Ans 1)

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

class LeftHeapNode {
	int element, sValue;
	LeftHeapNode left, right;
	int height = 1;

	public LeftHeapNode(int ele) {
		this(ele, null, null);
	}

	public LeftHeapNode(int ele, LeftHeapNode left, LeftHeapNode right) {
		this.element = ele;
		this.left = left;
		this.right = right;
		this.sValue = 0;
	}
}

class LeftistHeap {
	private LeftHeapNode root;

	public LeftistHeap() {
		root = null;
	}

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

	public void clear() {
		root = null;
	}

	public void insert(int x) {
		root = merge(new LeftHeapNode(x), root);
	}

	public void merge(LeftistHeap rhs) {
		if (this == rhs)
			return;
		root = merge(root, rhs.root);
		root.height = Math.max(calcheight(root.left), calcheight(root.right)) + 1;
		rhs.root = null;
	}

	private LeftHeapNode merge(LeftHeapNode x, LeftHeapNode y) {
		if (x == null)
			return y;
```

```
        if (y == null)
            return x;
        if (x.element > y.element) {
            LeftHeapNode temp = x;
            x = y;
            y = temp;
        }

        x.right = merge(x.right, y);

        if (x.left == null) {
            x.left = x.right;
            x.right = null;
        } else {
            if (x.left.sValue < x.right.sValue) {
                LeftHeapNode temp = x.left;
                x.left = x.right;
                x.right = temp;
            }
            x.sValue = x.right.sValue + 1;
        }
        x.height = Math.max(calcheight(x.left), calcheight(x.right)) + 1;
        return x;
    }

    public int deleteMin() {
        if (isEmpty())
            return -1;
        int minItem = root.element;
        root = merge(root.left, root.right);
        return minItem;
    }

    private int calcheight(LeftHeapNode N) {
        if (N == null) {
            return 0;
        }
        return N.height;
    }

    public void inorder() {
        print(root);
        System.out.println();
    }

    private void print(LeftHeapNode root) {

        if (root == null) {
            System.out.println("(XXXXXX)");
            return;
        }

        int height = root.height, width = (int) Math.pow(2, height - 1);
```

```
        List < LeftHeapNode > current = new ArrayList < LeftHeapNode > (1), next = new
ArrayList < LeftHeapNode > (2);
        current.add(root);
        final int maxHalfLength = 4;
        int elements = 1;

        StringBuilder sb = new StringBuilder(maxHalfLength * width);
        for (int i = 0; i < maxHalfLength * width; i++) {
            sb.append(' ');
        }

        String textBuffer;

        // Iterating through height levels.
        for (int i = 0; i < height; i++) {

            sb.setLength(maxHalfLength * ((int) Math.pow(2, height - 1 - i) - 1));

            // Creating spacer space indicator.
            textBuffer = sb.toString();

            // Print tree node elements
            for (LeftHeapNode n: current) {

                System.out.print(textBuffer);

                if (n == null) {
                    System.out.print("        ");
                    next.add(null);
                    next.add(null);
                } else {

                    System.out.printf("(%6d)", n.element);
                    next.add(n.left);
                    next.add(n.right);

                }

                System.out.print(textBuffer);

            }

            System.out.println();

            // Print tree node extensions for next level.
            if (i < height - 1) {

                for (LeftHeapNode n: current) {
                    System.out.print(textBuffer);

                    if (n == null) {
                        System.out.print("        ");
                    } else {
                        System.out.printf("%s      %s",
```

```
                                n.left == null ? " " : "/", n.right == null ? " " : "\\");
                        }

                            System.out.print(textBuffer);

                        }

                        System.out.println();

                    }

                    elements *= 2;
                    current = next;
                    next = new ArrayList < LeftHeapNode > (elements);
                }
            }

            public void search(int item) {
                if (root == null) {
                    System.out.println("(XXXXXX)");
                    return;
                }

                int heights = root.height, widths = (int) Math.pow(2, heights - 1);
                List < LeftHeapNode > currents = new ArrayList < LeftHeapNode > (1), nexts = new
ArrayList < LeftHeapNode > (2);
                currents.add(root);
                final int maxHalfLength = 4;
                int elements = 1;

                StringBuilder sbs = new StringBuilder(maxHalfLength * widths);
                for (int i = 0; i < maxHalfLength * widths; i++) {
                    sbs.append(' ');
                }

                String textBuffers;

                // Iterating through height levels.
                for (int i = 0; i < heights; i++) {

                    sbs.setLength(maxHalfLength * ((int) Math.pow(2, heights - 1 - i) - 1));

                    // Creating spacer space indicator.
                    textBuffers = sbs.toString();

                    // Print tree node elements
                    for (LeftHeapNode n: currents) {

                        // System.out.print(textBuffer);

                        if (n == null) {
                            // System.out.print("        ");
                            nexts.add(null);
                            nexts.add(null);
```

```
                } else {

                        // System.out.printf("(%6d)", n.element);
                        nexts.add(n.left);
                        nexts.add(n.right);
                }

                if (n!=null) {
                        if (n.element == item) {
                                System.out.println("Item found at height: "+heights);
                                System.exit(0);
                        }
                }


                // System.out.print(textBuffer);

        }

        System.out.println();

        // Print tree node extensions for next level.
        if (i < heights - 1) {

                for (LeftHeapNode n: currents) {
                        // System.out.print(textBuffer);

                        if (n == null) {
                                // System.out.print("        ");
                        } else {
                                 // System.out.print(textBuffer);
                                // System.out.printf("%s      %s", n.left == null ? " " : "/", n.right
== null ? " " : "\\");
                        }

                        if (n!=null) {
                                if (n.element == item) {
                                        System.out.println("Item found at height: "+heights);
                                        System.exit(0);
                                }
                        }


                }

                // System.out.println();

        }

        elements *= 2;
        currents = nexts;
        nexts = new ArrayList < LeftHeapNode > (elements);
    }
}
```

```
}

public class leftist {

	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);
		System.out.println("LeftistHeap Test\n\n");
		LeftistHeap lh = new LeftistHeap();
		int dataN;
		char ch;
		while (true) {
			System.out.println("\nLeftist Heap Operations\n");
			System.out.println("1. Insert ");
			System.out.println("2. Delete min");
			System.out.println("3. Search");
			int choice = scan.nextInt();
			switch (choice) {
				case 1:
					System.out.println("Enter integer element to insert");
					lh.insert(scan.nextInt());
					lh.inorder();
					break;
				case 2:
					lh.deleteMin();
					lh.inorder();
					break;
				case 3:
					System.out.print("Please provide the number to search: ");
					dataN = scan.nextInt();
					lh.search(dataN);
					break;
				case -1:
					break;
				default:
					System.out.println("Wrong Entry \n ");
					break;
			}
		}
	}
}
```

Explanation: The program adds and removes elements in the leftist heap. It also searches an element and provides the height at which it is located.

Ans 2)

```
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Scanner;

public class topology {
	static private class topologyNode {
		int nodeId;
		topologyNode next;

		public topologyNode(int id) {
			this.nodeId = id;
		}
	}

	ArrayList < topologyNode > nodeList;

	public topology() {
		nodeList = new ArrayList < topologyNode > ();
	}

	public topologyNode addNode(int id) {
		topologyNode node = new topologyNode(id);
		nodeList.add(0, node);

		return nodeList.get(0);
	}
	public void addEdge(int id1, int id2) {
		boolean node1Found = false,
			node2Found = false;
		topologyNode node1 = null,
			node2 = null;

		for (int i = 0; i < nodeList.size(); i++) {
			if (nodeList.get(i).nodeId == id1) {
				node1Found = true;
				node1 = nodeList.get(i);
			}
			if (nodeList.get(i).nodeId == id2) {
				node2Found = true;
				node2 = nodeList.get(i);
			}
			if (node1Found && node2Found) break;
		}

		if (!node1Found) {
			node1 = this.addNode(id1);
		}

		if (!node2Found) {
			node2 = this.addNode(id2);
		}
```

```
        topologyNode temp = new topologyNode(id2);
        temp.next = node1.next;
        node1.next = temp;

        return;
    }

    public topologyNode getNode(int id) {
        for (int i = 0; i < nodeList.size(); i++) {
            if (id == nodeList.get(i).nodeId) {
                return nodeList.get(i);
            }

        }

        return null;
    }

    public void printTopoSortedNodes() {
        Hashtable inDegrees = new Hashtable < Integer,
            Integer > ();
        ArrayList < topologyNode > zeroDegreeList = new ArrayList < topologyNode > ();
        ArrayList < topologyNode > nodes = this.nodeList;

        for (int i = 0; i < nodes.size(); i++) {
            topologyNode temp = nodes.get(i);
            temp = temp.next;
            while (temp != null) {
                int count = (inDegrees.get(temp.nodeId) == null) ? 0 : (int)
inDegrees.get(temp.nodeId);
                inDegrees.put(temp.nodeId, count + 1);
                temp = temp.next;
            }
        }

        for (int i = 0; i < nodes.size(); i++) {
            topologyNode temp = nodes.get(i);
            if (inDegrees.get(temp.nodeId) == null) {
                zeroDegreeList.add(0, temp);
            }

        }
        while (!zeroDegreeList.isEmpty()) {
            topologyNode curr = zeroDegreeList.remove(0);
            System.out.print(curr.nodeId + " ");

            topologyNode temp = curr.next;
            while (temp != null) {
                int prevInDegree = (int) inDegrees.get(temp.nodeId);

                inDegrees.put(temp.nodeId, prevInDegree - 1);
                if (prevInDegree == 1) {
                    zeroDegreeList.add(this.getNode(temp.nodeId));
```

```
                    }
                    temp = temp.next;
                }
            }

        }

        public static void main(String[] args) {

            topology gr = new topology();
            Scanner sc = new Scanner(System.in);
            while (true) {
                System.out.println("1. Insert data");
                int ch = sc.nextInt();
                if (ch == 1) {
                    System.out.print("ENTER the Source : ");
                    int src = sc.nextInt();
                    System.out.print("ENTER the Destination : ");
                    int dest = sc.nextInt();
                    gr.addEdge(src, dest);
                    gr.printTopoSortedNodes();
                }
            }
        }
}
```

Explanation: This is topological sort and the in degree calculation happens within the code