Program :

```
  package com.java.Trees;

import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;

public class RemoveDuplicateArrayList {
        public static void main(String[] args) {
      List<String> l = new ArrayList<String>();
      l.add("1");
      l.add("2");
      l.add("1");
      l.add("2");
      l.add("3");
      l.add("5");
      l.add("3");
      System.out.println(l.toString());
      Set<String> s = 5)new LinkedHashSet<String>(l);
      System.out.println(s);
  }
    }
```

Output :

[1, 2, 1, 2, 3, 5, 3]
[1, 2, 3, 5]

5) Program :

```
package com.java.Trees;

public class Test1 {
        public static int sumOfMiddleRow(int [][] matrix, int n, int m){
          int totalSum =0; //variable to store the total sum value

          // Iterating over the middle column and picking the middle value
          for(int col = 0; col<m; col++){
            totalSum += matrix[n/2][col];
          }
          return totalSum;
        }

        // function to calculate the sum of the middle column of a matrix
        public static int sumOfMiddleColumn(int [][] matrix, int n, int m){
          int totalSum =0; //variable to store the total sum value

          // Iterating over all rows and picking the middle value
          for(int row = 0; row<n; row++){
            totalSum += matrix[row][m/2];
          }
          return totalSum;
        }

        public static void main(String[] args) {
          int n= 3; // number of rows
          int m = 3; // number of columns
          // Input
          int [][]matrix = {{5, 7, 9},
                            {8, 5, 10},
                            {12, 8, 10}};

          System.out.println("Sum of the middle row: " +
Integer.toString(sumOfMiddleRow(matrix,n,m)));
          System.out.println("Sum of the middle column: "+
Integer.toString(sumOfMiddleColumn(matrix,n,m)));
        }


}
```

Output :

Sum of the middle row: 23
Sum of the middle column: 20

6) Program :

```
package com.java.collection;

public class Node {
	int data;
	Node next;
	Node(int d) {data = d;
				next = null;}
}

class Main
{
Node head;
public void addToTheLast(Node node)
{
	if (head == null)
	{
		head = node;
	}
	else
	{
		Node temp = head;
		while (temp.next != null)
			temp = temp.next;
		temp.next = node;
	}
}
void printList()
{
	Node temp = head;
	while (temp != null)
	{
		System.out.print(temp.data + " ");
		temp = temp.next;
	}
	System.out.println();
}
public static void main(String args[])
{
	Main head1 = new Main();
	Main head2 = new Main();

	head1.addToTheLast(new Node(1));
```

```
        head1.addToTheLast(new Node(2));
        head1.addToTheLast(new Node(4));
        head1.addToTheLast(new Node(6));
        head1.addToTheLast(new Node(9));

        head2.addToTheLast(new Node(3));
        head2.addToTheLast(new Node(4));
        head2.addToTheLast(new Node(7));
        head2.addToTheLast(new Node(8));


        head1.head = new Mergesortedlists().MergeSortedLists(head1.head,head2.head);
        head1.printList();

}
}
```

Output :

1 2 3 4 4 6 7 8 9

Q7) Program :

```
package com.java.Trees;


import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Queue;
import java.util.Set;
import java.util.TreeMap;
import java.util.LinkedList;

class BinaryTreeNode {
        int v;
        // for keeping the horizontal distance value of the node
        // taking root node as the reference
        int horDis;
        // l and r represent the
        // left child and the right child of a node
        BinaryTreeNode l, r;
        // constructor of the class BinaryTreeNode
        // the construct and initializes the class fields
        public BinaryTreeNode(int i)
        {
```

```
v = i;
r = null;
l = null;
horDis = Integer.MAX_VALUE;
}
}
//Tree class
class Tree
{
BinaryTreeNode rt; // root node of the tree
// constructor of the tree
public Tree()
{
}
// Parameterized constructor of the tree
public Tree(BinaryTreeNode node)
{
rt = node;
}
// a method that shows the bottom view of the tree
public void bottomViewBT()
{
if (rt == null)
{
return;
}
// Initializing a variable 'horDis' with the value 0 for the root element.
int horDis = 0;
// A TreeMap that keeps the key-value pair where sorting is done on the key-value
Map<Integer, Integer> mp = new TreeMap<>();
// A queue to keep the nodes of the tree in the level order traversal
Queue< BinaryTreeNode > que = new LinkedList< BinaryTreeNode >();
// Assigning the initialized horizontal distance value to the root
// node and adding the node to the queue.
rt.v = horDis;
que.add(rt);
// Loop until the queue is empty (standard level order loop)
while (!que.isEmpty())
{
BinaryTreeNode tmp = que.remove();
// extracting the value of horizontal distance value from
// the dequeued node of the binary tree.
horDis = tmp.horDis;
// putting the dequeued node of the binary tree to the TreeMap using key
// as the horizontal distance. Whenever we come across a node
// that has the same horizontal distance, it is required to replace
// the value in the TreeMap.
```

```
        mp.put(horDis, tmp.v);
        // If the node that has been dequeued contains a left child, then
        // add the left child to the queue along with the horizontal distance hd - 1. It is
because
        // we are moving in the left direction
        if (tmp.l != null)
        {
        tmp.l.horDis = horDis - 1;
        que.add(tmp.l);
        }
        // If the node that has been dequeued contains a right child, then
        // add the right child to the queue along with the horizontal distance hd + 1. It is
because
        // we are moving in the right direction
        if (tmp.r != null)
        {
        tmp.r.horDis = horDis + 1;
        que.add(tmp.r);
        }
        }
        // extracting the entries from map into a set s in order to traverse
        // an iterator.
        Set<Entry<Integer, Integer>> s = mp.entrySet();
        // Make an iterator
        Iterator<Entry<Integer, Integer>> itr = s.iterator();
        // Traverse the map elements using the iterator.
        while (itr.hasNext())
        {
        Map.Entry<Integer, Integer> me = itr.next();
        System.out.print(me.getValue() + " ");
        }
        }
        }
public class BottomViewExample {
        // main method
                public static void main(String[] argvs)
                {
                // root node
                BinaryTreeNode  rt = new BinaryTreeNode (20);
                // other nodes of the tree
                rt.l = new BinaryTreeNode (22);
                rt.r = new BinaryTreeNode (8);
                rt.l.l = new BinaryTreeNode (25);
                rt.l.r = new BinaryTreeNode (3);
                rt.r.r = new BinaryTreeNode (5);
                rt.l.r.r = new BinaryTreeNode (10);
                rt.l.r.l = new BinaryTreeNode (14);
```

```
                rt.l.r.l.r = new BinaryTreeNode (7);
                // creating an object of the class Tree
                Tree tr = new Tree(rt);
                System.out.println("The following are the nodes present in the bottom view
of the Binary Tree: ");
                // invoking the method bottomViewBT
                tr.bottomViewBT();
                }

}
```

Output :

The following are the nodes present in the bottom view of the Binary Tree:
10 5 25 14 7

Q8) Program :

```
package com.java.Trees;

public class Main1 {
        // a binary tree node has data, pointer to
        // left child and a pointer to right child
        static class Node {
           int data;
           Node left;
           Node right;
        }

        // function that allocates
        // a new node with the given data and null left and right pointers
        static Node newNode(int data) {
           Node newNode = new Node();
           newNode.data = data;
           newNode.left = null;
           newNode.right = null;
           return newNode;
        }

        // function to print Inorder traversal
        static void inorder(Node root) {
           if (root == null)
              return;
           inorder(root.left);
           System.out.print(root.data + " ");
           inorder(root.right);
        }
```

```
    static Node mirror(Node root) {
       if (root == null) {
          return null;

       }
       // mirror the subtrees
       Node mirror = newNode(root.data);
       mirror.right = mirror(root.left);
       mirror.left = mirror(root.right);
       return mirror;
    }

    public static void main(String args[]) {

       Node root = newNode(1);
       root.left = newNode(2);
       root.right = newNode(3);
       root.left.left = newNode(4);
       root.left.right = newNode(5);
       root.right.left = newNode(6);
       root.right.right = newNode(7);

       //  inorder traversal of the input tree
       System.out.print("Inorder traversal of original tree is \n");
       inorder(root);

       Node mirror = null;
       mirror = mirror(root);

       // inorder traversal of the mirrored tree
       System.out.print("\nInorder traversal of the mirrored tree is \n");
       inorder(mirror);
       }
    }
```

Output :

```
Inorder traversal of original tree is
4 2 5 1 6 3 7
Inorder traversal of the mirrored tree is
7 3 6 1 5 2 4
```

Q9) Program :

```
package com.java.Trees;
class Node {
```

```
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}


public class BinartTree1 {

        Node root1, root2;

    /* Given two trees, return true if they are
       structurally identical */
    boolean identicalTrees(Node a, Node b)
    {
        /*1. both empty */
        if (a == null && b == null)
            return true;

        /* 2. both non-empty -> compare them */
        if (a != null && b != null)
            return (a.data == b.data
                    && identicalTrees(a.left, b.left)
                    && identicalTrees(a.right, b.right));

        /* 3. one empty, one not -> false */
        return false;
    }

    /* Driver code*/
    public static void main(String[] args)
    {
      BinartTree1 tree = new BinartTree1 ();

        tree.root1 = new Node(1);
        tree.root1.left = new Node(2);
        tree.root1.right = new Node(3);
        tree.root1.left.left = new Node(4);
        tree.root1.left.right = new Node(5);

        tree.root2 = new Node(1);
        tree.root2.left = new Node(2);
        tree.root2.right = new Node(3);
```

```
        tree.root2.left.left = new Node(4);
        tree.root2.left.right = new Node(5);

         // Function call
        if (tree.identicalTrees(tree.root1, tree.root2))
            System.out.println("Both trees are identical");
        else
            System.out.println("Trees are not identical");
    }
}
```

Output:

Both Trees are identical.

Q10) Program :

```
package com.java.BinaryTree;

import java.util.Scanner;

public class Main {

        public static void main(String[] args)
         {
            boolean b = true;
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter a number : ");
            int num = in.nextInt();
           {
           while(num!=1)
            {
              if(num%2!=0)
                {
                  b=! b;
                  System.out.print(b);
                  System.exit(0);
                }
                  num = num / 2;
            }
             System.out.print(b);
           }
         }

        }
```

Output :

Enter a number : 64
true