==========String program==========

🔹 1. Palindrome Check

👉 Problem: Check if "madam" is palindrome.

public class PalindromeCheck { public static void main(String[] args) { String str = "madam"; String rev = "";

```
   // build reverse string
    for (int i = str.length() - 1; i >= 0; i--) {
        rev += str.charAt(i);
    }

    if (str.equals(rev))
        System.out.println("Palindrome");
    else
        System.out.println("Not Palindrome");
}


}
```

✅ Explanation: Compare original string with reversed string. ✅ Time Complexity: $O(n^2)$ (string concatenation in loop) 👉 Can improve with StringBuilder.append() to O(n). ✅ Space Complexity: O(n)

🔹 2. Count Vowels & Consonants

👉 Problem: Count vowels and consonants in "Hello World".

public class CountVowels { public static void main(String[] args) { String str = "Hello World".toLowerCase(); int vowels = 0, consonants = 0;

```
   for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);

        if (c >= 'a' && c <= 'z') {   // only alphabets
            if ("aeiou".indexOf(c) != -1) vowels++;
            else consonants++;
        }
    }
```

```
    System.out.println("Vowels: " + vowels + ", Consonants: " +
consonants);
}


}
```

✅ Explanation: Convert to lowercase → check each char → if vowel, count it. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🔹 3. Find Duplicate Characters

👉 Problem: Find duplicate characters in "programming".

import java.util.*;

public class DuplicateCharacters { public static void main(String[] args) { String str = "programming"; Map<Character, Integer> map = new HashMap<>();

```
   for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        map.put(c, map.getOrDefault(c, 0) + 1);
    }

    System.out.println("Duplicate characters:");
    for (Map.Entry<Character, Integer> entry : map.entrySet()) {
        if (entry.getValue() > 1)
            System.out.println(entry.getKey() + " : " +
entry.getValue());
    }
}


}
```

✅ Explanation: Use a HashMap to count frequencies, print chars > 1. ✅ Time Complexity: O(n) ✅ Space Complexity: O(k) (unique characters)

🔹 4. Implement strStr() (Substring Search)

👉 Problem: Find if "ll" exists in "hello". Return starting index.

public class SubstringSearch { public static int strStr(String haystack, String needle) { int n = haystack.length(); int m = needle.length();

```
    for (int i = 0; i <= n - m; i++) {
        if (haystack.substring(i, i + m).equals(needle)) {
            return i;
        }
    }
    return -1;
}

public static void main(String[] args) {
    System.out.println(strStr("hello", "ll"));  // Output: 2
}


}
```

✅ Explanation: Check substrings of haystack and compare with needle. ✅ Time Complexity: O((n-m+1) * m) ✅ Space Complexity: O(m)

🔹 5. Anagram Check (Extra – Very Common)

👉 Problem: Check if two strings are anagrams (contain same letters). Example: "listen" and "silent".

import java.util.Arrays;

public class AnagramCheck { public static void main(String[] args) { String s1 = "listen"; String s2 = "silent";

```
    char[] a1 = s1.toCharArray();
    char[] a2 = s2.toCharArray();

    Arrays.sort(a1);
    Arrays.sort(a2);

    if (Arrays.equals(a1, a2))
        System.out.println("Anagram");
    else
        System.out.println("Not Anagram");
}


}
```

✅ Explanation: Sort both strings and compare. ✅ Time Complexity: O(n log n) ✅ Space Complexity: O(n)

🔹 6. Reverse Words in a String (Extra – Average Level)

👉 Problem: "I love Java" → "Java love I"

public class ReverseWords { public static void main(String[] args) { String str = "I love Java"; String[] words = str.split(" "); String rev = "";

```
   for (int i = words.length - 1; i >= 0; i--) {
        rev += words[i] + " ";
    }

    System.out.println("Reversed Words: " + rev.trim());
}


}
```

✅ Explanation: Split string into words → reverse order → join. ✅ Time Complexity: O(n) ✅ Space Complexity: O(n)

🎯 Final Suggestion for B.Tech Final Year Students

Must prepare these 6 problems:

Palindrome ✅

Count vowels & consonants ✅

Duplicate characters ✅

Substring search (strStr) ✅

Anagram ✅

Reverse words in string ✅

==========ArrayProgram==========

🔹 1. Find Maximum and Minimum in Array

👉 Problem: Given [3, 5, 1, 9, 2], find max and min.

public class ArrayMaxMin { public static void main(String[] args) { int[] arr = {3, 5, 1, 9, 2}; int max = arr[0], min = arr[0];

```
    for (int i = 1; i < arr.length; i++) {
         if (arr[i] > max) max = arr[i];
         if (arr[i] < min) min = arr[i];
     }

     System.out.println("Max: " + max + ", Min: " + min);
}


}
```

✅ Explanation: Compare each element with current max/min. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🔹 2. Reverse an Array

👉 Problem: Reverse [1, 2, 3, 4, 5] → [5, 4, 3, 2, 1].

public class ReverseArrayNew { public static void main(String[] args) { int[] arr = {1, 2, 3, 4, 5}; int n = arr.length;

```
    int[] reversed = new int[n];
     for (int i = 0; i < n; i++) {
         reversed[i] = arr[n - 1 - i]; // Copy from end to start
     }

     for (int num : reversed) System.out.print(num + " ");
}


}
```

public class ReverseArray { public static void main(String[] args) { int[] arr = {1, 2, 3, 4, 5}; int left = 0, right = arr.length - 1;

```
    while (left < right) {
         int temp = arr[left];
         arr[left] = arr[right];
         arr[right] = temp;
```

```
        left++;
        right--;
    }

    for (int num : arr) System.out.print(num + " ");
}


}
```

✅ Explanation: Swap elements from both ends until middle. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🔹 3. Find Duplicate Elements

👉 Problem: Find duplicates in [1, 2, 3, 2, 4, 1].

import java.util.*;

public class DuplicateInArray { public static void main(String[] args) { int[] arr = {1, 2, 3, 2, 4, 1}; Map<Integer, Integer> freq = new HashMap<>();

```
   for (int num : arr) {
        freq.put(num, freq.getOrDefault(num, 0) + 1);
    }

    System.out.println("Duplicate elements:");
    for (Map.Entry<Integer, Integer> e : freq.entrySet()) {
        if (e.getValue() > 1)
            System.out.println(e.getKey());
    }
}


}
```

✅ Explanation: Count frequency using HashMap. Print numbers with frequency > 1. ✅ Time Complexity: O(n) ✅ Space Complexity: O(n)

🔹 4. Sum of Array / Prefix Sum

👉 Problem: Find sum of [10, 20, 30, 40].

public class SumArray { public static void main(String[] args) { int[] arr = {10, 20, 30, 40}; int sum = 0;

```
    for (int num : arr) sum += num;

     System.out.println("Sum = " + sum);
}


}
```

✅ Explanation: Add all elements. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🔹 5. Find Second Largest Element

👉 Problem: [10, 5, 20, 8] → second largest = 10.

public class SecondLargest { public static void main(String[] args) { int[] arr = {10, 5, 20, 8}; int first = Integer.MIN_VALUE, second = Integer.MIN_VALUE;

```
    for (int num : arr) {
         if (num > first) {
             second = first;
             first = num;
         } else if (num > second && num != first) {
             second = num;
         }
     }

     System.out.println("Second Largest: " + second);
}


}
```

✅ Explanation: Track first largest and second largest. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🔹 6. Check if Array is Sorted

👉 Problem: [1, 2, 3, 4, 5] → true.

public class CheckSorted { public static void main(String[] args) { int[] arr = {1, 2, 3, 4, 5};
boolean sorted = true;

```
   for (int i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            sorted = false;
            break;
        }
    }

    System.out.println("Is Sorted? " + sorted);
}


}
```

✅ Explanation: Compare each element with previous. ✅ Time Complexity: O(n) ✅ Space Complexity: O(1)

🎯 Must-Do Array Questions (Service-Based Interviews)

Find max/min ✅

Reverse array ✅

Find duplicates ✅

Array sum / prefix sum ✅

Second largest ✅

Check if sorted ✅

==========StackProgram==========

LIFO (Stack) – Important Programs for Interviews

1. Valid Parentheses Check ✅

Problem: Check if a string containing brackets ()[]{} is valid (every opening has a matching closing in correct order).

Java Solution:

import java.util.Stack;

public class ValidParentheses { public static boolean isValid(String s) { Stack stack = new Stack<>();

```
   for (char ch : s.toCharArray()) {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        } else {
            if (stack.isEmpty()) return false;
            char top = stack.pop();
            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
                (ch == ']' && top != '[')) {
                return false;
            }
        }
    }
    return stack.isEmpty();
}

public static void main(String[] args) {
    System.out.println(isValid("()[]{}")); // true
    System.out.println(isValid("([)]"));   // false
}


}
```

Time Complexity: O(n) Space Complexity: O(n) (stack storage)

2. Implement Stack Using Array 🧱

Problem: Build your own stack using array with push, pop, and peek.

Java Solution:

class MyStack { private int[] arr; private int top, capacity;

```
MyStack(int size) {
    arr = new int[size];
    capacity = size;
    top = -1;
}
```

```
public void push(int x) {
    if (top == capacity - 1) {
        System.out.println("Stack Overflow!");
        return;
    }
    arr[++top] = x;
}

public int pop() {
    if (top == -1) {
        System.out.println("Stack Underflow!");
        return -1;
    }
    return arr[top--];
}

public int peek() {
    if (top == -1) return -1;
    return arr[top];
}

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


}
```

public class StackDemo { public static void main(String[] args) { MyStack stack = new MyStack(5); stack.push(10); stack.push(20); stack.push(30); System.out.println("Top: " + stack.peek()); // 30 System.out.println("Popped: " + stack.pop()); // 30 System.out.println("Top: " + stack.peek()); // 20 } }

Time Complexity: O(1) per operation Space Complexity: O(n)

3. Next Greater Element 🔥

Problem: For each element in the array, find the next greater element (NGE) to its right. If none, put -1.

Java Solution:

import java.util.Stack;

public class NextGreaterElement { public static void main(String[] args) { int[] arr = {4, 5, 2, 25}; int n = arr.length; int[] result = new int[n]; Stack stack = new Stack<>();

```
   for (int i = n - 1; i >= 0; i--) {
        while (!stack.isEmpty() && stack.peek() <= arr[i]) {
            stack.pop();
        }
        result[i] = stack.isEmpty() ? -1 : stack.peek();
        stack.push(arr[i]);
    }

    for (int x : result) System.out.print(x + " ");
}


}
```

Output: 5 25 25 -1 Time Complexity: O(n) Space Complexity: O(n)

4. Stock Span Problem 📈

Problem: For each day, calculate how many consecutive days before it had stock price <= today's price.

Java Solution:

import java.util.Stack;

public class StockSpan { public static void calculateSpan(int[] price, int n) { int[] span = new int[n]; Stack stack = new Stack<>();

```
   stack.push(0);
    span[0] = 1;

    for (int i = 1; i < n; i++) {
        while (!stack.isEmpty() && price[stack.peek()] <= price[i])
{
            stack.pop();
        }
        span[i] = (stack.isEmpty()) ? (i + 1) : (i - stack.peek());
        stack.push(i);
    }
```

```
    for (int x : span) System.out.print(x + " ");
}

public static void main(String[] args) {
    int[] price = {100, 80, 60, 70, 60, 75, 85};
    calculateSpan(price, price.length);
}


}
```

Output: 1 1 1 2 1 4 6 Time Complexity: O(n) Space Complexity: O(n)

5. Reverse a Stack Using Recursion 🔁

Problem: Reverse elements of stack without using another data structure.

Java Solution:

import java.util.Stack;

public class ReverseStack { public static void insertAtBottom(Stack stack, int x) { if (stack.isEmpty()) { stack.push(x); return; } int top = stack.pop(); insertAtBottom(stack, x); stack.push(top); }

```
public static void reverse(Stack<Integer> stack) {
    if (!stack.isEmpty()) {
        int top = stack.pop();
        reverse(stack);
        insertAtBottom(stack, top);
    }
}

public static void main(String[] args) {
    Stack<Integer> stack = new Stack<>();
    stack.push(1); stack.push(2); stack.push(3);

    System.out.println("Original: " + stack);
    reverse(stack);
    System.out.println("Reversed: " + stack);
}
```

}

Time Complexity: $O(n^2)$ (because each insertion is O(n)) Space Complexity: O(n) (recursion stack)

👉 Kaif bhai, these 5 programs cover almost all common stack interview questions:

Parentheses validation ✅

Stack implementation ✅

Next Greater Element ✅

Stock Span ✅

Reverse Stack ✅

==========QueueProgram==========

1. Simple Queue Implementation using Array class SimpleQueue { private int[] arr; private int front, rear, size, capacity;

public SimpleQueue(int capacity) { this.capacity = capacity; arr = new int[capacity]; front = 0; rear = -1; size = 0; }

// Enqueue public void enqueue(int x) { if (size == capacity) { System.out.println("Queue is full"); return; } rear = (rear + 1) % capacity; // circular shift arr[rear] = x; size++; }

// Dequeue public int dequeue() { if (size == 0) { System.out.println("Queue is empty"); return -1; } int item = arr[front]; front = (front + 1) % capacity; size--; return item; }

public void printQueue() { for (int i = 0; i < size; i++) { System.out.print(arr[(front + i) % capacity] + " "); } System.out.println(); }

}

public class QueueDemo { public static void main(String[] args) { SimpleQueue q = new SimpleQueue(5); q.enqueue(10); q.enqueue(20); q.enqueue(30); q.printQueue(); // 10 20 30 q.dequeue(); q.printQueue(); // 20 30 } }

✅ Time Complexity:

Enqueue = O(1)

Dequeue = O(1) ✅ Space Complexity: O(n)

2. Queue using Linked List class Node { int data; Node next; Node(int data) { this.data = data; this.next = null; } }

class LinkedQueue { Node front, rear;

```
public void enqueue(int x) {
    Node temp = new Node(x);
    if (rear == null) {
        front = rear = temp;
        return;
    }
    rear.next = temp;
    rear = temp;
}

public int dequeue() {
    if (front == null) {
        System.out.println("Queue is empty");
        return -1;
    }
    int item = front.data;
    front = front.next;
    if (front == null) rear = null;
    return item;
}

public void printQueue() {
    Node temp = front;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}


}
```

public class LinkedQueueDemo { public static void main(String[] args) { LinkedQueue q = new LinkedQueue(); q.enqueue(5); q.enqueue(15); q.enqueue(25); q.printQueue(); // 5 15 25 q.dequeue(); q.printQueue(); // 15 25 } }

✅ Time Complexity: O(1) enqueue, O(1) dequeue ✅ Space Complexity: O(n)

3. Queue using Two Stacks (Interview Favorite)

👉 Idea: Enqueue = push into stack1, Dequeue = pop from stack2 (transfer elements if empty).

import java.util.Stack;

class QueueUsingStacks { Stack s1 = new Stack<>(); Stack s2 = new Stack<>();

```
public void enqueue(int x) {
    s1.push(x);
}

public int dequeue() {
    if (s1.isEmpty() && s2.isEmpty()) {
        System.out.println("Queue is empty");
        return -1;
    }
    if (s2.isEmpty()) {
        while (!s1.isEmpty()) {
            s2.push(s1.pop());
        }
    }
    return s2.pop();
}


}
```

public class QueueStacksDemo { public static void main(String[] args) { QueueUsingStacks q = new QueueUsingStacks(); q.enqueue(1); q.enqueue(2); q.enqueue(3); System.out.println(q.dequeue()); // 1 System.out.println(q.dequeue()); // 2 } }

✅ Time Complexity:

Enqueue = O(1)

Dequeue = Amortized O(1) (worst O(n) when shifting) ✅ Space Complexity: O(n)

4. Circular Queue (Interview Standard)

👉 A circular queue avoids wasted space in a simple queue implementation.

class CircularQueue { int[] arr; int front, rear, size, capacity;

```
CircularQueue(int capacity) {
    this.capacity = capacity;
    arr = new int[capacity];
    front = rear = -1;
}

boolean isFull() {
    return (front == 0 && rear == capacity - 1) || (rear == (front -
1) % (capacity - 1));
}

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

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

int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty");
        return -1;
    }
    int item = arr[front];
    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front + 1) % capacity;
    }
    return item;
}
```

```
void printQueue() {
    if (isEmpty()) return;
    int i = front;
    while (true) {
        System.out.print(arr[i] + " ");
        if (i == rear) break;
        i = (i + 1) % capacity;
    }
    System.out.println();
}


}
```

public class CircularQueueDemo { public static void main(String[] args) { CircularQueue q = new CircularQueue(5); q.enqueue(10); q.enqueue(20); q.enqueue(30); q.printQueue(); // 10 20 30 q.dequeue(); q.printQueue(); // 20 30 } }

✅ Time Complexity: O(1) ✅ Space Complexity: O(n)

🔥 Extra Queue Problems for Interviews

Implement Deque (Double Ended Queue) → Insert/remove from both ends.

BFS Traversal (Graph/Tree) → Queue is must.

Sliding Window Maximum → Uses deque.

==========LinkedListProgram==========

Linked List (Important Programs for Interview) ✅ 1. Implement a Simple Singly Linked List

👉 Every fresher should know how to create a Linked List and print it.

class Node { int data; Node next;

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

```
}

public class LinkedListDemo { Node head;

// Insert at end
public void insert(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        return;
    }
    Node temp = head;
    while (temp.next != null) temp = temp.next;
    temp.next = newNode;
}

// Print list
public void display() {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " -> ");
        temp = temp.next;
    }
    System.out.println("null");
}

public static void main(String[] args) {
    LinkedListDemo list = new LinkedListDemo();
    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.display();  // 10 -> 20 -> 30 -> null
}


}
```

✅ 2. Reverse a Linked List (Most Asked)

👉 Interviewer favorite → simple iterative solution using pointers.

public Node reverse(Node head) { Node prev = null, curr = head; while (curr != null) { Node next = curr.next; // store next curr.next = prev; // reverse link prev = curr; // move prev curr = next; // move curr } return prev; }

📌 Explanation: We keep 3 pointers (prev, curr, next) → update links in each step.

✅ 3. Find Middle of Linked List (Fast/Slow pointer trick) public Node findMiddle(Node head) { Node slow = head, fast = head; while (fast != null && fast.next != null) { slow = slow.next; // move 1 step fast = fast.next.next; // move 2 steps } return slow; // middle node }

📌 Explanation: slow moves 1 step, fast moves 2 steps. When fast reaches end → slow at middle.

✅ 4. Detect Cycle in Linked List (Floyd's Algorithm)

👉 Common in interview because it uses fast & slow pointer.

public boolean hasCycle(Node head) { Node slow = head, fast = head; while (fast != null && fast.next != null) { slow = slow.next; fast = fast.next.next; if (slow == fast) return true; // cycle found } return false; }

✅ 5. Remove Duplicates from Sorted Linked List public Node removeDuplicates(Node head) { Node curr = head; while (curr != null && curr.next != null) { if (curr.data == curr.next.data) curr.next = curr.next.next; // skip duplicate else curr = curr.next; } return head; }

✅ 6. Merge Two Sorted Linked Lists public Node merge(Node l1, Node l2) { if (l1 == null) return l2; if (l2 == null) return l1;

```
if (l1.data < l2.data) {
    l1.next = merge(l1.next, l2);
    return l1;
} else {
    l2.next = merge(l1, l2.next);
    return l2;
}


}
```

🎯 Summary (For Interviews)

👉 Frequently asked linked list problems for final year / fresher interviews:

Create and print a linked list.

Reverse a linked list.

Find middle element (fast/slow pointer).

Detect cycle in linked list.

Remove duplicates from sorted list.

Merge two sorted linked lists.

==========HashingProgram==========

4. Hashing in Java

Concept: Uses HashMap / HashSet to store elements in key-value form → average O(1) search/insert. Usage: Frequency counting, duplicates, searching problems. Interview Topics:

2-Sum problem

Count frequency of elements

Longest substring without repeating characters

✅ Program 1: 2-Sum Problem using HashMap

Find two numbers in an array that add up to a target.

```
import java.util.*;

public class TwoSum { public static void main(String[] args) { int[] nums = {2, 7, 11, 15};
int target = 9;

   Map<Integer, Integer> map = new HashMap<>(); // value -> index

    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i]; // check if pair exists
        if (map.containsKey(complement)) {
            System.out.println("Indices: " + map.get(complement) + "
and " + i);
            return;
```

```
            }
            map.put(nums[i], i); // store value with index
        }
}


}
```

👉 Output: Indices: 0 and 1 (because 2 + 7 = 9) 📌 Important: HashMap makes this O(n) instead of $O(n^2)$.

✅ Program 2: Count Frequency of Elements

Use HashMap to count occurrences of each element.

import java.util.*;

public class FrequencyCount { public static void main(String[] args) { int[] arr = {1, 2, 2, 3, 1, 4, 2};

```
   Map<Integer, Integer> freq = new HashMap<>();

    for (int num : arr) {
        freq.put(num, freq.getOrDefault(num, 0) + 1);
    }

    for (Map.Entry<Integer, Integer> entry : freq.entrySet()) {
        System.out.println(entry.getKey() + " -> " +
entry.getValue());
    }
}


}
```

👉 Output:

1 -> 2 2 -> 3 3 -> 1 4 -> 1

📌 Common interview question: “Find the most frequent element”.

✅ Program 3: Check for Duplicates using HashSet import java.util.*;

public class ContainsDuplicate { public static void main(String[] args) { int[] arr = {1, 2, 3, 4, 2};

```
   Set<Integer> set = new HashSet<>();
    boolean duplicate = false;

    for (int num : arr) {
        if (!set.add(num)) { // add returns false if element already
exists
            duplicate = true;
            break;
        }
    }

    System.out.println("Contains duplicate? " + duplicate);
}


}
```

👉 Output: Contains duplicate? true

✅ Program 4: Longest Substring Without Repeating Characters (Sliding Window + HashSet) import java.util.*;

public class LongestSubstring { public static void main(String[] args) { String s = "abcabcbb"; Set set = new HashSet<>();

```
   int left = 0, maxLen = 0;

    for (int right = 0; right < s.length(); right++) {
        while (set.contains(s.charAt(right))) {
            set.remove(s.charAt(left));
            left++;
        }
        set.add(s.charAt(right));
        maxLen = Math.max(maxLen, right - left + 1);
    }

    System.out.println("Longest length = " + maxLen);
}
```

}

👉 Output: Longest length = 3 (substring "abc") 📌 Asked frequently in interviews (Amazon, Microsoft, etc).

✅ So for Hashing, these are the must-do programs for B.Tech / interview prep:

2-Sum Problem

Frequency Count

Detect Duplicates

Longest Substring without Repeating Characters

==========Searching and sorting==========

Searching and sorting

Sorting Algorithms

1. Bubble Sort (Simple but slow)

👉 Compare and swap adjacent elements.

public class BubbleSort { public static void main(String[] args) { int[] arr = {5, 3, 8, 4, 2};

```
   for (int i = 0; i < arr.length - 1; i++) {
        for (int j = 0; j < arr.length - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    for (int num : arr) System.out.print(num + " ");
}


}
```

2. Insertion Sort

👉 Build sorted array one by one.

public class InsertionSort { public static void main(String[] args) { int[] arr = {5, 3, 8, 4, 2};

```
    for (int i = 1; i < arr.length; i++) {
         int key = arr[i];
         int j = i - 1;
         while (j >= 0 && arr[j] > key) {
             arr[j + 1] = arr[j];
             j--;
         }
         arr[j + 1] = key;
     }

     for (int num : arr) System.out.print(num + " ");
}


}
```

3. Merge Sort (Divide and Conquer)

👉 Recursive sorting (O(n log n)).

public class MergeSort { public static void merge(int[] arr, int l, int m, int r) { int n1 = m - l + 1, n2 = r - m; int[] L = new int[n1], R = new int[n2];

```
    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
     for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

     int i = 0, j = 0, k = l;
     while (i < n1 && j < n2) {
         arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
     }
     while (i < n1) arr[k++] = L[i++];
     while (j < n2) arr[k++] = R[j++];
}

public static void sort(int[] arr, int l, int r) {
     if (l < r) {
         int m = (l + r) / 2;
         sort(arr, l, m);
```

```
        sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

public static void main(String[] args) {
    int[] arr = {5, 3, 8, 4, 2};
    sort(arr, 0, arr.length - 1);
    for (int num : arr) System.out.print(num + " ");
}


}
```

4. Quick Sort (Divide and Conquer, very important)

👉 Partition-based sorting.

public class QuickSort { public static int partition(int[] arr, int low, int high) { int pivot = arr[high], i = low - 1; for (int j = low; j < high; j++) { if (arr[j] < pivot) { i++; int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp; } } int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp; return i + 1; }

```
public static void sort(int[] arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        sort(arr, low, pi - 1);
        sort(arr, pi + 1, high);
    }
}

public static void main(String[] args) {
    int[] arr = {5, 3, 8, 4, 2};
    sort(arr, 0, arr.length - 1);
    for (int num : arr) System.out.print(num + " ");
}


}
```

Searching Algorithms

1. Linear Search (Simple)

👉 Check one by one.

public class LinearSearch { public static void main(String[] args) { int[] arr = {5, 3, 8, 4, 2}; int key = 8, found = -1;

```
   for (int i = 0; i < arr.length; i++) {
        if (arr[i] == key) {
            found = i;
            break;
        }
    }

    System.out.println((found == -1) ? "Not Found" : "Found at index
" + found);
}


}
```

2. Binary Search (Sorted array only, very important in interviews)

👉 Divide and search (O(log n)).

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) {
    int[] arr = {2, 3, 4, 5, 8};
    int key = 5;
    int result = binarySearch(arr, key);
    System.out.println((result == -1) ? "Not Found" : "Found at
index " + result);
}


}
```

✅ Most Asked Interview Variations

Binary Search → First Occurrence / Last Occurrence.

Binary Search → Search in Rotated Sorted Array.

Sorting → Quick Sort & Merge Sort (very important).

Searching + Sorting together → "Two-Sum with Sorting + Two Pointers".

==========Recursion==========

Recursion

1. Factorial of a Number public class FactorialRec { static int factorial(int n) { if (n == 0) return 1; // base case return n * factorial(n - 1); }

public static void main(String[] args) { System.out.println(factorial(5)); // 120 }

}

2. Fibonacci Series public class FibonacciRec { static int fib(int n) { if (n <= 1) return n; // base case return fib(n - 1) + fib(n - 2); }

public static void main(String[] args) { for (int i = 0; i < 7; i++) System.out.print(fib(i) + " "); }

}

3. Print All Subsets of String/Array public class Subsets { static void printSubsets(String s, String ans, int i) { if (i == s.length()) { System.out.println(ans); return; } printSubsets(s, ans, i + 1); // exclude printSubsets(s, ans + s.charAt(i), i + 1); // include }

public static void main(String[] args) { printSubsets("abc", "", 0); }

}

4. Tower of Hanoi public class TowerOfHanoi { static void solve(int n, char from, char to, char aux) { if (n == 0) return; solve(n - 1, from, aux, to); System.out.println("Move disk " + n + " from " + from + " to " + to); solve(n - 1, aux, to, from); }

public static void main(String[] args) { solve(3, 'A', 'C', 'B'); }

}

✅ For Freshers, the most expected recursion interview programs are:

Factorial

Fibonacci

Subsets / Power Set

Tower of Hanoi

==========divide and conquer -Binary Search==========

divide and conquer -Binary Search ✅ 1. Binary Search on Integers

👉 Works only on sorted array.

public class BinarySearchInt { public static int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1;

```
   while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) return mid;   // Found
        else if (arr[mid] < target) left = mid + 1; // Search right half
        else right = mid - 1;  // Search left half
    }
    return -1;  // Not found
}

public static void main(String[] args) {
    int[] nums = {10, 20, 30, 40, 50};
    int target = 30;

    int result = binarySearch(nums, target);
    if (result != -1)
        System.out.println("Found at index " + result);
    else
        System.out.println("Not found");
}


}
```

✅ 2. Binary Search on Non-Primitive Objects (Student)

We use Comparator or Comparable for custom sorting. Example: Search a student by roll number.

import java.util.Arrays;

class Student implements Comparable { int rollNo; String name;

```
Student(int rollNo, String name) {
    this.rollNo = rollNo;
    this.name = name;
}

// Sorting by roll number
@Override
public int compareTo(Student other) {
    return Integer.compare(this.rollNo, other.rollNo);
}


}
```

public class BinarySearchStudent { public static int binarySearch(Student[] students, int rollNo) { int left = 0, right = students.length - 1;

```
   while (left <= right) {
        int mid = left + (right - left) / 2;

        if (students[mid].rollNo == rollNo) return mid;
        else if (students[mid].rollNo < rollNo) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

public static void main(String[] args) {
    Student[] students = {
        new Student(101, "Aman"),
        new Student(102, "Kaif"),
        new Student(103, "Rohit"),
        new Student(104, "Sneha")
    };

    // Array must be sorted before binary search
```

```
    Arrays.sort(students);

    int rollToFind = 103;
    int index = binarySearch(students, rollToFind);

    if (index != -1)
        System.out.println("Found Student: " +
students[index].name);
    else
        System.out.println("Student not found!");
}


}
```

🚀 Key Points for Interview:

Works only on sorted data.

Time Complexity: O(log n).

Can be applied to objects if we define Comparable/Comparator.

Common variations: First occurrence, last occurrence, rotated sorted array.

==========Backtracking==========

Backtracking

Example: Rat in a Maze (Simple Backtracking)

👉 Problem: A rat is at the top-left of an N x N maze.

1 means open path,

0 means blocked path. Find a path from start (0,0) to end (N-1,N-1) by moving only right or down.

✅ Implementation in Java public class RatInMaze { static int N = 4;

```
// Function to print solution matrix
static void printSolution(int sol[][]) {
    for (int i = 0; i < N; i++) {
```

```
        for (int j = 0; j < N; j++)
            System.out.print(sol[i][j] + " ");
        System.out.println();
    }
}

// Utility function to check valid move
static boolean isSafe(int maze[][], int x, int y) {
    return (x < N && y < N && maze[x][y] == 1);
}

// Solve maze using backtracking
static boolean solveMaze(int maze[][]) {
    int sol[][] = new int[N][N];

    if (solveMazeUtil(maze, 0, 0, sol) == false) {
        System.out.println("No solution exists");
        return false;
    }

    printSolution(sol);
    return true;
}

// Recursive backtracking function
static boolean solveMazeUtil(int maze[][], int x, int y, int
sol[][]) {
    // base case: reached destination
    if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
        sol[x][y] = 1;
        return true;
    }

    // check if maze[x][y] is safe
    if (isSafe(maze, x, y)) {
        sol[x][y] = 1;

        // move right
        if (solveMazeUtil(maze, x + 1, y, sol))
            return true;

        // move down
```

```
        if (solveMazeUtil(maze, x, y + 1, sol))
            return true;

        // backtrack
        sol[x][y] = 0;
        return false;
    }

    return false;
}

public static void main(String args[]) {
    int maze[][] = {
            { 1, 0, 0, 0 },
            { 1, 1, 0, 1 },
            { 0, 1, 0, 0 },
            { 1, 1, 1, 1 }
    };

    solveMaze(maze);
}


}
```

🔹 Output (Path Matrix) 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1

✅ This shows how backtracking explores paths and backtracks if stuck. 📌 Even if N-Queens or Sudoku look fancy, this simple program is enough to show your backtracking knowledge in an interview.

==========Bit Manipulation==========

Bit Manipulation

9. Bit Manipulation What it is

Working with data at bit level using operators like &, |, ^, ~, <<, >>.

Usage

Optimize space & speed

Low-level operations

Checking, setting, toggling bits

Common Interview Questions (for Freshers)

1. Check if a number is odd or even

👉 Logic:

If last bit (n & 1) is 0 → even

If last bit is 1 → odd

public class OddEvenCheck { public static void main(String[] args) { int n = 7; if ((n & 1) == 0) System.out.println(n + " is Even"); else System.out.println(n + " is Odd"); } }

2. Find element that appears once when all others appear twice

👉 Use XOR (^) → a ^ a = 0, a ^ 0 = a

public class SingleNumber { public static void main(String[] args) { int arr[] = {2, 3, 5, 4, 5, 3, 2}; int result = 0; for (int num : arr) { result ^= num; } System.out.println("Single number: " + result); } }

3. Swap two numbers without temp variable

👉 Using XOR

public class SwapNumbers { public static void main(String[] args) { int a = 5, b = 7; a = a ^ b; b = a ^ b; a = a ^ b; System.out.println("a = " + a + ", b = " + b); } }

4. Count number of 1s in binary (set bits)

👉 Brian Kernighan's Algorithm

public class CountSetBits { public static void main(String[] args) { int n = 13; // 1101 int count = 0; while (n > 0) { n = n & (n - 1); // clears the lowest set bit count++; } System.out.println("Set bits: " + count); } }

✅ Most asked for freshers:

Odd/Even check

Single number using XOR

Swap without temp

⭐ If you prepare these 3, you're safe in service-based interviews.

==========Miscellaneous Important Topics + Simple Java Programs==========

Miscellaneous Important Topics + Simple Java Programs

1. Palindrome Problems

👉 Very common in interviews.

Palindrome String

Palindrome Number

// Palindrome Number public class PalindromeCheck { public static void main(String[] args) { int num = 121, rev = 0, temp = num; while (num > 0) { rev = rev * 10 + num % 10; num /= 10; } System.out.println(temp == rev ? "Palindrome" : "Not Palindrome"); } }

2. Armstrong Number

👉 Popular number theory question.

public class Armstrong { public static void main(String[] args) { int num = 153, sum = 0, temp = num; while (num > 0) { int digit = num % 10; sum += digit * digit * digit; num /= 10; } System.out.println(sum == temp ? "Armstrong" : "Not Armstrong"); } }

3. Factorial & Fibonacci

👉 Recursion + Iterative both are asked.

// Factorial (Recursion) static int fact(int n) { if (n == 0) return 1; return n * fact(n - 1); }

// Fibonacci (Iterative) static void fibonacci(int n) { int a = 0, b = 1; for (int i = 0; i < n; i++) { System.out.print(a + " "); int next = a + b; a = b; b = next; } }

4. Prime Number Check public static boolean isPrime(int n) { if (n <= 1) return false; for (int i = 2; i <= Math.sqrt(n); i++) { if (n % i == 0) return false; } return true; }
5. Reverse String / Number

👉 Classic interview starter.

// Reverse String String str = "hello"; String rev = new StringBuilder(str).reverse().toString(); System.out.println(rev);

// Reverse Number int num = 1234, rev = 0; while (num > 0) { rev = rev * 10 + num % 10; num /= 10; } System.out.println(rev);

6. Anagram Check (Strings)

👉 Asked many times.

import java.util.Arrays; class Anagram { public static void main(String[] args) { String s1 = "listen", s2 = "silent"; char[] a = s1.toCharArray(), b = s2.toCharArray(); Arrays.sort(a); Arrays.sort(b); System.out.println(Arrays.equals(a, b) ? "Anagram" : "Not Anagram"); } }

7. Duplicate Characters in String import java.util.HashMap; class DuplicateChar { public static void main(String[] args) { String s = "programming"; HashMap<Character, Integer> map = new HashMap<>(); for (char c : s.toCharArray()) map.put(c, map.getOrDefault(c, 0) + 1); map.forEach((k,v) -> { if (v > 1) System.out.println(k + " -> " + v); }); }

}

8. Swapping Without Temp Variable int a = 5, b = 10; a = a + b;
   b = a - b;
   a = a - b; System.out.println(a + " " + b);
9. Factor Count / Divisors

👉 Helps in prime + GCD questions.

int num = 28; for (int i = 1; i <= num; i++) { if (num % i == 0) System.out.print(i + " "); }

10. LCM and GCD // Euclidean Algorithm static int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }

static int lcm(int a, int b) { return (a * b) / gcd(a, b); }

11. Number-based Programs

👉 Very commonly asked in service-based interviews.

Check Prime Number

public class PrimeCheck { public static void main(String[] args) { int num = 17; boolean isPrime = true; for (int i = 2; i <= Math.sqrt(num); i++) { if (num % i == 0) { isPrime = false; break; } } System.out.println(num + " is prime? " + isPrime); } }

Factorial of a Number (Iterative + Recursive)

public class Factorial { static int fact(int n) { if (n == 0) return 1; return n * fact(n-1); } public static void main(String[] args) { System.out.println("Factorial of 5 = " + fact(5)); } }

Palindrome Number

public class PalindromeNum { public static void main(String[] args) { int num = 121, rev = 0, temp = num; while (num > 0) { rev = rev*10 + num%10; num /= 10; } System.out.println(temp == rev ? "Palindrome" : "Not Palindrome"); } }

12. Pattern Printing Programs

👉 Asked to test logic + loops.

Triangle Pattern

-

**

public class Pattern { public static void main(String[] args) { int n = 4; for (int i = 1; i <= n; i++) { for (int j = 1; j <= i; j++) { System.out.print("*"); } System.out.println(); } } }

13. String-based Programs

👉 Very common in interviews.

Check String Palindrome

public class PalindromeString { public static void main(String[] args) { String str = "madam"; String rev = new StringBuilder(str).reverse().toString(); System.out.println(str.equals(rev) ? "Palindrome" : "Not Palindrome"); } }

Count Vowels and Consonants

public class VowelCount { public static void main(String[] args) { String str = "hello world"; int vowels = 0, consonants = 0; str = str.toLowerCase(); for (char ch :

str.toCharArray()) { if ("aeiou".indexOf(ch) != -1) vowels++; else if (ch >= 'a' && ch <= 'z') consonants++; } System.out.println("Vowels: " + vowels + ", Consonants: " + consonants); } }

14. Miscellaneous Programs

Swap 2 Numbers Without Temp

public class SwapNumbers { public static void main(String[] args) { int a = 5, b = 10; a = a ^ b; b = a ^ b; a = a ^ b; System.out.println("a = " + a + ", b = " + b); } }

Fibonacci Series (Iterative)

public class Fibonacci { public static void main(String[] args) { int n = 10, a = 0, b = 1; for (int i = 0; i < n; i++) { System.out.print(a + " "); int temp = a + b; a = b; b = temp; } } }

✅ These simple yet common Java programs cover:

Numbers

Strings

Patterns

Bitwise tricks

👉 For fresher interviews, most questions revolve around these + arrays/strings we already discussed.

✅ These 10 programs are the most asked in fresher interviews (especially service-based companies). They test your basics, logic building, and coding style.

==========LeetCode==========

LeetCode

1. Two Sum (LeetCode #1)

Problem: Given an array of integers, return indices of the two numbers such that they add up to a target.

👉 Why important? Frequently asked in interviews, tests hash map knowledge.

import java.util.*;

public class TwoSum { public static int[] twoSum(int[] nums, int target) { Map<Integer, Integer> map = new HashMap<>(); for(int i = 0; i < nums.length; i++) { int remaining = target - nums[i]; if(map.containsKey(remaining)) { return new int[]{map.get(remaining), i}; } map.put(nums[i], i); } return new int[]{}; }

```
public static void main(String[] args) {
    int[] nums = {2, 7, 11, 15};
    int target = 9;
    int[] result = twoSum(nums, target);
    System.out.println(result[0] + " " + result[1]);  // Output: 0 1
}


}
```

2. Valid Parentheses (LeetCode #20)

Problem: Given a string containing just ()[]{}, check if it's valid.

👉 Why important? Tests stack concept, commonly asked.

import java.util.*;

public class ValidParentheses { public static boolean isValid(String s) { Stack stack = new Stack<>(); for(char c : s.toCharArray()) { if(c == '(') stack.push(')'); else if(c == '{') stack.push('}'); else if(c == '[') stack.push(']'); else if(stack.isEmpty() || stack.pop() != c) return false; } return stack.isEmpty(); }

```
public static void main(String[] args) {
    System.out.println(isValid("()[]{}")); // true
    System.out.println(isValid("("));     // false
}


}
```

3. Maximum Subarray (Kadane’s Algorithm, LeetCode #53)

Problem: Find the contiguous subarray with the largest sum.

👉 Why important? Easy DP-based problem asked often.

public class MaxSubArray { public static int maxSubArray(int[] nums) { int maxSoFar = nums[0]; int currMax = nums[0]; for(int i = 1; i < nums.length; i++) { currMax =

Math.max(nums[i], currMax + nums[i]); maxSoFar = Math.max(maxSoFar, currMax); } return maxSoFar; }

```
public static void main(String[] args) {
    int[] nums = {-2,1,-3,4,-1,2,1,-5,4};
    System.out.println(maxSubArray(nums)); // Output: 6 (subarray
[4,-1,2,1])
}


}
```

4. Merge Two Sorted Lists (LeetCode #21)

Problem: Merge two sorted linked lists into one sorted list.

👉 Why important? Tests linked list basics.

class ListNode { int val; ListNode next; ListNode(int x) { val = x; } }

public class MergeSortedLists { public static ListNode mergeTwoLists(ListNode l1, ListNode l2) { if(l1 == null) return l2; if(l2 == null) return l1;

```
   if(l1.val < l2.val) {
        l1.next = mergeTwoLists(l1.next, l2);
        return l1;
    } else {
        l2.next = mergeTwoLists(l1, l2.next);
        return l2;
    }
}

public static void main(String[] args) {
    ListNode l1 = new ListNode(1);
    l1.next = new ListNode(2);
    l1.next.next = new ListNode(4);

    ListNode l2 = new ListNode(1);
    l2.next = new ListNode(3);
    l2.next.next = new ListNode(4);

    ListNode merged = mergeTwoLists(l1, l2);
    while(merged != null) {
        System.out.print(merged.val + " ");
```

```
        merged = merged.next;
    }
    // Output: 1 1 2 3 4 4
}


}
```

5. Best Time to Buy and Sell Stock (LeetCode #121)

Problem: Find the maximum profit from stock prices.

👉 Why important? Simple array problem testing min/max logic.

public class BuySellStock { public static int maxProfit(int[] prices) { int minPrice = Integer.MAX_VALUE; int maxProfit = 0; for(int price : prices) { if(price < minPrice) minPrice = price; else if(price - minPrice > maxProfit) maxProfit = price - minPrice; } return maxProfit; }

```
public static void main(String[] args) {
    int[] prices = {7,1,5,3,6,4};
    System.out.println(maxProfit(prices)); // Output: 5 (buy at 1,
sell at 6)
}


}
```

Problem 3: Reverse a String

LeetCode #344

- Problem: Given a string, reverse it.

Example:

Input: "hello" Output: "olleh"

Solution (Java):

class Solution { public String reverseString(String s) { char[] arr = s.toCharArray(); int left = 0, right = arr.length - 1;

```
   while (left < right) {
        char temp = arr[left];
```

```
        arr[left] = arr[right];
        arr[right] = temp;
        left++;
        right--;
    }
    return new String(arr);
}


}
```

✅ Explanation: Use two pointers, swap characters from left and right until they meet.

Problem 4: Palindrome Number

LeetCode #9

🔹 Problem: Check if a number is palindrome (same forward and backward).

Example:

Input: 121 Output: true

Input: 123 Output: false

Solution (Java):

class Solution { public boolean isPalindrome(int x) { if (x < 0) return false; // negative numbers can't be palindrome

```
   int original = x;
    int reversed = 0;

    while (x != 0) {
        int digit = x % 10;
        reversed = reversed * 10 + digit;
        x /= 10;
    }
    return original == reversed;
}


}
```

✅ Explanation: Reverse the number and compare with the original.

Problem 5: Valid Anagram

LeetCode #242

🔹 Problem: Check if two strings are anagrams (contain same letters, just rearranged).

Example:

Input: s = "listen", t = "silent" Output: true

Solution (Java):

import java.util.*;

class Solution { public boolean isAnagram(String s, String t) { if (s.length() != t.length()) return false;

```
   int[] count = new int[26];

    for (char c : s.toCharArray())
        count[c - 'a']++;

    for (char c : t.toCharArray()) {
        count[c - 'a']--;
        if (count[c - 'a'] < 0) return false;
    }

    return true;
}


}
```

✅ Explanation: Count frequency of each character and compare.

✨ These 5 are super important for freshers:

Two Sum → Hashing

Valid Parentheses → Stack

Max Subarray → DP basics

Merge Two Sorted Lists → Linked list

Buy/Sell Stock → Array logic

==========Complete==========
==========