K.R. MANGALAM UNIVERSITY

DATA STRUCTURES LAB

Course Code: ENCS253

Course Type: Practical

Course Credit: 1.00

Faculty Name: Dr. Swati

Submitted By:

Aryan Mishra

Roll No: 2401010304

Section: B

B.Tech CSE

INDEX

	Pop, Peek, Display).
9	Evaluate Postfix Expression using Stack in Java.
10	Simulate Browser Back Button using Stack in Java.
11	Implement Bubble Sort in Java and analyse time complexity.
12	Implement Binary Search in Java and analyse time complexity.

Experiment 1: Inventory Management System in Java

AIM: To implement an inventory management system in Java that allows insertion of new products and display of all products using ArrayList.

Question: Write a menu-driven Java program to store product details (SKU, name, quantity) in an inventory. The program should allow the user to insert new products after validating quantity and to display the complete inventory in tabular form.

Introduction: Inventory management keeps track of items available in stock. Using ArrayList in Java, we can store a dynamic list of products where each product is represented as an object containing SKU, name and quantity.

Algorithm:

• Start.

• Create a Product class with fields sku, name and quantity.

• Use an ArrayList<Product> to store all product records.

• Display menu with options: 1. Insert product 2. Display inventory 3. Exit.

• For insertion: read sku, name and quantity; validate that quantity is positive and sku does not
already exist; then add new Product object to ArrayList.

• For display: if list is empty, show message; otherwise print all products in table form.

• Repeat menu until user selects Exit.

• Stop.

Java Code:

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

class Product {
    String sku;
    String name;
    int quantity;

    Product(String sku, String name, int quantity) {
        this.sku = sku;
        this.name = name;
        this.quantity = quantity;
    }
```

```java
}

public class InventoryManager {

    private static ArrayList<Product> inventory = new ArrayList<>();
    private static Scanner sc = new Scanner(System.in);

    public static void insertProduct() {
        System.out.print("Enter SKU: ");
        String sku = sc.nextLine();

        for (Product p : inventory) {
            if (p.sku.equalsIgnoreCase(sku)) {
                System.out.println("Product with this SKU already exists!");
                return;
            }
        }

        System.out.print("Enter Product Name: ");
        String name = sc.nextLine();

        if  (name.trim().isEmpty())  {
            System.out.println("Product name cannot be empty.");
            return;
        }

        System.out.print("Enter Quantity: ");
        int qty;
        try {
            qty = Integer.parseInt(sc.nextLine());
            if (qty <= 0) {
                System.out.println("Quantity must be positive.");
                return;
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid quantity.");
            return;
        }

        inventory.add(new Product(sku, name, qty));
        System.out.println("Product inserted successfully.");
    }

    public static void displayInventory() {
        if (inventory.isEmpty()) {
            System.out.println("Inventory is empty.");
            return;
        }
        System.out.println("SKU\t\tName\t\tQuantity");
        System.out.println("------------------------------------------------------------------------------------------");
        for (Product p : inventory) {
            System.out.println(p.sku + "\t\t" + p.name + "\t\t" + p.quantity);
        }
    }
    public static void main(String[] args) {
        while (true) {
            System.out.println("\nInventory Management System");
            System.out.println("1. Insert Product");
            System.out.println("2. Display Inventory");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            String ch = sc.nextLine();

            if (ch.equals("1")) {
                insertProduct();
            } else if (ch.equals("2")) {
                displayInventory();
            } else if (ch.equals("3")) {
                System.out.println("Exiting...");
                break;
            } else {
                System.out.println("Invalid choice.");
            }
        }
        sc.close();
    }
}
```

Output:

```
Enter SKU: P101
Enter Product Name: Pen
Enter Quantity: 50
Product inserted successfully.

Enter SKU: P102
Enter Product Name: Notebook
Enter Quantity: 30
Product inserted successfully.

Enter SKU: P101
Product with this SKU already exists!

Enter SKU: P103
Enter Product Name:
Product name cannot be empty.

Enter SKU: P104
Enter Product Name: Marker
Enter Quantity: -5
Quantity must be positive.

Enter SKU: P105
Enter Product Name: Pencil
Enter Quantity: abc
Invalid quantity.

Enter SKU: P106
Enter Product Name: Eraser
Enter Quantity: 20
Product inserted successfully.

---------------------------------------
Display Inventory
---------------------------------------
SKU         Name        Quantity
---------------------------------------
P101        Pen         50
```

P102	Notebook	30
P106	Eraser	20

Experiment 2: Inventory Stock Manager – Process Sales & Zero Stock (Java)

AIM: To implement an inventory stock manager in Java that processes sales for a given SKU and identifies items with zero stock.

Question: Write a Java program that maintains a list of items (SKU and quantity). Implement a method to process a sale given SKU and quantity sold, updating stock if available or showing appropriate error messages. Also implement a method to list all SKUs whose quantity becomes zero.

Algorithm:

• Use a class Item with fields sku and quantity.

• Store all items in ArrayList<Item>.

• For processSale(sku, qtySold): search the list for given sku.

• If sku not found: display message.

• If found and quantity >= qtySold: reduce quantity and show success message.

• If found and quantity < qtySold: do not update quantity and show insufficient stock message.

• For identifyZeroStock(): traverse list and collect all items whose quantity is 0 and display them.

Java Code:

```
import java.util.ArrayList;
import java.util.List;

class Item {
    int sku;
    int quantity;

    Item(int sku, int quantity) {
        this.sku = sku;
        this.quantity = quantity;
    }
}

public class InventorySalesManager {

    public static void processSale(List<Item> inventory, int sku, int qtySold)
{
        boolean found = false;
        for (Item item : inventory) {
            if (item.sku == sku) {
                found = true;
                if (item.quantity >= qtySold) {
                    item.quantity -= qtySold;
                    System.out.println("Sale processed: " + qtySold + " units
of SKU " + sku);
                } else {
                    System.out.println("Insufficient stock for SKU " + sku + ".
Available: " + item.quantity);
                }
                break;
            }
        }
        if (!found) {
            System.out.println("SKU " + sku + " not found in inventory.");
        }
    }

    public static List<Integer> identifyZeroStock(List<Item> inventory) {
        List<Integer> zeroList = new ArrayList<>();
        for (Item item : inventory) {
            if (item.quantity == 0) {
                zeroList.add(item.sku);
            }
        }
        if (zeroList.isEmpty()) {
            System.out.println("No zero stock items found.");
        } else {
            System.out.println("Zero stock SKUs: " + zeroList);
        }
        return zeroList;
    }

    public static void main(String[] args) {
        List<Item> inventory = new ArrayList<>();
        inventory.add(new Item(101, 50));
        inventory.add(new Item(102, 20));
        inventory.add(new Item(103, 0));

        processSale(inventory, 101, 30); // normal sale
        processSale(inventory, 102, 25); // insufficient stock
        processSale(inventory, 104, 10); // sku not found

        identifyZeroStock(inventory);

        System.out.print("Updated Inventory: ");
        for (Item item : inventory) {
            System.out.print("(" + item.sku + ", " + item.quantity + ") ");
        }
        System.out.println();
    }
}
```

Output:

```
Sale processed: 30 units of SKU 101
Insufficient stock for SKU 102. Available: 20
SKU 104 not found in inventory.
Zero stock SKUs: [103]
Updated Inventory: (101, 20) (102, 20) (103, 0)
```

Experiment 3: Linear Search in Java

AIM: To implement linear search in Java and analyse its best and worst case time complexity.

Question: Write a Java program to perform linear search on an array of integers. Also state the best and worst case time complexities of linear search.

Introduction: Linear search scans the array sequentially from left to right and compares each element with the key. It works on both sorted and unsorted arrays but it is inefficient for large datasets.

Algorithm:

• Input array A of n elements and key K.

• Set i = 0.

• While i < n, compare A[i] with K.

• If A[i] == K, return index i.

• Else increment i and continue loop.

• If no element matches, return -1 meaning key not found.

Java Code:

```java
import java.util.Scanner;

public class LinearSearch {

    public static int linearSearch(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i;
            }
        }
        return -1;
    }

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

        System.out.print("Enter element to search: ");
        int key = sc.nextInt();

        int index = linearSearch(arr, key);
        if (index == -1) {
            System.out.println("Element not found.");
        } else {
            System.out.println("Element found at index: " + index);
        }
        sc.close();
    }
}
```

Time Complexity: Best Case: O(1) when the key is at the first position. Worst Case: O(n) when the key is at the last position or not present in the array.

Output:

```
Enter number of elements: 5
Enter 5 elements:
4 7 1 9 3
Enter element to search: 9
Element found at index: 3
```

Experiment 4: Insertion in Circular Linked List (Beginning & End) in Java

AIM: To implement insertion at beginning and at end in a circular linked list using Java.

Question: Write a Java program to create a circular linked list and perform insertion of nodes at the beginning and at the end, displaying the list after each operation.

Java Code:

```java
class CNode {
    int data;
    CNode next;

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

class CircularLinkedListInsert {

    CNode head;

    public void insertAtEnd(int data) {
        CNode newNode = new CNode(data);
        if (head == null) {
            head = newNode;
            newNode.next = head;
            return;
        }
        CNode temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.next = head;
    }

    public void insertAtBeginning(int data) {
        CNode newNode = new CNode(data);
        if (head == null) {
            head = newNode;
            newNode.next = head;
            return;
        }
        CNode temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        newNode.next = head;
        temp.next = newNode;
        head = newNode;
    }

    public void display() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        CNode temp = head;
        System.out.print("Circular List: ");
        do {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        } while (temp != head);
        System.out.println("(back to head)");
    }

    public static void main(String[] args) {
        CircularLinkedListInsert cll = new CircularLinkedListInsert();
        cll.insertAtEnd(10);
        cll.insertAtEnd(20);
        cll.insertAtEnd(30);
        System.out.println("Original list:");
        cll.display();

        cll.insertAtBeginning(5);
        System.out.println("After inserting 5 at beginning:");
        cll.display();

        cll.insertAtEnd(40);
        System.out.println("After inserting 40 at end:");
        cll.display();
    }
}
```

Output:

```
Original list:
Circular List: 10-> 20-> 30-> (back to head)

After inserting 5 at beginning:
Circular List: 5-> 10-> 20-> 30-> (back to head)

After inserting 40 at end:
Circular List: 5-> 10-> 20-> 30-> 40-> (back to head)
```

Experiment 5: Deletion in Circular Linked List (Beginning & End) in Java

AIM: To delete a node from the beginning and from the end of a circular linked list using Java.

Java Code:

```
class CircularLinkedListDelete {

    CNode head;

    public void insert(int data) {
        CNode newNode = new CNode(data);
        if (head == null) {
            head = newNode;
            head.next = head;
            return;
        }
        CNode temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.next = head;
    }

    public void deleteFromBeginning() {
        if (head == null) {
            System.out.println("List is empty, nothing to delete.");
            return;
        }
        if (head.next == head) {
            head = null;
            System.out.println("Deleted the only node in the list.");
            return;
        }
        CNode last = head;
        while (last.next != head) {
            last = last.next;
        }
        head = head.next;
        last.next = head;
        System.out.println("Node deleted from beginning.");
    }

    public void deleteFromEnd() {
        if (head == null) {
            System.out.println("List is empty, nothing to delete.");
            return;
        }
        if (head.next == head) {
            head = null;
            System.out.println("Deleted the only node in the list.");
            return;
        }
        CNode prev = null;
        CNode temp = head;
        while (temp.next != head) {
            prev = temp;
            temp = temp.next;
        }
        prev.next = head;
        System.out.println("Node deleted from end.");
    }

    public void display() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        CNode temp = head;
        System.out.print("Circular List: ");
        do {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        } while (temp != head);
        System.out.println("(back to head)");
    }

    public static void main(String[] args) {
        CircularLinkedListDelete cll = new CircularLinkedListDelete();
        cll.insert(10);
        cll.insert(20);
        cll.insert(30);
        cll.insert(40);
        System.out.println("Initial list:");
        cll.display();

        cll.deleteFromBeginning();
        cll.display();

        cll.deleteFromEnd();
        cll.display();
    }
}
```

Output:

```
Initial list:
Circular List: 10-> 20-> 30-> 40-> (back to head)

Node deleted from beginning.
Circular List: 20-> 30-> 40-> (back to head)

Node deleted from end.
Circular List: 20-> 30-> (back to head)
```

Experiment 6: Deletion from Singly Linked List (Beginning & End) in Java

AIM: To implement deletion of nodes from the beginning and end of a singly linked list using Java.

Java Code:

```
class SNode {
    int data;
    SNode next;

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

class SinglyLinkedListDelete {

    SNode head;

    public void insert(int data) {
        SNode newNode = new SNode(data);
        if (head == null) {
            head = newNode;
            return;
        }
        SNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    public void deleteFromBeginning() {
        if (head == null) {
            System.out.println("List is empty!");
            return;
        }
        head = head.next;
        System.out.println("Node deleted from beginning.");
    }

    public void deleteFromEnd() {
        if (head == null) {
            System.out.println("List is empty!");
            return;
        }
        if (head.next == null) {
            head = null;
            System.out.println("Last node deleted.");
            return;
        }
        SNode temp = head;
        while (temp.next.next != null) {
            temp = temp.next;
        }
        temp.next = null;
        System.out.println("Node deleted from end.");
    }

    public void display() {
        if (head == null) {
            System.out.println("List is empty!");
            return;
        }
        SNode temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        SinglyLinkedListDelete list = new SinglyLinkedListDelete();
        list.insert(10);
        list.insert(20);
        list.insert(30);
        list.insert(40);

        System.out.println("Original List:");
        list.display();

        list.deleteFromBeginning();
        System.out.println("After deleting from beginning:");
        list.display();

        list.deleteFromEnd();
        System.out.println("After deleting from end:");
        list.display();
    }
}
```

Output:

```
Original List:
10-> 20-> 30-> 40-> null

Node deleted from beginning.
After deleting from beginning:
20-> 30-> 40-> null

Node deleted from end.
After deleting from end:
20-> 30-> null
```

Experiment 7: Circular Queue using Array in Java

AIM: To implement a circular queue using array in Java with enqueue, dequeue and display operations.

Java Code:

```
import java.util.Scanner;

class CircularQueue {

    private int[] arr;
    private int front;
    private int rear;
    private int size;

    public CircularQueue(int capacity) {
        arr = new int[capacity];
        front = -1;
        rear = -1;
        size = capacity;
    }

    public boolean isEmpty() {
        return front == -1;
    }

    public boolean isFull() {
        return (front == 0 && rear == size - 1) || (rear + 1 == front);
    }

    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue is full.");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % size;
        arr[rear] = value;
        System.out.println("Enqueued: " + value);
    }

    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return -1;
        }
        int element = arr[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % size;
        }
        System.out.println("Dequeued: " + element);
        return element;
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Elements in queue: ");
        int i = front;
        while (true) {
            System.out.print(arr[i] + " ");
            if (i == rear) break;
            i = (i + 1) % size;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularQueue q = new CircularQueue(5);
        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.enqueue(40);
        q.display();
        q.dequeue();
        q.dequeue();
        q.display();
        q.enqueue(50);
        q.enqueue(60);
        q.display();
        sc.close();
    }
}
```

Output:

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40

Elements in queue: 10 20 30 40

Dequeued: 10
Dequeued: 20

Elements in queue: 30 40

Enqueued: 50
Enqueued: 60

Elements in queue: 30 40 50 60
```

Experiment 8: Stack using Array in Java (Push, Pop, Peek, Display)

AIM: To implement stack operations push, pop, peek and display using array in Java.

Java Code:

```
import java.util.Scanner;

class ArrayStack {

    private int[] stack;
    private int top;
    private int capacity;

    public ArrayStack(int capacity) {
        this.capacity = capacity;
        stack = new int[capacity];
        top = -1;
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == capacity - 1;
    }

    public void push(int item) {
        if (isFull()) {
            System.out.println("Stack overflow.");
            return;
        }
        stack[++top] = item;
        System.out.println("Pushed " + item);
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack underflow.");
            return -1;
        }
        int item = stack[top--];
        System.out.println("Popped " + item);
        return item;
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return -1;
        }
        System.out.println("Top element is " + stack[top]);
        return stack[top];
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
        }
        System.out.print("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayStack st = new ArrayStack(10);
        while (true) {
            System.out.println("\\n1. Push  2. Pop  3. Peek  4. Display  5.
Exit");
            System.out.print("Enter your choice: ");
            int ch = sc.nextInt();
            if (ch == 1) {
                System.out.print("Enter element: ");
                int x = sc.nextInt();
                st.push(x);
            } else if (ch == 2) {
                st.pop();
            } else if (ch == 3) {
                st.peek();
            } else if (ch == 4) {
                st.display();
            } else if (ch == 5) {
                System.out.println("Exiting.");
                break;
            } else {
                System.out.println("Invalid choice.");
            }
        }
        sc.close();
    }
}
```

Output:

```
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter element: 10
Pushed 10

1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter element: 20
Pushed 20
```

```
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 4
Stack elements: 10 20

1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 3
Top element is 20

1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 2
Popped 20

1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 5
Exiting.
```

Experiment 9: Evaluate Postfix Expression using Stack in Java

AIM: To evaluate a postfix arithmetic expression using stack in Java.

Java Code:

```java
import java.util.Scanner;
import java.util.Stack;

public class PostfixEvaluation {

    public static int applyOperation(int op1, int op2, char operator) {
        switch (operator) {
            case '+': return op1 + op2;
            case '-': return op1 - op2;
            case '*': return op1 * op2;
            case '/': return op1 / op2;
            default: throw new IllegalArgumentException("Invalid operator");
        }
    }

    public static int evaluatePostfix(String expression) {
        Stack<Integer> stack = new Stack<>();
        String[] tokens = expression.split("\\s+");
        for (String token : tokens) {
            if (token.matches("\\d+")) {
                stack.push(Integer.parseInt(token));
            } else {
                int op2 = stack.pop();
                int op1 = stack.pop();
                int result = applyOperation(op1, op2, token.charAt(0));
                stack.push(result);
            }
        }
        return stack.pop();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter postfix expression (space separated): ");
        String expr = sc.nextLine();
        int result = evaluatePostfix(expr);
        System.out.println("Result = " + result);
        sc.close();
    }
}
```

Output:

```
Enter postfix expression (space separated): 10 2 8 * + 3 -
Result = 23
```

Experiment 10: Browser Back Button Simulation using Stack in Java

AIM: To simulate a browser back button using stack in Java.

Question: Write a Java program that allows the user to visit pages, go back to the previous page and show history using stack operations.

Java Code:

```java
import java.util.Scanner;
import java.util.Stack;

public class BrowserBackSimulation {

    public static void main(String[] args) {
        Stack<String> history = new Stack<>();
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("\\n1. Visit Page");
            System.out.println("2. Back");
            System.out.println("3. Show History");
            System.out.println("4. Exit");
            System.out.print("Enter  choice:  ");
            int ch = sc.nextInt();
            sc.nextLine(); // consume newline

            if (ch == 1) {
                System.out.print("Enter page name: ");
                String page = sc.nextLine();
                history.push(page);
                System.out.println("Visited: " + page);
            } else if (ch == 2) {
                if  (history.isEmpty())  {
                    System.out.println("No pages in history.");
                } else {
                    String last = history.pop();
                    System.out.println("Going back from: " + last);
                    if (history.isEmpty()) {
                        System.out.println("No pages left in history.");
                    } else {
                        System.out.println("Current page: " + history.peek());
                    }
                }
            } else if (ch == 3) {
                if (history.isEmpty()) {
                    System.out.println("History is empty.");
                } else {
                    System.out.println("History: " + history);
                }
            } else if (ch == 4) {
                System.out.println("Exiting browser simulation.");
                break;
            } else {
                System.out.println("Invalid choice.");
            }
        }
        sc.close();
    }
}
```

Output:

```
1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: Google
Visited: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: YouTube
Visited: YouTube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: GitHub
Visited: GitHub

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 3
History: [Google, YouTube, GitHub]

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: GitHub
Current page: YouTube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: YouTube
Current page: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
```

```
Going back from: Google
No pages left in history.

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 4
Exiting browser simulation.
```

Experiment 11: Bubble Sort in Java

AIM: To implement bubble sort in Java and analyse its time complexity.

Question: Write a Java program to sort an array of integers using bubble sort technique and display the sorted array.

Java Code:

```java
import java.util.Scanner;

public class BubbleSort {

    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            boolean swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        bubbleSort(arr);
        System.out.println("Sorted array:");
        for (int x : arr) {
            System.out.print(x + " ");
        }
        System.out.println();
        sc.close();
    }
}
```

Output:

```
Enter number of elements: 6
Enter 6 elements:
5 1 4 2 8 3
Sorted array:
1 2 3 4 5 8
```

Time Complexity: Best Case: O(n) when array is already sorted and inner loop breaks early. Worst and Average Case: O(n^2).

Experiment 12: Binary Search in Java

AIM: To implement binary search in Java and analyse its time complexity.

Question: Write a Java program to perform binary search on a sorted array of integers and find the position of a given key element.

Java Code:

```java
import java.util.Scanner;

public class BinarySearch {

    public static int binarySearch(int[] arr, int key) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) {
                return mid;
            } else if (arr[mid] < key) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter " + n + " sorted elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter key to search: ");
        int key = sc.nextInt();
        int index = binarySearch(arr, key);
        if (index == -1) {
            System.out.println("Element not found.");
        } else {
            System.out.println("Element found at index: " + index);
        }
        sc.close();
    }
}
```

Output:

```
Enter number of elements: 6
Enter 6 sorted elements:
2 5 8 12 16 20
Enter key to search: 12
Element found at index: 3
```

Time Complexity: Best, average and worst case time complexity of binary search is O(log n).