Worksheet-6

Ans.1

```
package linked;

public class SinglyLinkedList {                  //linked list creation
private ListNode head;
private class ListNode{
      private int data;
      private ListNode next;
      public ListNode (int data) {
this.data=data;
this.next=null;
      }
}
public void insertAtBeginning (int data) {            //method creation
      ListNode newNode = new ListNode (data);
      newNode.next=head;
      head=newNode;

}
public void printLinkedList() {                       //list printed by  print method
      if (head == null) {
            System.out.println("null");
      }
      ListNode current=head;
      while(null !=current) {
            System.out.println(current.data+ "-->");
            current=current.next;
      }
      System.out.println("null");

}
public ListNode getMiddleNode() {
      if (head==null) {
            return null;
      }
      ListNode slowPtr=head;
      ListNode fastPtr=head;

      while(fastPtr !=null && fastPtr.next !=null) {
            slowPtr=slowPtr.next;
            fastPtr=fastPtr.next.next;
      }
      return slowPtr;
}
  public ListNode getNthNodeFromEnd (int n) {               //method creation
        if(head==null) {
              return null;
        }
        if(n<=0) {
  throw new IllegalArgumentException("Invalid value:n="+n);

  }
ListNode mainPtr =head;
ListNode refPtr =head;
int count=0;
while (count<n) {
      if (refPtr ==null) {
            throw new IllegalArgumentException(n+"is greater thanthe number of nodes on
list");

      }
      refPtr=refPtr.next;
      count++;

}
```

```
while(refPtr !=null) {
      refPtr =refPtr.next;
      mainPtr=mainPtr.next;
}
return mainPtr;
  }
  public void removeDuplicate() {
        if (head==null) {
              return;
}
  ListNode current=head;
  while(current !=null && current.next !=null) {
        if(current.data== current.next.data) {
              current.next=current.next.next;
        }else {
              current=current.next;
        }
        }
  }
  public ListNode insertInSortedList(int value) {     //node inserted with parameter
ListNode newNode=new ListNode(value);
if(head==null) {
      return newNode;
}
ListNode current=head;
ListNode temp=null;

while(current !=null && current.data<newNode.data) {
temp=current;
current=current.next;

}
newNode .next=current;
temp.next=newNode;
return head;
  }
  public static void main(String[] args) {            //Main program started
        SinglyLinkedList sll=new SinglyLinkedList();
        //insert elements
        sll.insertAtBeginning(16);
        sll.insertAtBeginning(10);
        sll.insertAtBeginning(8);
        sll.insertAtBeginning(1);
        sll.printLinkedList();
        sll.insertInSortedList(11);       //node insertion
       sll.printLinkedList();                    //function call

  }
}
```

Ans.2

```
package linked;
import java.util.*;
public class Main {
 public static class BinaryTree  //Binary tree class
 {
public class Node  //Node class
{
      int data;
      Node left;
      Node right;

      public Node (int data)
```

```
	{
		this.data=data;
		this.left=null;
		this.right=null;

	}
}
private Node root;
 public BinaryTree (int[]pre, int[]post)
 {
	 this.root= this.construct (pre, 0, pre.length  - 1, post, 0,post.length-1);

 }
 private Node construct(int[] pre,int presi ,int preei,int[]post, int postsi,int posti)
 {
                                  // This case occurs when a node has only one child.
	 if(presi > preei)
	 {
		return null;

	 }
	 Node node=new Node (pre[presi]);
	 node.left=null;
	 node.right=null;
	 if(presi==preei)
	 {
		return node;

	 }
                                          // searching pre[presi+1]in post order array
	 int pos=-1;
	 for(int i=postsi; i<=posti; i++)
	 {
		if(post[i] == pre[presi + 1])
		{
	 pos=i;
	 break;

	 }
 }
                                                  //Number of elements in left subtree
 int clc=pos-postsi+1;
                                                                          //left subtree

 node.left =this.construct(pre,presi+1,presi+clc,post,postsi,pos);
                                                                          //right subtree
 node.right=this.construct(pre,presi+clc+1,preei,post,pos+1,postsi-1);
 return node;
}
public int height()
{
	return this.height (this.root);
}
                                                  //Function to fing height of binary
                                          tree
private int height (Node node)
{
                                                        //Base case
	if(node==null)
	{
		return-1;
	}
                                                  //calculate height of left subtree
	int lht=this.height(node.left);

                                                  //Calculate height of right subtree
	int rht=this.height(node.right);
```

```
                                //Height of the tree is max of left and right subtree
height+1
      int rv=Math.max(lht, rht)+1;
      return rv;
}
}
public static void main (String[] args) throws Exception
{
                                                                    // Construct binary tree
      int[] pre = {50,25,12,37,30,40,75,62,60,70,87};
      int[]post= {12,30,40,37,25,60,70,62,87,75,50};
      BinaryTree bt=new BinaryTree(pre,post);
      System.out.println("Height of the tree is:"+bt.height());
 }
}
```

Ans.3

```
      package linked;
      import java.io.*;

      class GFG {


                        /* A binary tree node has data, pointer to left child

                                and a pointer to right child */
        static class node {
          int data;
          node left, right;
        }

      /* Helper function that allocates a new node with the

      given data and NULL left and right pointers. */
static node newNode(int data)
{
  node Node = new node();
  Node.data = data;
  Node.left = Node.right = null;

  return Node;
}

static int maxValue(node Node)
{
  if (Node == null) {
    return Integer.MIN_VALUE;
  }
  int value = Node.data;
  int leftMax = maxValue(Node.left);
  int rightMax = maxValue(Node.right);
  return Math.max(value, Math.max(leftMax, rightMax));
}

static int minValue(node Node)
{
  if (Node == null) {
    return Integer.MAX_VALUE;
  }
  int value = Node.data;
  int leftMax = minValue(Node.left);
  int rightMax = minValue(Node.right);

  return Math.min(value, Math.min(leftMax, rightMax));
}
```

```
                              /* Returns true if a binary tree is a binary search
tree*/


static int isBST(node Node)
{
  if (Node == null) {
    return 1;

}



                                      /* false if the max of the left is > than us */
if (Node.left != null
    && maxValue(Node.left) > Node.data) {
  return 0;
}


                                           /* false if the min of the right is <= than
us */
if (Node.right != null
    && minValue(Node.right) < Node.data) {
  return 0;
}

                                      /* false if, recursively, the left or right is not
a


                         * BST*/
if (isBST(Node.left) != 1
|| isBST(Node.right) != 1) {
return 0;
}


                                           /* passing all that, it's a BST */
return 1;
}

public static void main(String[] args)
{
node root = newNode(4);
root.left = newNode(2);
root.right = newNode(5);


                                      // root->right->left = newNode(7);
root.left.left = newNode(1);
root.left.right = newNode(3);


                                      // Function call
if (isBST(root) == 1) {
  System.out.print("Is BST");
}
else {
  System.out.print("Not a BST");
}
}
}
```

Ans.5

```
package linked;
import java.util.*;

class GFG {

                    // Binary Tree Node
    static class Node {
        int data;
        Node left, right;

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

            // function to print the left view of binary tree
    public static ArrayList<Integer> leftView(Node root)
    {

                                                                    // code  is here
        ArrayList<Integer> ans = new ArrayList<>();

        if (root == null) {
            return ans;
        }

        Queue<Node> q = new LinkedList<>();
        q.add(root);
        q.add(null);
        boolean ok = true;

        while (!q.isEmpty()) {
                  //while loop checking
            Node it = q.poll();
            if (it == null) {
                if (ok == false) {
                    ok = true;
                }

                if (q.size() == 0)
                    break;

                else {
                    q.add(null);
                }
            }
            else {

                if (ok) {
                    ans.add(it.data);
                    ok = false;
                }

                if (it.left != null) {
                    q.add(it.left);
                }

                if (it.right != null) {
                    q.add(it.right);
                }
            }
        }
```

```
        return ans;
    }

                        // main program started
    public static void main(String[] args)
    {
        Node root = new Node(10);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(7);
        root.left.right = new Node(8);
        root.right.right = new Node(15);
        root.right.left = new Node(12);
        root.right.right.left = new Node(14);

        ArrayList<Integer> vec = leftView(root);
        for (int x : vec) {
            System.out.print(x + " ");
        }
        System.out.println();
    }
}
```