

## # Q1 two sum problem()

```
import java.util.Scanner;

public class TwoSumSimpleInput {
    public static int[] twoSum(int[] nums, int target) {
        // Brute-force approach
        for (int i = 0; i < nums.length; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[i] + nums[j] == target) {
                    return new int[] { i, j };
                }
            }
        }
        return new int[] {};
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of elements: ");
        int n = scanner.nextInt();

        int[] nums = new int[n];

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        System.out.print("Enter target: ");
```

```

int target = scanner.nextInt();

int[] result = twoSum(nums, target);
if (result.length == 2) {
    System.out.println("Output: [" + result[0] + ", " + result[1] + "]");
} else {
    System.out.println("No solution found.");
}

scanner.close();
}
}

```

## # Q2 Remove duplicates from array

```

import java.util.Scanner;

public class RemoveDuplicates {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter sorted array elements:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
    }
}

```

```

int k = removeDuplicates(nums);

System.out.println("Unique elements count: " + k);
System.out.print("Unique elements: ");
for (int i = 0; i < k; i++) {
    System.out.print(nums[i] + " ");
}
}

public static int removeDuplicates(int[] nums) {
    if (nums.length == 0) return 0;

    int k = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] != nums[i-1]) {
            nums[k++] = nums[i];
        }
    }
    return k;
}
}

```

### # Q3 First repeating elements

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

```

public class FirstRepeatingElement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter array size: ");
        int n = sc.nextInt();

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

        int position = -1;

        outer:
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (arr[i] == arr[j]) {
                    position = i + 1; // 1-based indexing
                    break outer;
                }
            }
        }

        if (position == -1) {
            System.out.println("No repeating elements");
        } else {
            System.out.println("First repeating element at position: " + position);
        }
    }
}

```

```
}
```

## # Q4 Find pivot indexing

```
import java.util.Scanner;

public class PivotIndex {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter array size: ");
        int n = sc.nextInt();

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

        int totalSum = 0;
        for (int num : nums) {
            totalSum += num;
        }

        int leftSum = 0;
        int pivot = -1;

        for (int i = 0; i < nums.length; i++) {
            if (leftSum == totalSum - leftSum - nums[i]) {
```

```

        pivot = i;
        break;
    }
    leftSum += nums[i];
}

System.out.println("Pivot index: " + pivot);
}
}

```

## # Q5 Reverse the array

```

import java.util.Scanner;

public class ReverseAfterM {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter array size and M position: ");
        int n = sc.nextInt();
        int m = sc.nextInt();

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

        int start = m + 1;
        int end = n - 1;
    }
}

```

```

while (start < end) {
    int temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;
    start++;
    end--;
}

System.out.println("Modified array:");
for (int num : arr) {
    System.out.print(num + " ");
}
}
}

```

## # Q6 Removing minimum and maximum

```

import java.util.Scanner;

public class MinDeletionsToRemoveMinMax {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
    }
}

```

```

int result = minimumDeletions(nums);
System.out.println("Minimum number of deletions: " + result);

scanner.close();
}

public static int minimumDeletions(int[] nums) {
    if (nums.length == 1) {
        return 1;
    }

    int minIndex = 0;
    int maxIndex = 0;

    for (int i = 1; i < nums.length; i++) {
        if (nums[i] < nums[minIndex]) {
            minIndex = i;
        }
        if (nums[i] > nums[maxIndex]) {
            maxIndex = i;
        }
    }

    int left = Math.min(minIndex, maxIndex);
    int right = Math.max(minIndex, maxIndex);

    // Scenario 1: remove both from left
    int option1 = right + 1;

    // Scenario 2: remove both from right
    int option2 = nums.length - left;

```



```

// Scenario 3: remove one from left and one from right
int option3 = (left + 1) + (nums.length - right);

return Math.min(option1, Math.min(option2, option3));
}
}

```

## # Q7 Count element with maximum frequency

```

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class MaxFrequencyElements {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        int result = maxFrequencyElements(nums);
        System.out.println("Total frequencies of elements with maximum frequency: " + result);
    }
}

```

```

        scanner.close();
    }

    public static int maxFrequencyElements(int[] nums) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();

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

        int maxFrequency = 0;
        for (int freq : frequencyMap.values()) {
            if (freq > maxFrequency) {
                maxFrequency = freq;
            }
        }

        int count = 0;
        for (int freq : frequencyMap.values()) {
            if (freq == maxFrequency) {
                count += freq;
            }
        }

        return count;
    }
}

```

**# Q8 Rotate array to the right by k steps**

```

import java.util.Scanner;

public class RotateArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        System.out.println("Enter the number of rotations (k):");
        int k = scanner.nextInt();

        rotate(nums, k);

        System.out.println("Rotated array:");
        for (int num : nums) {
            System.out.print(num + " ");
        }

        scanner.close();
    }

    public static void rotate(int[] nums, int k) {
        k = k % nums.length;
        reverse(nums, 0, nums.length - 1);
    }
}

```

```

        reverse(nums, 0, k - 1);
        reverse(nums, k, nums.length - 1);
    }

    public static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
}

```

## # Q9 Bubble sort, selection sort, Insertion Sort

```

                                # bubble sort

import java.io.*;

class a {

    static void bubbleSort(int arr[], int n){
        int i, j, temp;
        boolean swapped;
        for (i = 0; i < n - 1; i++) {
            swapped = false;
            for (j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {

```

```

        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
        swapped = true;
    }
}

    if (swapped == false)
        break;
}
}

static void printArray(int arr[], int size){
    int i;
    for (i = 0; i < size; i++)
        System.out.print(arr[i] + " ");
    System.out.println();
}

public static void main(String args[]){
    int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
    int n = arr.length;
    bubbleSort(arr, n);
    System.out.println("Sorted array: ");
    printArray(arr, n);
}
}

# selection sort

import java.util.Arrays;

```

```
class a {

    static void selectionSort(int[] arr){
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {

            int min_idx = i;

            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[min_idx]) {

                    min_idx = j;
                }
            }

            int temp = arr[i];
            arr[i] = arr[min_idx];
            arr[min_idx] = temp;
        }
    }

    static void printArray(int[] arr){
        for (int val : arr) {
            System.out.print(val + " ");
        }
        System.out.println();
    }

    public static void main(String[] args){
        int[] arr = { 64, 25, 12, 22, 11 };

        System.out.print("Original array: ");
    }
}
```

```

        printArray(arr);

        selectionSort(arr);

        System.out.print("Sorted array: ");
        printArray(arr);
    }
}

```

# insertion sort

```

public class InsertionSort {
    void sort(int arr[])
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }

    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
    }
}

```

```

        System.out.println();
    }

    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6 };

        InsertionSort ob = new InsertionSort();
        ob.sort(arr);

        printArray(arr);
    }
}

```

### **# Q10 Check whether given string is a pallindrome or not**

```

import java.util.Scanner;

public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a string to check if it's a palindrome:");
        String input = scanner.nextLine();

        boolean isPalindrome = isPalindrome(input);
        System.out.println("Is the string a palindrome? " + isPalindrome);

        scanner.close();
    }
}

```



```

    }

    public static boolean isPalindrome(String s) {
        StringBuilder cleaned = new StringBuilder();

        for (char c : s.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                cleaned.append(Character.toLowerCase(c));
            }
        }

        String filtered = cleaned.toString();
        int left = 0;
        int right = filtered.length() - 1;

        while (left < right) {
            if (filtered.charAt(left) != filtered.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }

        return true;
    }
}

```

## # Q11 Count number of vowels and consonents

```

public static void solve(String str, int length) {

```

```

int vowels = 0, consonants = 0, whitespaces = 0;
str = str.toLowerCase();
for (int i = 0; i < length; i++) {
    char ch = str.charAt(i);
    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
        vowels++;
    else if (ch >= 'a' && ch <= 'z')
        consonants++;
    else if (ch == ' ')
        whitespaces++;
}

System.out.println("Vowels: " + vowels);
System.out.println("Consonants: " + consonants);
System.out.println("White spaces: " + whitespaces);
}

public static void main(String args[]) {
    String str = "Take u forward is Awesome";
    int length = str.length();
    solve(str, length);
}

```

## # Q12 Remove characters except alphabets

```

import java.util.Scanner;

public class RemoveNonAlphabets {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a string:");
    }
}

```

```

String input = scanner.nextLine();

String result = removeNonAlphabets(input);
System.out.println("String with only alphabets: " + result);

scanner.close();
}

public static String removeNonAlphabets(String str) {
    StringBuilder result = new StringBuilder();

    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        if (Character.isLetter(c)) {
            result.append(c);
        }
    }

    return result.toString();
}
}

```

### **# Q13 Finding frequency of a character in a string**

```

import java.util.Scanner;

public class SimpleCharFrequency {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
    }
}

```

```

System.out.print("Enter a string: ");
String s = sc.nextLine();

int[] freq = new int[256];

for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    freq[c]++;
}

System.out.print("Character frequencies: ");
for (int i = 0; i < 256; i++) {
    if (freq[i] > 0) {
        System.out.print((char)i + " " + freq[i] + " ");
    }
}
}
}

```

## # Q14 Finding max occurrence character

```

import java.util.Scanner;

public class MaxOccurringChar {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
    }
}

```

```

char maxChar = findMaxOccurringChar(str);

System.out.println("Character with maximum occurrence: " + maxChar);

scanner.close();
}

public static char findMaxOccurringChar(String str) {
    int[] count = new int[256];
    int max = -1;
    char result = ' ';

    for (int i = 0; i < str.length(); i++) {
        count[str.charAt(i)]++;
        if (count[str.charAt(i)] > max) {
            max = count[str.charAt(i)];
            result = str.charAt(i);
        }
    }

    return result;
}
}

```

## **# Q15 Factorial of a number**

```

import java.util.Scanner;

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

```

```

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a non-negative integer: ");
int n = scanner.nextInt();

long factorial = iterativeFactorial(n);
System.out.println("Factorial (iterative): " + factorial);

scanner.close();
}

public static long iterativeFactorial(int n) {
    long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
}

```

### **# Q16 Number raised to the power of its own reverse**

```

import java.util.Scanner;

public class PowerOfReverse {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int n = scanner.nextInt();
    }
}

```

```

        int result = powerOfReverse(n);
        System.out.println(n + " raised to the power of its reverse: " + result);

        scanner.close();
    }

    public static int powerOfReverse(int n) {
        int reversed = reverseNumber(n);
        return (int) Math.pow(n, reversed);
    }

    public static int reverseNumber(int num) {
        int reversed = 0;
        while (num != 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }
        return reversed;
    }
}

```

### **# Q17 print 1 to n without using loops**

```

import java.util.Scanner;

public class PrintNumbersWithoutLoop {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
    }
}

```

```

        System.out.print("Enter a positive integer (n): ");
        int n = scanner.nextInt();

        printTillN(n);

        scanner.close();
    }

    public static void printTillN(int n) {
        if (n > 0) {
            printTillN(n - 1);
            System.out.print(n + " ");
        }
    }
}

```

### **# Q18 Count digits**

```

import java.util.Scanner;

public class CountDividingDigits {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a positive integer (n): ");
        int n = scanner.nextInt();

        int count = countDividingDigits(n);
        System.out.println("Number of digits that divide " + n + " evenly: " + count);

        scanner.close();
    }
}

```



```

    }

    public static int countDividingDigits(int n) {
        int originalNumber = n;
        int count = 0;

        while (n > 0) {
            int digit = n % 10;
            if (digit != 0 && originalNumber % digit == 0) {
                count++;
            }
            n /= 10;
        }

        return count;
    }
}

```

### # Q19 sum of array using recursion

```

import java.util.Scanner;

public class ArraySum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter the elements of the array:");
    }
}

```

```

    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }

    int sum = calculateSum(arr);
    System.out.println("Sum of the array elements: " + sum);

    scanner.close();
}

public static int calculateSum(int[] arr) {
    int sum = 0;
    for (int num : arr) {
        sum += num;
    }
    return sum;
}
}

```

## # Q20 fibonacci number

```

import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a non-negative integer (n): ");
        int n = scanner.nextInt();

        long fibNumber = iterativeFibonacci(n);
    }
}

```

```

        System.out.println("The " + n + "th Fibonacci number is: " + fibNumber);

        scanner.close();
    }

    public static long iterativeFibonacci(int n) {
        if (n == 0) return 0;
        if (n == 1) return 1;

        long a = 0, b = 1, c = 0;
        for (int i = 2; i <= n; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
}

```

## **# Q21 tower of hanoi with recursion tree presentation**

```

import java.util.Scanner;

public class TowerOfHanoi {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of disks (n): ");
        int n = scanner.nextInt();
    }
}

```

```

        int totalMoves = solveHanoi(n, 'A', 'C', 'B');
        System.out.println("Total moves required: " + totalMoves);

        scanner.close();
    }

    public static int solveHanoi(int n, char fromRod, char toRod, char auxRod) {
        if (n == 1) {
            System.out.println("Move disk 1 from rod " + fromRod + " to rod " + toRod);
            return 1;
        }

        int moves = 0;
        moves += solveHanoi(n - 1, fromRod, auxRod, toRod);
        System.out.println("Move disk " + n + " from rod " + fromRod + " to rod " + toRod);
        moves++;
        moves += solveHanoi(n - 1, auxRod, toRod, fromRod);

        return moves;
    }
}

```

## # Q22 Spiral traversal

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

```

import java.util.List;
import java.util.Scanner;

public class SpiralMatrix {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of rows (m): ");
        int m = scanner.nextInt();
        System.out.print("Enter number of columns (n): ");
        int n = scanner.nextInt();

        int[][] matrix = new int[m][n];
        System.out.println("Enter matrix elements row-wise:");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }

        List<Integer> spiralOrder = spiralOrder(matrix);
        System.out.println("Spiral order: " + spiralOrder);

        scanner.close();
    }

    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix == null || matrix.length == 0) return result;

        int top = 0, bottom = matrix.length - 1;
        int left = 0, right = matrix[0].length - 1;

```

```

while (top <= bottom && left <= right) {
    for (int i = left; i <= right; i++) {
        result.add(matrix[top][i]);
    }
    top++;

    for (int i = top; i <= bottom; i++) {
        result.add(matrix[i][right]);
    }
    right--;

    if (top <= bottom) {
        for (int i = right; i >= left; i--) {
            result.add(matrix[bottom][i]);
        }
        bottom--;
    }

    if (left <= right) {
        for (int i = bottom; i >= top; i--) {
            result.add(matrix[i][left]);
        }
        left++;
    }
}

return result;
}
}

```

## # Q23 searching elements in a matrix

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length, n = matrix[0].length;
        int left = 0, right = m * n - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            int midValue = matrix[mid / n][mid % n];

            if (midValue == target) return true;
            else if (midValue < target) left = mid + 1;
            else right = mid - 1;
        }

        return false;
    }
}
```

## # Q24 Printing elements in sorted order

```
class Solution {
    public int[] sortArray(int[] nums) {
        mergeSort(nums, 0, nums.length - 1);
    }
}
```

```

        return nums;
    }

    private void mergeSort(int[] nums, int left, int right) {
        if (left >= right) return;
        int mid = left + (right - left) / 2;
        mergeSort(nums, left, mid);
        mergeSort(nums, mid + 1, right);
        merge(nums, left, mid, right);
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;

        while (i <= mid && j <= right) {
            if (nums[i] < nums[j]) {
                temp[k++] = nums[i++];
            } else {
                temp[k++] = nums[j++];
            }
        }

        while (i <= mid) temp[k++] = nums[i++];
        while (j <= right) temp[k++] = nums[j++];

        for (int l = 0; l < temp.length; l++) {
            nums[left + l] = temp[l];
        }
    }
}

```



## # Q25 valid parentheses

```
import java.util.Stack;

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

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);

            if (ch == '(') {
                stack.push('(');
            } else if (ch == '{') {
                stack.push('{');
            } else if (ch == '[') {
                stack.push('[');
            } else {
                if (stack.isEmpty() || stack.pop() != ch) {
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        String s = "()";
```

```

        System.out.println(solution.isValid(s)); // Output: true
    }
}

```

## # Q26 Evaluate postfix expression

```

import java.util.*;

public class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();

        for (String token : tokens) {
            if (isOperator(token)) {
                int b = stack.pop();
                int a = stack.pop();
                stack.push(applyOperator(a, b, token));
            } else {
                stack.push(Integer.parseInt(token));
            }
        }

        return stack.pop();
    }

    private boolean isOperator(String token) {
        return token.equals("+") || token.equals("-") ||
            token.equals("*") || token.equals("/");
    }
}

```

```

private int applyOperator(int a, int b, String op) {
    switch (op) {
        case "+": return a + b;
        case "-": return a - b;
        case "*": return a * b;
        case "/": return a / b; // integer division rounds toward zero
        default: throw new IllegalArgumentException("Invalid operator: " + op);
    }
}

public static void main(String[] args) {
    Solution sol = new Solution();
    String[] arr = {"2", "3", "1", "*", "+", "9", "-"};
    System.out.println(sol.evalRPN(arr)); // Output: -4
}
}

```

## # Q27 min stack

```

import java.util.Stack;

class MinStack {
    private Stack<Integer> mainStack;
    private Stack<Integer> minStack;

    public MinStack() {

```

```

    mainStack = new Stack<>();
    minStack = new Stack<>();
}

public void push(int val) {
    mainStack.push(val);
    if (minStack.isEmpty() || val <= minStack.peek()) {
        minStack.push(val);
    } else {
        minStack.push(minStack.peek());
    }
}

public void pop() {
    mainStack.pop();
    minStack.pop();
}

public int top() {
    return mainStack.peek();
}

public int getMin() {
    return minStack.peek();
}
}

```

## # Q28 Stack Implementation using Array

```

public class Stack {

```

```

private char[] stackArray;
private int top;
private int maxSize;

public Stack(int size) {
    maxSize = size;
    stackArray = new char[maxSize];
    top = -1;
}

public void push(char ch) {
    if (top < maxSize - 1) {
        stackArray[++top] = ch;
    }
}

public char pop() {
    if (top >= 0) {
        return stackArray[top--];
    }
    return '\0';
}

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

public static String reverseString(String input) {
    Stack stack = new Stack(input.length());

    for (int i = 0; i < input.length(); i++) {
        stack.push(input.charAt(i));
    }
}

```

```

    }

    StringBuilder reversed = new StringBuilder();

    while (!stack.isEmpty()) {
        reversed.append(stack.pop());
    }

    return reversed.toString();
}

public static void main(String[] args) {
    String input = "Hello, World!";
    String reversed = reverseString(input);
    System.out.println(reversed);
}
}

```

## # Q29 Next Greater Element

```

import java.util.*;

public class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();

        for (int i = nums2.length - 1; i >= 0; i--) {
            int current = nums2[i];
            while (!stack.isEmpty() && stack.peek() <= current) {
                stack.pop();
            }
        }
    }
}

```

```

    }
    int nextGreater = stack.isEmpty() ? -1 : stack.peek();
    nextGreaterMap.put(current, nextGreater);
    stack.push(current);
}

int[] result = new int[nums1.length];
for (int i = 0; i < nums1.length; i++) {
    result[i] = nextGreaterMap.get(nums1[i]);
}

return result;
}

public static void main(String[] args) {
    Solution sol = new Solution();
    int[] nums1 = {4, 1, 2};
    int[] nums2 = {1, 3, 4, 2};
    System.out.println(Arrays.toString(sol.nextGreaterElement(nums1, nums2)));
}
}

```

### # Q30 smaller element on left

```

import java.util.*;

public class Solution {
    public static int[] findGreatestSmallerLeft(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        TreeSet<Integer> set = new TreeSet<>();
    }
}

```

```

        for (int i = 0; i < n; i++) {
            Integer smaller = set.lower(arr[i]);
            result[i] = (smaller == null) ? -1 : smaller;
            set.add(arr[i]);
        }

        return result;
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 5, 1};
        int[] result = findGreatestSmallerLeft(arr);

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

```

### # Q31 Two sum problem

```

class Solution {
    public int[] twoSum(int[] nums, int target) {
        for (int i = 0; i < nums.length; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[i] + nums[j] == target) {
                    return new int[] { i, j };
                }
            }
        }
    }
}

```



```
    }  
    return new int[] {};  
}  
}
```

## # Q12 Best Time to Buy and Sell Stock

```
class Solution {  
    public int maxProfit(int[] prices) {  
        int maxProfit = 0;  
        int minPrice = Integer.MAX_VALUE;  
  
        for (int i = 0; i < prices.length; i++) {  
            minPrice = Math.min(prices[i], minPrice);  
  
            int profit = prices[i] - minPrice;  
  
            maxProfit = Math.max(maxProfit, profit);  
        }  
  
        return maxProfit;  
    }  
}
```

## # Q33 Sort Colors

```

class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;

        while (mid <= high) {
            if (nums[mid] == 0) {

                int temp = nums[low];
                nums[low] = nums[mid];
                nums[mid] = temp;
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                int temp = nums[mid];
                nums[mid] = nums[high];
                nums[high] = temp;
                high--;
            }
        }
    }
}

```

### # Q34 Container With Most Water

```

class Solution {

```

```

public int maxArea(int[] height) {
    int left = 0;
    int right = height.length - 1;
    int maxArea = 0;

    while (left < right) {
        int currentArea = Math.min(height[left], height[right]) * (right - left);
        maxArea = Math.max(maxArea, currentArea);

        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return maxArea;
}

```

### # Q35 Merge Sorted Array

```

class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1;
        int j = n - 1;
        int k = m + n - 1;
    }
}

```

```

while (i >= 0 && j >= 0) {
    if (nums1[i] > nums2[j]) {
        nums1[k--] = nums1[i--];
    } else {
        nums1[k--] = nums2[j--];
    }
}

while (j >= 0) {
    nums1[k--] = nums2[j--];
}
}
}

```

### # Q36 Trapping Rain Water

```

class Solution {
public int trap(int[] height) {
    int left = 0, right = height.length - 1;
    int leftMax = 0, rightMax = 0;
    int waterTrapped = 0;

    while (left < right) {
        if (height[left] < height[right]) {
            if (height[left] >= leftMax) {
                leftMax = height[left];
            }
        }
    }
}

```

```

        } else {
            waterTrapped += leftMax - height[left];
        }
        left++;
    } else {
        if (height[right] >= rightMax) {
            rightMax = height[right];
        } else {
            waterTrapped += rightMax - height[right];
        }
        right--;
    }
}

return waterTrapped;
}
}

```

### # Q37 Implement Lower Bound

```

public class LowerBound {

    public static int lowerBound(int[] arr, int x) {
        int left = 0;
        int right = arr.length;

        while (left < right) {

```

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

```
if (arr[mid] < x) {
```

```
    left = mid + 1;
```

```
} else {
```

```
    right = mid;
```

```
}
```

```
}
```

```
return left;
```

```
}
```

```
public static void main(String[] args) {
```

```
    int[] arr = {1, 2, 4, 4, 5, 6, 8};
```

```
    int x = 4;
```

```
    int index = lowerBound(arr, x);
```

```
    if (index < arr.length) {
```

```
        System.out.println("Lower bound of " + x + " is at index: " + index + ", value:  
" + arr[index]);
```

```
    } else {
```

```
        System.out.println("No element >= " + x + " found in the array.");
```

```
    }
```

```
}
```

```
}
```

### # Q38 Implement Upper Bound

```
public class UpperBound {

    public static int upperBound(int[] arr, int x) {
        int left = 0;
        int right = arr.length;

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

            if (arr[mid] <= x) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        return left; // Index of upper bound
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 4, 5, 6, 8};
        int x = 4;

        int index = upperBound(arr, x);
```

```

        if (index < arr.length) {
            System.out.println("Upper bound of " + x + " is at index: " + index + ", value: " + arr[index]);
        } else {
            System.out.println("No element > " + x + " found in the array.");
        }
    }
}

```

### # Q39 Koko Eating Bananas

```

class Solution {
    public int minEatingSpeed(int[] piles, int h) {
        int left = 1;
        int right = 0;

        for (int pile : piles) {
            right = Math.max(right, pile);
        }

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (canFinish(piles, mid, h)) {
                right = mid; // Try a smaller eating speed
            } else {
                left = mid + 1; // Increase the speed
            }
        }
    }
}

```



```

    }

    return left;
}

private boolean canFinish(int[] piles, int k, int h) {
    int hours = 0;
    for (int pile : piles) {
        hours += (pile + k - 1) / k;
    }
    return hours <= h;
}
}

```

## # Q40 First Bad Version

```

public class Solution {

    // This is a mock of the isBadVersion function, which is provided by LeetCode in
    // the actual problem.
    boolean isBadVersion(int version) {
        return version >= 4;
    }

    public int firstBadVersion(int n) {
        int left = 1;
        int right = n;
    }
}

```

```

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

    if (isBadVersion(mid)) {
        right = mid;
    } else {
        left = mid + 1;
    }
}

return left;
}

public static void main(String[] args) {
    Solution solution = new Solution();
    int result = solution.firstBadVersion(5);
    System.out.println("First Bad Version: " + result);
}
}

```

## # Q41 Search in Rotated Sorted Array

```

class Solution {
    public int search(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
    }
}

```

```

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

    if (nums[mid] == target) {
        return mid;
    }

    if (nums[left] <= nums[mid]) {
        if (target >= nums[left] && target < nums[mid]) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    } else {

        if (target > nums[mid] && target <= nums[right]) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
}

return -1;
}
}

```

## # Q42 Search in Rotated Sorted Array II

```
class Solution {  
    public boolean search(int[] nums, int target) {  
        int left = 0;  
        int right = nums.length - 1;  
  
        while (left <= right) {  
            int mid = left + (right - left) / 2;  
  
            if (nums[mid] == target) {  
                return true;  
            }  
  
            if (nums[left] == nums[mid] && nums[mid] == nums[right]) {  
                left++;  
                right--;  
            }  
            else if (nums[left] <= nums[mid]) {  
                if (target >= nums[left] && target < nums[mid]) {  
                    right = mid - 1;  
                } else {  
                    left = mid + 1;  
                }  
            }  
        }  
    }  
}
```

```

    }

    else {

        if (target > nums[mid] && target <= nums[right]) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
}

return false;
}
}

```

### # Q43 Create Binary Tree from descriptions

```

class Solution {
    public TreeNode createBinaryTree(int[][] descriptions) {
        Map<Integer, TreeNode> map = new HashMap<>();
        Set<Integer> children = new HashSet<>();

        for (int[] desc : descriptions) {
            int parentVal = desc[0];
            int childVal = desc[1];
            boolean isLeft = desc[2] == 1;

```

```

map.putIfAbsent(parentVal, new TreeNode(parentVal));
map.putIfAbsent(childVal, new TreeNode(childVal));

TreeNode parent = map.get(parentVal);
TreeNode child = map.get(childVal);

if (isLeft) {
    parent.left = child;
} else {
    parent.right = child;
}
children.add(childVal);
}

for (int[] desc : descriptions) {
    int parentVal = desc[0];
    if (!children.contains(parentVal)) {
        return map.get(parentVal);
    }
}

return null;
}
}

```

## **# Q44 Binary Tree Preorder Traversal**

```
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);

        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
            result.add(node.val);

            if (node.right != null) {
                stack.push(node.right);
            }

            if (node.left != null) {
                stack.push(node.left);
            }
        }

        return result;
    }
}
```

## # Q45 Binary Inorder Tree Traversal

```
class Solution {  
    public List<Integer> inorderTraversal(TreeNode root) {  
        List<Integer> result = new ArrayList<>();  
        Stack<TreeNode> stack = new Stack<>();  
        TreeNode current = root;  
  
        while (current != null || !stack.isEmpty()) {  
            while (current != null) {  
                stack.push(current);  
                current = current.left;  
            }  
  
            current = stack.pop();  
            result.add(current.val);  
  
            current = current.right;  
        }  
  
        return result;  
    }  
}
```

## # Q46 Binary Tree Postorder Traversal



```

class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Stack<TreeNode> stack = new Stack<>();
        TreeNode lastVisited = null;

        while (!stack.isEmpty() || root != null) {
            // Reach the leftmost node
            while (root != null) {
                stack.push(root);
                root = root.left;
            }

            TreeNode peekNode = stack.peek();

            // If right child is null or already visited, process the root
            if (peekNode.right == null || peekNode.right == lastVisited) {
                result.add(peekNode.val);
                lastVisited = stack.pop();
            } else {
                root = peekNode.right;
            }
        }
    }
}

```

```

    }

    return result;
}
}

```

## # Q47 Binary Tree Level Order Traversal

```

import java.util.*;

class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                currentLevel.add(node.val);
            }
        }
    }
}

```

```

        // Enqueue left and right children
        if (node.left != null) {
            queue.offer(node.left);
        }
        if (node.right != null) {
            queue.offer(node.right);
        }
    }

    result.add(currentLevel);
}

return result;
}
}

```

## **# Q48 Maximum Depth Of Binary Tree**

```

import java.util.*;

class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
    }
}

```

```

Queue<TreeNode> queue = new LinkedList<>();
queue.offer(root);
int depth = 0;

while (!queue.isEmpty()) {
    int levelSize = queue.size();
    for (int i = 0; i < levelSize; i++) {
        TreeNode node = queue.poll();

        if (node.left != null) {
            queue.offer(node.left);
        }
        if (node.right != null) {
            queue.offer(node.right);
        }
    }
    depth++;
}

return depth;
}
}

```

## # Q49 Same Tree

```

class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {

```

```

    if (p == null && q == null) {
        return true;
    }

    if (p == null || q == null) {
        return false;
    }

    if (p.val != q.val) {
        return false;
    }

    return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
}
}

```

## # Q50 Symmetric Tree

```

class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) return true;
        return isMirror(root.left, root.right);
    }

    private boolean isMirror(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) return true;
        if (t1 == null || t2 == null) return false;

```

```

        if (t1.val != t2.val) return false;

        return isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
    }
}

```

## # Q51 Diameter Of Binary Tree

```

class Solution {
    private int diameter = 0;

    public int diameterOfBinaryTree(TreeNode root) {
        depth(root);
        return diameter;
    }

    private int depth(TreeNode node) {
        if (node == null) return 0;

        int left = depth(node.left);
        int right = depth(node.right);

        diameter = Math.max(diameter, left + right);

        return 1 + Math.max(left, right);
    }
}

```

## # Q52 Path Sum

```
class Solution {  
    public boolean hasPathSum(TreeNode root, int targetSum) {  
        if (root == null) return false;  
  
        if (root.left == null && root.right == null) {  
            return targetSum == root.val;  
        }  
  
        int remaining = targetSum - root.val;  
  
        return hasPathSum(root.left, remaining) || hasPathSum(root.right, remaining);  
    }  
}
```

## # Q53 Binary Tree Right Side View

```
import java.util.*;  
  
class Solution {  
    public List<Integer> rightSideView(TreeNode root) {  
        List<Integer> result = new ArrayList<>();  
        if (root == null) return result;
```

```

Queue<TreeNode> queue = new LinkedList<>();
queue.offer(root);

while (!queue.isEmpty()) {
    int levelSize = queue.size();

    for (int i = 0; i < levelSize; i++) {
        TreeNode curr = queue.poll();

        if (i == levelSize - 1) {
            result.add(curr.val);
        }

        if (curr.left != null) queue.offer(curr.left);
        if (curr.right != null) queue.offer(curr.right);
    }
}

return result;
}
}

```

## # Q54 Validate Binary Search Tree

```

class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValid(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
}

```



```

    }

    private boolean isValid(TreeNode node, long min, long max) {
        if (node == null) return true;

        if (node.val <= min || node.val >= max) return false;

        return isValid(node.left, min, node.val) &&
            isValid(node.right, node.val, max);
    }
}

```

## # Q55 Convert Sorted Array To Binary Search Tree

```

class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return buildBST(nums, 0, nums.length - 1);
    }

    private TreeNode buildBST(int[] nums, int left, int right) {
        if (left > right) return null;

        int mid = left + (right - left) / 2;
        TreeNode node = new TreeNode(nums[mid]);

        node.left = buildBST(nums, left, mid - 1);
        node.right = buildBST(nums, mid + 1, right);
    }
}

```

```
        return node;
    }
}
```

## #Q56 Delete Node In BST

```
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) return null;

        if (key < root.val) {
            root.left = deleteNode(root.left, key);
        } else if (key > root.val) {
            root.right = deleteNode(root.right, key);
        } else {
            if (root.left == null) return root.right;
            if (root.right == null) return root.left;

            TreeNode successor = findMin(root.right);
            root.val = successor.val;
            root.right = deleteNode(root.right, successor.val);
        }

        return root;
    }
}
```

```

private TreeNode findMin(TreeNode node) {
    while (node.left != null) {
        node = node.left;
    }
    return node;
}
}

```

### # Q57 Lowest Common Ancestor Of Binary Tree

```

class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
    {
        if (root == null || root == p || root == q) {
            return root;
        }

        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);

        if (left != null && right != null) {
            return root;
        }

        return left != null ? left : right;
    }
}

```

```
}
```

## # Q58 Missing Number

```
class Solution {  
    public int missingNumber(int[] nums) {  
        int n = nums.length;  
        int expectedSum = (n * (n + 1)) / 2;  
        int actualSum = 0;  
  
        for (int num : nums) {  
            actualSum += num;  
        }  
  
        return expectedSum - actualSum;  
    }  
}
```

## # Q59 Intersection Of Two Arrays

```
import java.util.HashSet;  
import java.util.Set;  
  
class Solution {  
    public int[] intersection(int[] nums1, int[] nums2) {  
        Set<Integer> set1 = new HashSet<>();
```

```

Set<Integer> result = new HashSet<>();

for (int num : nums1) {
    set1.add(num);
}

for (int num : nums2) {
    if (set1.contains(num)) {
        result.add(num);
    }
}

int[] intersection = new int[result.size()];
int i = 0;
for (int num : result) {
    intersection[i++] = num;
}

return intersection;
}
}

```

## # Q60 Set Matrix Zero

```

class Solution {
    public void setZeroes(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
    }
}

```

```
boolean firstRowZero = false;
```

```
boolean firstColZero = false;
```

```
for (int j = 0; j < n; j++) {  
    if (matrix[0][j] == 0) {  
        firstRowZero = true;  
        break;  
    }  
}
```

```
for (int i = 0; i < m; i++) {  
    if (matrix[i][0] == 0) {  
        firstColZero = true;  
        break;  
    }  
}
```

```
for (int i = 1; i < m; i++) {  
    for (int j = 1; j < n; j++) {  
        if (matrix[i][j] == 0) {  
            matrix[i][0] = 0;  
            matrix[0][j] = 0;  
        }  
    }  
}
```

```
for (int i = 1; i < m; i++) {
```

```

        for (int j = 1; j < n; j++) {
            if (matrix[i][0] == 0 || matrix[0][j] == 0) {
                matrix[i][j] = 0;
            }
        }
    }

    // Step 5: Handle the first row
    if (firstRowZero) {
        for (int j = 0; j < n; j++) {
            matrix[0][j] = 0;
        }
    }

    if (firstColZero) {
        for (int i = 0; i < m; i++) {
            matrix[i][0] = 0;
        }
    }
}

```

## # Q61 asteroid collision

```

import java.util.*;

public class Solution {
    public int[] asteroidCollision(int[] asteroids) {

```

```
Stack<Integer> stack = new Stack<>();
```

```
for (int asteroid : asteroids) {
```

```
    boolean exploded = false;
```

```
    while (!stack.isEmpty() && asteroid < 0 && stack.peek() > 0) {
```

```
        if (Math.abs(asteroid) > Math.abs(stack.peek())) {
```

```
            stack.pop();
```

```
            continue;
```

```
        } else if (Math.abs(asteroid) == Math.abs(stack.peek())) {
```

```
            stack.pop();
```

```
        }
```

```
        exploded = true;
```

```
        break;
```

```
    }
```

```
    if (!exploded) {
```

```
        stack.push(asteroid);
```

```
    }
```

```
}
```

```
int[] result = new int[stack.size()];
```

```
for (int i = stack.size() - 1; i >= 0; i--) {
```

```
    result[i] = stack.pop();
```

```
}
```

```
return result;
```

```
}
```

```
public static void main(String[] args) {
```

```
    Solution sol = new Solution();
```

```
    int[] asteroids = {5, 10, -5};
```



```

        System.out.println(Arrays.toString(sol.asteroidCollision(asteroids)));
    }
}

```

## # Q62 stock span problem

```

import java.util.*;

class StockSpanner {

    private Stack<PriceSpan> stack;

    private static class PriceSpan {
        int price;
        int span;

        PriceSpan(int price, int span) {
            this.price = price;
            this.span = span;
        }
    }

    public StockSpanner() {
        stack = new Stack<>();
    }

    public int next(int price) {
        int span = 1;

        while (!stack.isEmpty() && stack.peek().price <= price) {
            span += stack.pop().span;
        }
    }
}

```

```
}
```

```
stack.push(new PriceSpan(price, span));
```

```
return span;
```

```
}
```

```
public static void main(String[] args) {
```

```
    StockSpanner spanner = new StockSpanner();
```

```
    System.out.println(spanner.next(100));
```

```
    System.out.println(spanner.next(80));
```

```
    System.out.println(spanner.next(60));
```

```
    System.out.println(spanner.next(70));
```

```
    System.out.println(spanner.next(60));
```

```
    System.out.println(spanner.next(75));
```

```
    System.out.println(spanner.next(85));
```

```
}
```

```
}
```

