GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

LAB RECORDS

E2UC503C: Advanced Data Structures and Algorithms

Submitted By:

Admission Number		**YEAR : III**
Student Name		**SEMESTER : V**

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA ,INDIA

JANUARY 2024

LIST OF EXPERIMENTS

Experiment No: 1	Date:

Title

Write a JAVA program to implement Linear Search.

Algorithm

1) Read the value of n and elements of the array arr.
2) Read the key element to search.
3) Initialize a variable found as false to track if the element is found.
4) Loop through each element in the array:
   a) If the current element matches the key:
      i) Print the element found message with its position/index.
      ii) Set found to true.
   b) Otherwise, continue to the next element.
5) If found is still false after iterating through the entire array:
   a) Print an element not found message.

Program

```
import java.io.*;
class LinearSearch {
   public static void main(String args[]) throws IOException {
      int count = 0;
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("enter n value");
      int n = Integer.parseInt(br.readLine());
      int arr[] = new int[n];
      System.out.println("enter elements");
      for (int i = 0; i < n; i++) {
         arr[i] = Integer.parseInt(br.readLine());
      }
      System.out.println("enter element to search");
      int key = Integer.parseInt(br.readLine());
      for (int i = 0; i < n; i++) {
         if (arr[i] == key)
            System.out.println("element found : " + key + " in position :" + (i + 1));
         else
            count++;
      }
      if (count == n)
         System.out.println(key + " element not found, search failed");
   }
}
```

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java LinearSearch enter n value 4 enter elements 54 65 34 56 enter element to search 34 element found : 34 in position :3

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 2	Date:

Title

Write a JAVA program to implement Binary Search.

Algorithm

1. Initialize first to 0 and last to n-1.
2. Calculate the mid index as (first + last) / 2.
3. While first is less than or equal to last:
   - If the element at arr[mid] is less than key, update first = mid + 1.
   - If the element at arr[mid] is equal to key, print the index mid where the element is found and break the loop.
   - If the element at arr[mid] is greater than key, update last = mid - 1.
   - Recalculate mid as (first + last) / 2 within the loop.
4. If first becomes greater than last, print an "Element is not found!" message.

Program

```
class BinarySearch {
   public static void binarySearch(int arr[], int first, int last, int key) {
      int mid = (first + last) / 2;
      while (first <= last) {
         if (arr[mid] < key) {
            first = mid + 1;
         } else if (arr[mid] == key) {
            System.out.println("Element is found at index: " + mid);
            break;
         } else {
            last = mid - 1;
         }
         mid = (first + last) / 2;
      }
      if (first > last) {
         System.out.println("Element is not found!");
      }
   }

   public static void main(String args[]) {
      int arr[] = { 10, 20, 30, 40, 50 };
      int key = 30;
      int last = arr.length - 1;
      binarySearch(arr, 0, last, key);
   }
}
```

Sample Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java BinarySearch Element is found at index: 2

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 3	Date:

Title

Write a JAVA program to implement Binary Search Tree.

Algorithm

Insertion:

1. If the tree is empty (root is null), create a new node with the key and make it the root.
2. If the key is less than the root node's key:
3. Recursively call insertKey on the left subtree until a suitable empty spot is found.
4. Set the newly created node as the left child of the appropriate parent node.
5. If the key is greater than the root node's key:
6. Recursively call insertKey on the right subtree until a suitable empty spot is found.
7. Set the newly created node as the right child of the appropriate parent node.
8. Return the modified root.

Inorder Traversal:

If **root** is not **null**:

- Recursively traverse the left subtree.
- Print the key of the current node.
- Recursively traverse the right subtree.

Deletion:

- Search for the node with the **key** to be deleted:
  - If the **key** is less than the current **root**'s key, recursively search in the left subtree.
  - If the **key** is greater than the current **root**'s key, recursively search in the right subtree.
  - If the **key** matches the **root**'s key:
    - Handle cases based on the number of children:
      - Node has no children: Simply remove the node.
      - Node has one child: Replace the node with its child.
      - Node has two children: Find the inorder successor, replace node's value with successor's value, and delete the successor node.
- Return the modified **root**.

Program

```
class BinarySearchTree {
  class Node {
    int key;
    Node left, right;

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

```
Node root;
BinarySearchTree() {
  root = null;
}
void insert(int key) {
  root = insertKey(root, key);
}
Node insertKey(Node root, int key) {
  if (root == null) {
    root = new Node(key);
    return root;
  }
  if (key < root.key)
    root.left = insertKey(root.left, key);
  else if (key > root.key)
    root.right = insertKey(root.right, key);
  return root;
}
void inorder() {
  inorderRec(root);
}
void inorderRec(Node root) {
  if (root != null) {
    inorderRec(root.left);
    System.out.print(root.key + " -> ");
    inorderRec(root.right);
  }
}
void deleteKey(int key) {
  root = deleteRec(root, key);
}
Node deleteRec(Node root, int key) {
  if (root == null)
    return root;
  if (key < root.key)
    root.left = deleteRec(root.left, key);
  else if (key > root.key)
    root.right = deleteRec(root.right, key);
  else {
    if (root.left == null)
      return root.right;
    else if (root.right == null)
      return root.left;

    root.key = minValue(root.right);
    root.right = deleteRec(root.right, root.key);
  }
  return root;
}
int minValue(Node root) {
```

```
    int minv = root.key;
    while (root.left != null) {
      minv = root.left.key;
      root = root.left;
    }
    return minv;
  }
  public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(8);
    tree.insert(3);
    tree.insert(1);
    tree.insert(6);
    tree.insert(7);
    tree.insert(10);
    tree.insert(14);
    tree.insert(4);
    System.out.print("Inorder traversal: ");
    tree.inorder();
    System.out.println("\n\nAfter deleting 10");
    tree.deleteKey(10);
    System.out.print("Inorder traversal: ");
    tree.inorder();
  }
}
```

Sample Input & Output

```
PS C:\Users\anmol\OneDrive\Desktop\DSA File> java BinarySearchTree
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->

After deleting 10
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->
```

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 4	Date:

Title	Write a JAVA program to implement Tree Traversal.
Algorithm	Preorder Traversal: • If the node **ptr** is null, return. • Print the value of the current node **ptr**. • Recur on the left subtree (**ptr.left**). • Recur on the right subtree (**ptr.right**). Inorder Traversal: e **ptr** is null, return. he left subtree (**ptr.left**). alue of the current node **ptr**. he right subtree (**ptr.right**). Postorder Traversal: • If the node **ptr** is null, return. • Recur on the left subtree (**ptr.left**). • Recur on the right subtree (**ptr.right**). • Print the value of the current node **ptr**.

Program	class Node {
	int key;
	Node left, right;
	public Node(int item) {
	key = item;
	left = right = null;
	}
	}
	class BinaryTree {
	Node root;
	BinaryTree() {
	root = null;
	}
	void postorder(Node ptr) {
	if (ptr == null)
	return;
	postorder(ptr.left);
	postorder(ptr.right);
	System.out.print(ptr.key + " ");
	}

```
  void inorder(Node ptr) {
    if (ptr == null)
      return;
    inorder(ptr.left);
    System.out.print(ptr.key + " ");
    inorder(ptr.right);
  }
  void preorder(Node ptr) {
    if (ptr == null)
      return;
    System.out.print(ptr.key + " ");
    preorder(ptr.left);
    preorder(ptr.right);
  }
}

public class Tree_Traverse {
  public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);
    System.out.println("Preorder traversal");
    tree.preorder(tree.root);
    System.out.println("\nInorder traversal");
    tree.inorder(tree.root);
    System.out.println("\nPostorder traversal");
    tree.postorder(tree.root);
  }
}
```

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java Tree_Traverse Preorder traversal 1 2 4 5 3 Inorder traversal 4 2 5 1 3 Postorder traversal 4 5 2 3 1

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 5	Date:

Title	Write a JAVA program to implement Tower of Hanoi.

Algorithm	• If there is only one disk (**n == 1**), simply move it from **from_rod** to **to_rod**.
	• If there are more than one disk:
	  • Move **n-1** disks from the **from_rod** to the **helper_rod** using **to_rod** as a helper.
	  • Move the largest disk (**n**) from **from_rod** to **to_rod**.
	  • Move the **n-1** disks from the **helper_rod** to the **to_rod** using **from_rod** as a helper.

Program	public class Tower {
	  static void towerOfHanoi(int n, char from_rod, char to_rod, char helper_rod)
	  {
	  if (n == 1)
	  {
	  System.out.println("Take disk 1 from rod " + from_rod + " to rod " + to_rod);
	  return;
	  }
	  towerOfHanoi(n-1, from_rod, helper_rod, to_rod);
	  System.out.println("Take disk "+n+" from rod"+ from_rod+" to rod " + to_rod);
	  towerOfHanoi(n-1, helper_rod, to_rod, from_rod);
	  }
	
	  public static void main(String args[])
	  {
	  int n = 3;
	  towerOfHanoi(n,'A','C', 'B');
	  }
	}

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java Tower Take disk 1 from rod A to rod C Take disk 2 from rod A to rod B Take disk 1 from rod C to rod B Take disk 3 from rod A to rod C Take disk 1 from rod B to rod A Take disk 2 from rod B to rod C Take disk 1 from rod A to rod C

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 6	Date:

Title

Write a JAVA program to implement Stack.

Algorithm

1. **push(int x)**:
   - Create a new node with value **x**.
   - Set the new node's **next** to the current **top**.
   - Set the **top** to the new node.
   - Increment **nodesCount** to reflect the addition of a new node.
2. **isEmpty()**:
   - Check if the stack is empty by verifying if **top** is **null**.
3. **peek()**:
   - Check if the stack is empty. If empty, exit with an error.
   - Return the value of the top element without removing it.
4. **pop()**:
   - Check if the stack is empty. If empty, exit with an error.
   - Store the value of the top element.
   - Move **top** to the next node (removing the top element).
   - Decrement **nodesCount** to reflect the removal of the top node.
   - Return the stored top element's value.
5. **size()**:
   - Return the number of elements in the stack (**nodesCount**).

Program

```
class Node {
   int data;
   Node next;
}
class Stack {
   private Node top;
   private int nodesCount;
   public Stack() {
      this.top = null;
      this.nodesCount = 0;
   }
   public void push(int x) {
      Node node = new Node();
      node.data = x;
      node.next = top;
      top = node;
      this.nodesCount += 1;
   }
   public boolean isEmpty() {
      return top == null;
   }
   public int peek() {
      if (isEmpty()) {
```

```
            System.out.println("The stack is empty");
            System.exit(-1);
        }
        return top.data;
    }
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow");
            System.exit(-1);
        }
        int top = peek();
        this.nodesCount -= 1;
        this.top = this.top.next;
        return top;
    }
    public int size() {
        return this.nodesCount;
    }
}
class Main_Stack {
    public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        System.out.println("The top element is " + stack.peek());
        stack.pop();
        stack.pop();
        stack.pop();
        if (stack.isEmpty()) {
            System.out.println("The stack is empty");
        } else {
            System.out.println("The stack is not empty");
        }
    }
}
```

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java Main_Stack Inserting 1 Inserting 2 Inserting 3 The top element is 3 Removing 3 Removing 2 Removing 1 The stack is empty

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 7	Date:

Title

Write a JAVA program to implement Queue.

Algorithm

1. **enqueue(int item)**:
   - Create a new node with value **item**.
   - If the queue is empty:
     - Set both **front** and **rear** to the new node.
   - Otherwise:
     - Add the new node at the end (update **rear** and link the last node to the new node).
   - Increment **count**.
2. **dequeue()**:
   - If the queue is empty (i.e., **front** is **null**), exit with an error (underflow).
   - Remove the front element by moving **front** to the next node.
   - If the queue becomes empty, update **rear** to **null**.
   - Decrement **count**.
   - Return the removed element.
3. **peek()**:
   - If the queue is empty (i.e., **front** is **null**), exit with an error.
   - Return the value of the front element without removing it.
4. **isEmpty()**:
   - Check if both **rear** and **front** are **null** to determine if the queue is empty.
5. **size()**:
   - Return the value of **count**, indicating the number of elements in the queue.

Program

```
class Node {
   int data;
   Node next;

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

class Queue {
   private static Node rear = null, front = null;
   private static int count = 0;

   public static int dequeue() {
      if (front == null) {
         System.out.println("\nQueue Underflow");
         System.exit(-1);
```

```
        }
        Node temp = front;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        count -= 1;
        return temp.data;
    }

    public static void enqueue(int item) {
        Node node = new Node(item);
        if (rear == null) {
            front = node;
            rear = node;
        } else {
            rear.next = node;
            rear = node;
        }
        count += 1;
    }

    public static int peek() {
        if (front == null) {
            System.exit(-1);
        }
        return front.data;
    }

    public static boolean isEmpty() {
        return rear == null && front == null;
    }

    private static int size() {
        return count;
    }
}

class Main_Queue {
    public static void main(String[] args) {
        Queue q = new Queue();
        q.enqueue(1);
        q.enqueue(2);
        q.enqueue(3);
        q.enqueue(4);
        System.out.printf("The front element is %d\n", q.peek());
        q.dequeue();
        q.dequeue();
        q.dequeue();
        q.dequeue();
```

	if (q.isEmpty()) {
	System.out.println("The queue is empty");
	} else {
	System.out.println("The queue is not empty");
	}
	}
	}

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java Main_Queue Inserting 1 Inserting 2 Inserting 3 Inserting 4 The front element is 1 Removing 1 Removing 2 Removing 3 Removing 4 The queue is empty

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 8	Date:

Title	Write a JAVA program to implement Infix to Postfix.
Algorithm	1. Create two stacks: one for operators and another for the output (postfix expression). 2. Scan the infix expression from left to right. 3. If the scanned character is an operand, add it to the output. 4. If the scanned character is an open parenthesis '(', push it onto the operator stack. 5. If the scanned character is a close parenthesis ')': a. Pop and output all the operators from the stack until an open parenthesis '(' is encountered. b. Pop the open parenthesis from the stack (but don't output it). 6. If the scanned character is an operator: a. While the stack is not empty and the precedence of the current operator is less than or equal to the precedence of the operator at the top of the stack: i. Pop the operator from the stack and add it to the output. b. Push the current operator onto the stack. 7. Repeat steps 3-6 until all characters in the infix expression are scanned. 8. Pop and output any remaining operators from the stack to the output. 9. The output stack now contains the postfix expression.

Program	import java.util.Stack;
	public class InfixtoPostfix {
	public static int precedence(char ch) {
	if (ch == '*' || ch == '/')
	return 2;
	else if (ch == '+' || ch == '-')
	return 1;
	return 0;
	}
	public static String convertToPostfix(String exp) {
	Stack<Character> operators = new Stack<>();
	Stack<String> postFix = new Stack<>();
	for (int i = 0; i < exp.length(); i++) {
	char ch = exp.charAt(i);
	if (ch == '(')
	operators.push(ch);
	else if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
	postFix.push(ch + "");
	else if (ch == ')') {
	while (operators.peek() != '(') {
	char op = operators.pop();
	String first = postFix.pop();
	String second = postFix.pop();
	String new_postFix = second + first + op;
	postFix.push(new_postFix);
	}
	operators.pop();

```
      } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
        while (!operators.isEmpty() && operators.peek() != '(' && precedence(ch) <=
precedence(operators.peek())) {
          char op = operators.pop();
          String first = postFix.pop();
          String second = postFix.pop();
          String new_postFix = second + first + op;
          postFix.push(new_postFix);
        }
        operators.push(ch);
      }
    }
    while (!operators.isEmpty()) {
      char op = operators.pop();
      String first = postFix.pop();
      String second = postFix.pop();
      String new_postFix = second + first + op;
      postFix.push(new_postFix);
    }
    return postFix.pop();
  }
  public static void main(String args[]) {
    String infixExpression = "A*(B-C)/D+E";
    System.out.println("The Infix Expression is: " + infixExpression);
    String result = convertToPostfix(infixExpression);
    System.out.println("The Postfix of the given Infix Expression is: " + result);
    System.out.println();
    infixExpression = "a*(b-c+d)/e";
    System.out.println("The Infix Expression is: " + infixExpression);
    result = convertToPostfix(infixExpression);
    System.out.println("The Postfix of the given Infix Expression is: " + result);
  }
}
```

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java InfixtoPostfix The Infix Expression is: A*(B-C)/D+E The Postfix of the given Infix Expression is: ABC-*D/E+ The Infix Expression is: a*(b-c+d)/e The Postfix of the given Infix Expression is: abc-d+*e/

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 9	Date:

Title

Write a JAVA program to implement Kruskal.

Algorithm

- Sort edges in ascending order of weight.
- Initialize an empty list **mst** to store the MST.
- Initialize an index variable to 0.
- While the MST size is less than **vertices - 1**:
  - Remove the edge with the lowest weight from the priority queue.
  - Check if adding this edge creates a cycle:
    - Find the subsets of source and destination vertices.
    - If both subsets are different, add the edge to the MST and merge the subsets.
  - Increment the index.
- Print the MST.

Program

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

public class KruskalMST {
   static class Edge {
      int source;
      int destination;
      int weight;

      public Edge(int source, int destination, int weight) {
         this.source = source;
         this.destination = destination;
         this.weight = weight;
      }
   }

   static class Graph {
      int vertices;
      ArrayList<Edge> allEdges = new ArrayList<>();

      Graph(int vertices) {
         this.vertices = vertices;
      }

      public void addEgde(int source, int destination, int weight) {
         Edge edge = new Edge(source, destination, weight);
         allEdges.add(edge);
      }

      public void kruskalMST() {
```

```
        PriorityQueue<Edge> pq = new PriorityQueue<>(allEdges.size(),
Comparator.comparingInt(o -> o.weight));

        for (int i = 0; i < allEdges.size(); i++) {
            pq.add(allEdges.get(i));
        }

        int[] parent = new int[vertices];
        makeSet(parent);
        ArrayList<Edge> mst = new ArrayList<>();
        int index = 0;
        while (index < vertices - 1) {
            Edge edge = pq.remove();
            int x_set = find(parent, edge.source);
            int y_set = find(parent, edge.destination);
            if (x_set == y_set) {
                // ignore, will create cycle
            } else {
                mst.add(edge);
                index++;
                union(parent, x_set, y_set);
            }
        }
        System.out.println("Minimum Spanning Tree: ");
        printGraph(mst);
    }
    public void makeSet(int[] parent) {
        for (int i = 0; i < vertices; i++) {
            parent[i] = i;
        }
    }
    public int find(int[] parent, int vertex) {
        if (parent[vertex] != vertex)
            return find(parent, parent[vertex]);
        return vertex;
    }
    public void union(int[] parent, int x, int y) {
        int x_set_parent = find(parent, x);
        int y_set_parent = find(parent, y);
        parent[y_set_parent] = x_set_parent;
    }
    public void printGraph(ArrayList<Edge> edgeList) {
        for (int i = 0; i < edgeList.size(); i++) {
            Edge edge = edgeList.get(i);
            System.out.println("Edge-" + i + " source: " + edge.source +
                    " destination: " + edge.destination +
                    " weight: " + edge.weight);
        }
    }
}
```

```
    public static void main(String[] args) {
        int vertices = 6;
        Graph graph = new Graph(vertices);
        graph.addEgde(0, 1, 4);
        graph.addEgde(0, 2, 3);
        graph.addEgde(1, 2, 1);
        graph.addEgde(1, 3, 2);
        graph.addEgde(2, 3, 4);
        graph.addEgde(3, 4, 2);
        graph.addEgde(4, 5, 6);
        graph.kruskalMST();
    }
}
```

Sample Input & Output

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 10	Date:

Title

Write a JAVA program to implement Missing Value.

Algorithm

1. Take input for the total number of elements **n** in the array.
2. Read **n** elements into an array.
3. Calculate the expected sum of a sequence of integers from **1** to **n+1** (including the missing element) using the formula: **expectedSum = (n+1) * (n+2) / 2**.
4. Traverse through the given array and compute the sum of all the elements. Let's call it **actualSum**.
5. The missing element can be determined by subtracting **actualSum** from **expectedSum**: **missingElement = expectedSum - actualSum**.
6. Print or return the **missingElement**, which represents the missing value in the array.

Program

```
import java.util.Scanner;

public class Missing_Value {
   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      int n;
      System.out.println("Enter the total number of elements ");
      n = sc.nextInt();
      int arr[] = new int[n];
      System.out.println("Enter the elements of the array ");
      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextInt();
      }

      int sum = (n + 1) * (n + 2) / 2;
      for (int i = 0; i < n; i++) {
         sum = sum - arr[i];
      }
      System.out.println("Missing Element is " + sum);
   }
}
```

Sample Input & Output	C:\Users\drsak>java Missing_Value Enter the total number of elements 5 Enter the elements of the array 1 3 4 5 6 Missing Element is 2

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 11	Date:

Title	Write a JAVA program to implement Radix Sort.
Algorithm	1. Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place. 2. Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place. 3. Sort the elements based on the tens place digits. 4. Sort the elements based on the hundreds place digits. 5. The array is now sorted in ascending order.

Program	import java.util.Arrays;
	class RadixSort {
	
	  void countingSort(int array[], int size, int place) {
	  int[] output = new int[size + 1];
	  int max = array[0];
	  for (int i = 1; i < size; i++) {
	  if (array[i] > max)
	  max = array[i];
	  }
	  int[] count = new int[max + 1];
	
	  for (int i = 0; i < max; ++i)
	  count[i] = 0;
	
	  for (int i = 0; i < size; i++)
	  count[(array[i] / place) % 10]++;
	
	  for (int i = 1; i < 10; i++)
	  count[i] += count[i - 1];
	
	  for (int i = size - 1; i >= 0; i--) {
	  output[count[(array[i] / place) % 10] - 1] = array[i];
	  count[(array[i] / place) % 10]--;
	  }
	
	  for (int i = 0; i < size; i++)
	  array[i] = output[i];
	  }
	
	  int getMax(int array[], int n) {
	  int max = array[0];
	  for (int i = 1; i < n; i++)
	  if (array[i] > max)
	  max = array[i];

	return max;
	}
	
	void radixSort(int array[], int size) {
	int max = getMax(array, size);
	for (int place = 1; max / place > 0; place *= 10)
	countingSort(array, size, place);
	}
	
	public static void main(String args[]) {
	int[] data = { 121, 432, 564, 23, 17, 45, 788 };
	int size = data.length;
	RadixSort rs = new RadixSort();
	rs.radixSort(data, size);
	System.out.println("Sorted Array in Ascending Order: ");
	System.out.println(Arrays.toString(data));
	}
	}

Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java RadixSort Sorted Array in Ascending Order: [17, 23, 45, 121, 432, 564, 788]

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 12	Date:

Title

Write a JAVA program to implement Bitonic Sort.

Algorithm

1. Bitonic sequence is created.
2. Comparison between the corresponding element of the bitonic sequence.
3. Swapping the second element of the sequence.
4. Swapping the adjacent element.

Program

```
public class BitonicSort
{
	void compAndSwap(int a[], int i, int j, int dir)
	{
		if ( (a[i] > a[j] && dir == 1) ||
			(a[i] < a[j] && dir == 0))
		{
			// Swapping elements
			int temp = a[i];
			a[i] = a[j];
			a[j] = temp;
		}
	}
	void bitonicMerge(int a[], int low, int cnt, int dir)
	{
		if (cnt>1)
		{
			int k = cnt/2;
			for (int i=low; i<low+k; i++)
				compAndSwap(a,i, i+k, dir);
			bitonicMerge(a,low, k, dir);
			bitonicMerge(a,low+k, k, dir);
		}
	}
	void bitonicSort(int a[], int low, int cnt, int dir)
	{
		if (cnt>1)
		{
			int k = cnt/2;

			// sort in ascending order since dir here is 1
			bitonicSort(a, low, k, 1);

			// sort in descending order since dir here is 0
			bitonicSort(a,low+k, k, 0);

			// Will merge whole sequence in ascending order
			// since dir=1.
			bitonicMerge(a, low, cnt, dir);
```

```
                }
        }

        /*Caller of bitonicSort for sorting the entire array
        of length N in ASCENDING order */
        void sort(int a[], int N, int up)
        {
                bitonicSort(a, 0, N, up);
        }

        /* A utility function to print array of size n */
        static void printArray(int arr[])
        {
                int n = arr.length;
                for (int i=0; i<n; ++i)
                        System.out.print(arr[i] + " ");
                System.out.println();
        }

        // Driver method
        public static void main(String args[])
        {
                int a[] = {3, 7, 4, 8, 6, 2, 1, 5};
                int up = 1;
                BitonicSort ob = new BitonicSort();
                ob.sort(a, a.length,up);
                System.out.println("\nSorted array");
                printArray(a);
        }
}
```

Sample Input & Output

```
PS C:\Users\anmol\OneDrive\Desktop\DSA File> java BitonicSort

Sorted array
1 2 3 4 5 6 7 8
```

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 13	Date:

Title

Write a JAVA program to implement Repeat Frequency.

Algorithm

1. Take the size of the array **n** and the array elements as input.
2. Create an empty HashMap (**map**) to store elements as keys and their frequencies as values.
3. Iterate through the array.
4. For each element **a[i]**:
   i) If **a[i]** exists in the HashMap (**map**), increment its frequency value by 1.
   ii) If it doesn't exist, add **a[i]** to the HashMap with a frequency of 1.
5. Iterate through the HashMap.
6. For each entry in the HashMap:
   i) If the frequency of an element (**getValue()**) is greater than 1, print the element (**getKey()**) and its frequency.

Program

```
import java.util.*;
public class Repeat_Freq {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      int n;
      System.out.println("Enter the length of the array");
      n = sc.nextInt();

      int a[] = new int[n];
      System.out.println("Enter the array elements ");
      for (int i = 0; i < n; i++) {
         a[i] = sc.nextInt();
      }

      HashMap<Integer, Integer> map = new HashMap<>();
      for (int i = 0; i < n; i++) {
         if (map.containsKey(a[i])) {
            int c = map.get(a[i]);
            map.replace(a[i], c + 1);
         } else
            map.put(a[i], 1);
      }

      System.out.println("Elements  Frequency");
      for (Map.Entry<Integer, Integer> i : map.entrySet()) {
         if (i.getValue() > 1)
            System.out.println("  " + i.getKey() + "          " + i.getValue());
         else
            continue;
      }
```

	} }
Sample Input & Output	C:\Users\drsak>java Repeat_Freq Enter the length of the array 10 Enter the array elements 11 33 44 11 55 55 66 77 77 88 Elements Frequency 55 2 11 2 77 2

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 14	Date:

Title

Write a JAVA program to implement NQueen.

Algorithm

1) Start in the leftmost column
2) If all queens are placed return true
3) Try all rows in the current column. Do the following for every row.
   a) If the queen can be placed safely in this row
      i) Then mark this **[row, column]** as part of the solution and recursively check if placing queen here leads to a solution.
      ii) If placing the queen in **[row, column]** leads to a solution then return **true**.
      iii) If placing queen doesn't lead to a solution then unmark this **[row, column]** then backtrack and try other rows.
   b) If all rows have been tried and valid solution is not found return **false** to trigger backtracking.

Program

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


public class NQueen {
   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);
      int size = sc.nextInt();

      char[][] board = new char[size][size];
      for (int rowNo = 0; rowNo < size; rowNo++) {
         for (int colNo = 0; colNo < size; colNo++) {
            board[rowNo][colNo] = '.';
         }
      }
      ArrayList<ArrayList<String>> ans = new ArrayList<>();

      NQueen(size, board, 0, ans);

      if (ans.size() == 0) {
         System.out.println("No solution!");
      } else {
         System.out.println(ans);
      }



   }

   public static void NQueen(int size, char[][] b, int colNo,
ArrayList<ArrayList<String>> ans) {
```

```
        if (colNo == size) {
            ArrayList<String> al = new ArrayList<>();
            for (int rowNo = 0; rowNo < size; rowNo++) {
                char[] arr = b[rowNo];
                String s = new String(arr);
                al.add(s);
            }
            ans.add(al);
            return;
        }

        for (int rowNo = 0; rowNo < size; rowNo++) {

            if (isSafe(rowNo, colNo, b, size)) {

                b[rowNo][colNo] = 'Q';

                NQueen(size, b, colNo + 1, ans);

                b[rowNo][colNo] = '.';

            }
        }

        return;
    }

    public static boolean isSafe(int i, int j, char[][] b, int size) {

        for (int colNo = 0; colNo < j; colNo++) {
            if (b[i][colNo] == 'Q') {
                return false;
            }
        }
        int rowNo = i;
        int colNo = j;
        while (rowNo >= 0 && colNo >= 0) {
            if (b[rowNo][colNo] == 'Q') {
                return false;
            }
            rowNo--;
            colNo--;
        }
        colNo = j;
        rowNo = i;
        while (rowNo < size && colNo >= 0) {

            if (b[rowNo][colNo] == 'Q') {
                return false;
            }
```

```
            rowNo++;
            colNo--;
        }

        return true;

    }
}
```

Sample Input & Output

```
C:\Users\drsak>java NQueen
4
[[..Q., Q..., ...Q, .Q..], [.Q.., ...Q, Q..., ..Q.]]
```

Description	Mark
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.

Experiment No: 15	Date:

Title

Write a JAVA program to implement Trie – Insert and Search.

Algorithm

- **Insertion**:
  - Initialize a TrieNode structure with an array to hold children nodes and a boolean to mark the end of a word.
  - For each character in the input key:
    - Check if the character's node exists; if not, create a new node for it.
    - Move to the next node.
  - Mark the last node as the end of a word.

- **Search**:
  - Traverse through the Trie nodes for each character in the search key.
  - If any character's node doesn't exist, return **false**.
  - If the traversal completes and the last node is marked as the end of a word, return **true**; otherwise, return **false**.

Program

```
public class Trie {
   static final int ALPHABET_SIZE = 26;

   static class TrieNode {
      TrieNode[] children = new TrieNode[ALPHABET_SIZE];
      boolean isEndOfWord;

      TrieNode() {
         isEndOfWord = false;
         for (int i = 0; i < ALPHABET_SIZE; i++)
            children[i] = null;
      }
   };

   static TrieNode root;

   static void insert(String key) {
      int level;
      int length = key.length();
      int index;
      TrieNode pCrawl = root;

      for (level = 0; level < length; level++) {
         index = key.charAt(level) - 'a';
         if (pCrawl.children[index] == null)
            pCrawl.children[index] = new TrieNode();

         pCrawl = pCrawl.children[index];
      }
```

```
        pCrawl.isEndOfWord = true;
    }

    static boolean search(String key) {
        int level;
        int length = key.length();
        int index;
        TrieNode pCrawl = root;

        for (level = 0; level < length; level++) {
            index = key.charAt(level) - 'a';
            if (pCrawl.children[index] == null)
                return false;
            pCrawl = pCrawl.children[index];
        }
        return (pCrawl.isEndOfWord);
    }

    public static void main(String args[]) {
        String keys[] = {"the", "a", "there", "answer", "any", "by", "bye", "their", "tea"};
        String output[] = {"Not present in trie", "Present in trie"};
        root = new TrieNode();

        for (String key : keys)
            insert(key);

        if (search("the"))
            System.out.println("the --- " + output[1]);
        else
            System.out.println("the --- " + output[0]);

        if (search("these"))
            System.out.println("these --- " + output[1]);
        else
            System.out.println("these --- " + output[0]);

        if (search("their"))
            System.out.println("their --- " + output[1]);
        else
            System.out.println("their --- " + output[0]);

        if (search("thaw"))
            System.out.println("thaw --- " + output[1]);
        else
            System.out.println("thaw --- " + output[0]);

        if (search("tea"))
            System.out.println("tea --- " + output[1]);
        else
            System.out.println("tea --- " + output[0]);
```

	} }
Sample Input & Output	PS C:\Users\anmol\OneDrive\Desktop\DSA File> java Trie the --- Present in trie these --- Not present in trie their --- Present in trie thaw --- Not present in trie tea --- Present in trie

Description	**Mark**
Performance (2)	
Result (3)	
File (2)	
Viva (3)	
Total (10)	

Result :

The Program was executed and output Verified.