**NOV 10, 24 PRACTICE SET-1**

**MEENA RHOSHINI. C**

**22CS034**

1. **Maximum Subarray Sum – Kadane’s Algorithm:**

public class KadaneAlgorithm {

public static int maxSubarraySum(int arr[]) {

int currentMax = arr[0];

int globalMax = arr[0];

for (int i = 1; i < arr.length; i++) {

currentMax = Math.max(arr[i], currentMax + arr[i]);

globalMax = Math.max(currentMax, globalMax);

}

return globalMax;

}

public static void main(String[] args) {

int[] arr1 = {2, 3, -8, 7, -1, 2, 3};

int[] arr2 = {-2, -4};

int[] arr3 = {5, 4, 1, 7, 8};

int[] arr4 = {1, -3, 2, 1, -1}; // Hidden test case 1

int[] arr5 = {-1, -2, -3, -4}; // Hidden test case 2

int[] arr6 = {0, 0, 0, 0, 0}; // Hidden test case 3

int[] arr7 = {3, -1, 4, -1, 5, -9, 2, 6}; // Hidden test case 4

System.out.println(maxSubarraySum(arr1));

System.out.println(maxSubarraySum(arr2));

System.out.println(maxSubarraySum(arr3));

System.out.println(maxSubarraySum(arr4));

System.out.println(maxSubarraySum(arr5));

System.out.println(maxSubarraySum(arr6));

System.out.println(maxSubarraySum(arr7));

}

}

OUTPUT:



Time Complexity: O(n)

Space complexity: O(1)

1. **Maximum Product Subarray**

public class maxProductSubarray {

public static int maxSubarrProduct(int[] arr) {

int max = arr[0];

int min = arr[0];

int res = arr[0];

int temp;

for (int i = 1; i < arr.length; i++) {

if (arr[i] < 0) {

temp = max;

max = min;

min = temp;

}

max = Math.max(arr[i], max \* arr[i]);

min = Math.min(arr[i], min \* arr[i]);

res = Math.max(res, max);

}

return res;

}

public static void main(String[] args) {

int[] arr1 = {-2, 6, -3, -10, 0, 2};

int[] arr2 = {-1, -3, -10, 0, 60};

int[] arr3 = {2, 3, -2, 4}; // Hidden test case 1

int[] arr4 = {-2, 0, -1}; // Hidden test case 2

int[] arr5 = {1, 2, 3, 4, 5}; // Hidden test case 3

int[] arr6 = {-1, -2, -3, -4, -5}; // Hidden test case 4

System.out.println(maxSubarrProduct(arr1));

System.out.println(maxSubarrProduct(arr2));

System.out.println(maxSubarrProduct(arr3));

System.out.println(maxSubarrProduct(arr4));

System.out.println(maxSubarrProduct(arr5));

System.out.println(maxSubarrProduct(arr6));

}

}

OUTPUT:



Time complexity: O(n)

Space complexity: O(1)

1. **Search in a sorted and rotated Array**

public class searchInRotatedArr {

public static int searchSort(int[] arr, int key) {

int left = 0, right = arr.length - 1;

while (left <= right) {

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

if (arr[mid] == key) {

return mid;

}

if (arr[left] <= arr[mid]) {

if (key >= arr[left] && key < arr[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

}

else {

if (key > arr[mid] && key <= arr[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

int[] arr1 = {4, 5, 6, 7, 0, 1, 2};

int[] arr2 = {4, 5, 6, 7, 0, 1, 2};

int[] arr3 = {50, 10, 20, 30, 40};

int[] arr4 = {3, 4, 5, 1, 2}; // Hidden test case 1

int[] arr5 = {10, 20, 30, 40, 50}; // Hidden test case 2

int[] arr6 = {1}; // Hidden test case 3

int[] arr7 = {2, 3, 4, 5, 1}; // Hidden test case 4

System.out.println(searchSort(arr1, 0));

System.out.println(searchSort(arr2, 3));

System.out.println(searchSort(arr3, 10));

System.out.println(searchSort(arr4, 1));

System.out.println(searchSort(arr5, 40));

System.out.println(searchSort(arr6, 1));

System.out.println(searchSort(arr7, 1));

}

}

OUTPUT:



Time complexity: O(log n)

Space complexity: O(1)

1. **Container with Most Water**

import java.util.\*;

public class ContainerWithMostWater {

public static int calculateMaxArea(int[] heightArray) {

int leftPointer = 0, rightPointer = heightArray.length - 1;

int maxWaterArea = 0;

while (leftPointer < rightPointer) {

int height = Math.min(heightArray[leftPointer], heightArray[rightPointer]);

int width = rightPointer - leftPointer;

maxWaterArea = Math.max(maxWaterArea, height \* width);

if (heightArray[leftPointer] < heightArray[rightPointer]) {

leftPointer++;

} else {

rightPointer--;

}

}

return maxWaterArea;

}

public static void main(String[] args) {

int[] testCase1 = {1, 5, 4, 3};

int[] testCase2 = {3, 1, 2, 4, 5};

int[] testCase3 = {1, 1}; // Hidden test case 1

int[] testCase4 = {6, 9, 3, 4, 5, 8}; // Hidden test case 2

int[] testCase5 = {8, 10, 14, 0, 4, 6}; // Hidden test case 3

int[] testCase6 = {1, 3, 2, 5, 25, 24, 5}; // Hidden test case 4

System.out.println(calculateMaxArea(testCase1));

System.out.println(calculateMaxArea(testCase2));

System.out.println(calculateMaxArea(testCase3));

System.out.println(calculateMaxArea(testCase4));

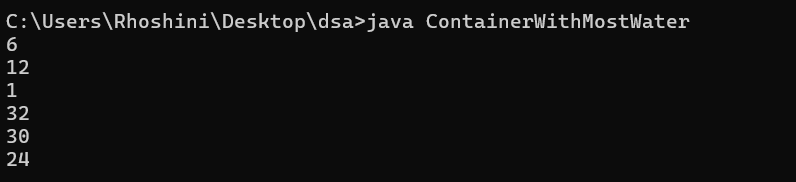
System.out.println(calculateMaxArea(testCase5));

System.out.println(calculateMaxArea(testCase6));

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(1)

1. **Find the Factorial of a large number**

import java.math.BigInteger;

public class Factorial {

public static BigInteger factorialOfN(int n) {

BigInteger res = BigInteger.ONE;

for (int i = 2; i <= n; i++) {

res = res.multiply(BigInteger.valueOf(i));

}

return res;

}

public static void main(String[] args) {

System.out.println(factorialOfN(100));

System.out.println(factorialOfN(50));

int[] hiddenTestCase1 = {0}; // Hidden test case 1

int[] hiddenTestCase2 = {1}; // Hidden test case 2

int[] hiddenTestCase3 = {5}; // Hidden test case 3

int[] hiddenTestCase4 = {10}; // Hidden test case 4

System.out.println(factorialOfN(hiddenTestCase1[0]));

System.out.println(factorialOfN(hiddenTestCase2[0]));

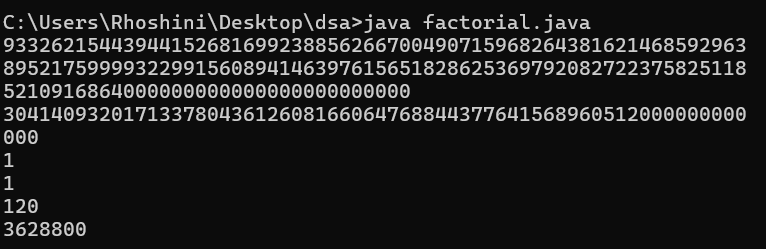
System.out.println(factorialOfN(hiddenTestCase3[0]));

System.out.println(factorialOfN(hiddenTestCase4[0]));

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(1)

1. **Trapping Rainwater Problem**

public class TrappingRainwater {

public static int calculateTrappedWater(int[] elevationMap) {

int n = elevationMap.length;

if (n <= 2) return 0;

int[] leftMax = new int[n];

int[] rightMax = new int[n];

int totalWater = 0;

leftMax[0] = elevationMap[0];

for (int i = 1; i < n; i++) {

leftMax[i] = Math.max(leftMax[i - 1], elevationMap[i]);

}

rightMax[n - 1] = elevationMap[n - 1];

for (int i = n - 2; i >= 0; i--) {

rightMax[i] = Math.max(rightMax[i + 1], elevationMap[i]);

}

for (int i = 0; i < n; i++) {

totalWater += Math.min(leftMax[i], rightMax[i]) - elevationMap[i];

}

return totalWater;

}

public static void main(String[] args) {

int[] testCase1 = {3, 0, 1, 0, 4, 0, 2};

int[] testCase2 = {3, 0, 2, 0, 4};

int[] testCase3 = {1, 2, 3, 4};

int[] testCase4 = {10, 9, 0, 5};

int[] testCase5 = {4, 2, 0, 3, 2, 5};

int[] testCase6 = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};

System.out.println(calculateTrappedWater(testCase1));

System.out.println(calculateTrappedWater(testCase2));

System.out.println(calculateTrappedWater(testCase3));

System.out.println(calculateTrappedWater(testCase4));

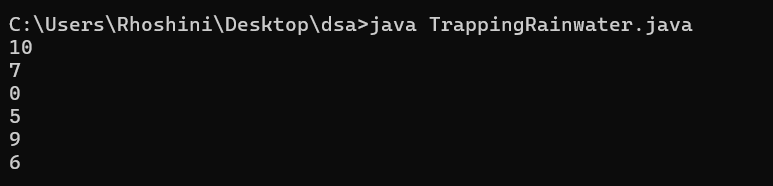
System.out.println(calculateTrappedWater(testCase5));

System.out.println(calculateTrappedWater(testCase6));

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(n)

1. **Chocolate Distribution Problem**

import java.util.Arrays;

public class ChocolateDistribution {

public static int findMinDifference(int[] chocolatePackets, int students) {

int n = chocolatePackets.length;

if (students == 0 || n == 0) return 0;

Arrays.sort(chocolatePackets);

if (n < students) return -1;

int minDifference = Integer.MAX\_VALUE;

for (int i = 0; i + students - 1 < n; i++) {

int difference = chocolatePackets[i + students - 1] - chocolatePackets[i];

minDifference = Math.min(minDifference, difference);

}

return minDifference;

}

public static void main(String[] args) {

int[] testCase1 = {7, 3, 2, 4, 9, 12, 56};

int[] testCase2 = {7, 3, 2, 4, 9, 12, 56};

int[] testCase3 = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int[] testCase4 = {5, 10, 15, 20, 25};

int[] testCase5 = {1, 2, 4, 5, 6, 8, 10};

int[] testCase6 = {100, 200, 300, 350, 400};

System.out.println(findMinDifference(testCase1, 3));

System.out.println(findMinDifference(testCase2, 5));

System.out.println(findMinDifference(testCase3, 4));

System.out.println(findMinDifference(testCase4, 3));

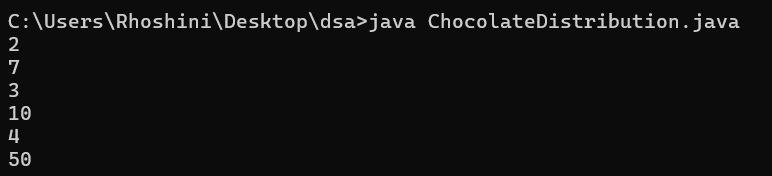
System.out.println(findMinDifference(testCase5, 4));

System.out.println(findMinDifference(testCase6, 2));

}

}

OUTPUT:



Time Complexitty: O(n log n )

Space Complexity: O(1)

**8. Merge Overlapping Intervals**

import java.util.Arrays;

import java.util.ArrayList;

public class MergeOverlappingIntervals {

public static ArrayList<int[]> mergeIntervals(int[][] intervals) {

if (intervals.length == 0) return new ArrayList<>();

Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

ArrayList<int[]> mergedIntervals = new ArrayList<>();

int[] currentInterval = intervals[0];

mergedIntervals.add(currentInterval);

for (int i = 1; i < intervals.length; i++) {

if (currentInterval[1] >= intervals[i][0]) {

currentInterval[1] = Math.max(currentInterval[1], intervals[i][1]);

} else {

currentInterval = intervals[i];

mergedIntervals.add(currentInterval);

}

}

return mergedIntervals;

}

public static void main(String[] args) {

int[][] testCase1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

int[][] testCase2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

int[][] testCase3 = {{1, 3}, {5, 7}, {8, 10}};

int[][] testCase4 = {{1, 3}, {3, 5}, {6, 8}, {7, 9}};

int[][] testCase5 = {{1, 2}, {3, 4}, {5, 6}};

int[][] testCase6 = {{1, 10}, {2, 6}, {8, 12}};

ArrayList<int[]> result1 = mergeIntervals(testCase1);

ArrayList<int[]> result2 = mergeIntervals(testCase2);

ArrayList<int[]> result3 = mergeIntervals(testCase3);

ArrayList<int[]> result4 = mergeIntervals(testCase4);

ArrayList<int[]> result5 = mergeIntervals(testCase5);

ArrayList<int[]> result6 = mergeIntervals(testCase6);

printIntervals(result1);

printIntervals(result2);

printIntervals(result3);

printIntervals(result4);

printIntervals(result5);

printIntervals(result6);

}

private static void printIntervals(ArrayList<int[]> intervals) {

for (int[] interval : intervals) {

System.out.print("[" + interval[0] + ", " + interval[1] + "] ");

}

System.out.println();

}

}

OUTPUT:



Time Complexitty: O(n log n )

Space Complexity: O(n)

**9. A Boolean Matrix Question**

public class BooleanMatrix {

public static void modifyMatrix(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

boolean[] rowFlag = new boolean[m];

boolean[] colFlag = new boolean[n];

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

if (matrix[i][j] == 1) {

rowFlag[i] = true;

colFlag[j] = true;

}

}

}

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

if (rowFlag[i] || colFlag[j]) {

matrix[i][j] = 1;

}

}

}

}

public static void printMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

int[][] testCase1 = {{1, 0}, {0, 0}};

int[][] testCase2 = {{0, 0, 0}, {0, 0, 1}};

int[][] testCase3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

int[][] testCase4 = {{0, 0}, {0, 0}};

int[][] testCase5 = {{1, 1, 1}, {0, 0, 0}, {1, 0, 1}};

int[][] testCase6 = {{0, 1}, {1, 0}};

int[][] testCase7 = {{0}};

int[][] testCase8 = {{1, 0}, {0, 1}};

modifyMatrix(testCase1);

printMatrix(testCase1);

modifyMatrix(testCase2);

printMatrix(testCase2);

modifyMatrix(testCase3);

printMatrix(testCase3);

modifyMatrix(testCase4);

printMatrix(testCase4);

modifyMatrix(testCase5);

printMatrix(testCase5);

modifyMatrix(testCase6);

printMatrix(testCase6);

modifyMatrix(testCase7);

printMatrix(testCase7);

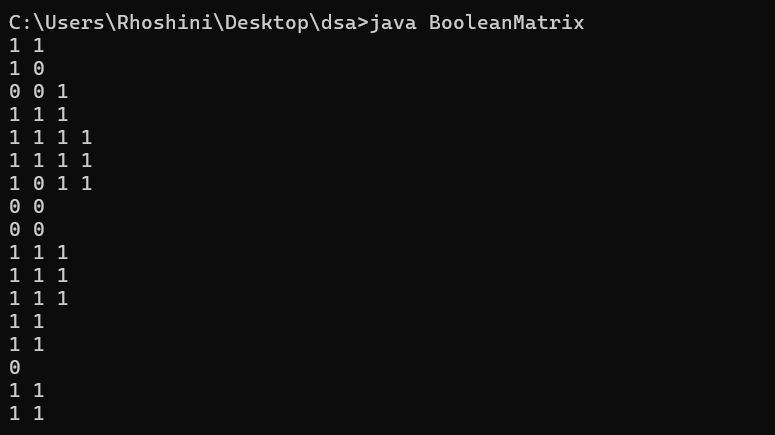
modifyMatrix(testCase8);

printMatrix(testCase8);

}

}

OUTPUT:



Time Complexitty: O(MN)

Space Complexity: O(M+N)

**10. Print a given matrix in spiral form**

public class SpiralMatrix {

public static void printSpiral(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

int top = 0, left = 0, bottom = m - 1, right = n - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) {

System.out.print(matrix[top][i] + " ");

}

top++;

for (int i = top; i <= bottom; i++) {

System.out.print(matrix[i][right] + " ");

}

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--) {

System.out.print(matrix[bottom][i] + " ");

}

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--) {

System.out.print(matrix[i][left] + " ");

}

left++;

}

}

}

public static void main(String[] args) {

int[][] testCase1 = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}}; // Hidden test case 1

int[][] testCase2 = {{1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}}; // Hidden test case 2

int[][] testCase3 = {{1}}; // Hidden test case 3

int[][] testCase4 = {{1, 2}, {3, 4}}; // Hidden test case 4

int[][] testCase5 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

printSpiral(testCase1);

System.out.println();

printSpiral(testCase2);

System.out.println();

printSpiral(testCase3);

System.out.println();

printSpiral(testCase4);

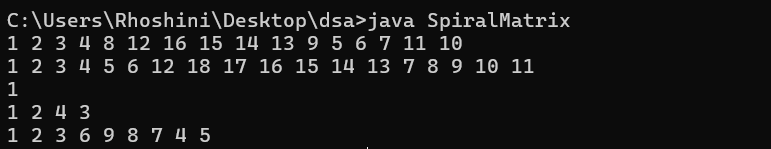
System.out.println();

printSpiral(testCase5);

}

}

OUTPUT:



Time Complexitty: O(MN)

Space Complexity: O(1)

**13. Check if given Parentheses expression is balanced or not**

import java.util.Stack;

public class ParenthesesBalance {

public static String checkBalance(String str) {

Stack<Character> stack = new Stack<>();

for (int i = 0; i < str.length(); i++) {

char ch = str.charAt(i);

if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {

if (stack.isEmpty()) {

return "Not Balanced";

}

stack.pop();

}

}

return stack.isEmpty() ? "Balanced" : "Not Balanced";

}

public static void main(String[] args) {

String testCase1 = "((()))()()"; // Hidden test case 1

String testCase2 = "())((())"; // Hidden test case 2

String testCase3 = "(((())))"; // Hidden test case 3

String testCase4 = "()()()"; // Hidden test case 4

String testCase5 = "((())())";

System.out.println(checkBalance(testCase1));

System.out.println(checkBalance(testCase2));

System.out.println(checkBalance(testCase3));

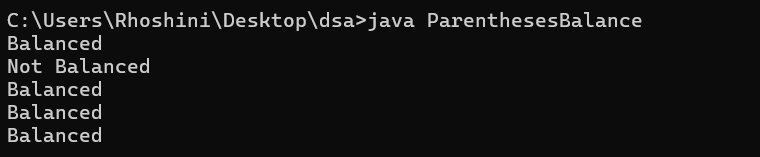
System.out.println(checkBalance(testCase4));

System.out.println(checkBalance(testCase5));

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(n)

**14. Check if two Strings are Anagrams of each other**

import java.util.Arrays;

public class AnagramChecker {

public static boolean areAnagrams(String s1, String s2) {

if (s1.length() != s2.length()) {

return false;

}

char[] arr1 = s1.toCharArray();

char[] arr2 = s2.toCharArray();

Arrays.sort(arr1);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

public static void main(String[] args) {

String testCase1 = "geeks";

String testCase2 = "kseeg";

String testCase3 = "allergy";

String testCase4 = "allergic";

String testCase5 = "g";

String testCase6 = "g"; // Hidden test case 1

String testCase7 = "silent"; // Hidden test case 2

String testCase8 = "listen"; // Hidden test case 3

String testCase9 = "hello"; // Hidden test case 4

System.out.println(areAnagrams(testCase1, testCase2));

System.out.println(areAnagrams(testCase3, testCase4));

System.out.println(areAnagrams(testCase5, testCase6));

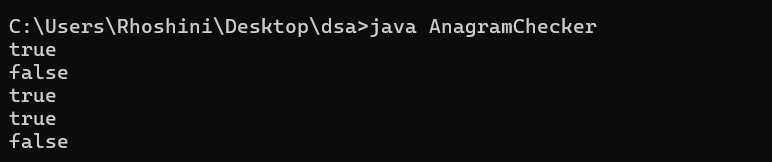
System.out.println(areAnagrams(testCase7, testCase8));

System.out.println(areAnagrams(testCase9, testCase1));

}

}

OUTPUT:



Time Complexitty: O(n log n)

Space Complexity: O(n)

**15. Longest Palindromic Substring**

public class LongestPalindromicSubstring {

public static String longestPalindrome(String str) {

if (str == null || str.length() < 1) {

return "";

}

int start = 0, maxLength = 1;

for (int i = 0; i < str.length(); i++) {

int len1 = expandFromCenter(str, i, i);

int len2 = expandFromCenter(str, i, i + 1);

int len = Math.max(len1, len2);

if (len > maxLength) {

maxLength = len;

start = i - (maxLength - 1) / 2;

}

}

return str.substring(start, start + maxLength);

}

private static int expandFromCenter(String str, int left, int right) {

while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {

left--;

right++;

}

return right - left - 1;

}

public static void main(String[] args) {

String testCase1 = "forgeeksskeegfor";

String testCase2 = "Geeks";

String testCase3 = "abc";

String testCase4 = "";

String testCase5 = "babad"; // Hidden test case 1

String testCase6 = "civic"; // Hidden test case 2

String testCase7 = "aabbcc"; // Hidden test case 3

String testCase8 = "racecar"; // Hidden test case 4

System.out.println(longestPalindrome(testCase1));

System.out.println(longestPalindrome(testCase2));

System.out.println(longestPalindrome(testCase3));

System.out.println(longestPalindrome(testCase4));

System.out.println(longestPalindrome(testCase5));

System.out.println(longestPalindrome(testCase6));

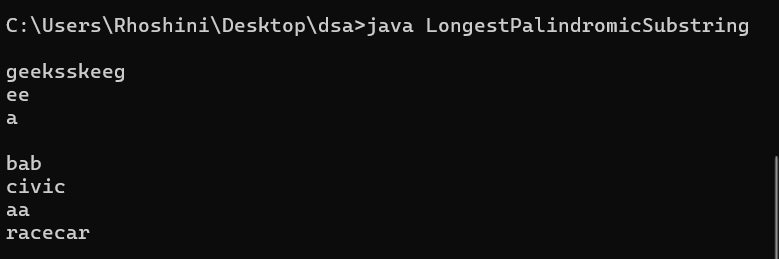
System.out.println(longestPalindrome(testCase7));

System.out.println(longestPalindrome(testCase8));

}

}

OUTPUT:



Time Complexitty: O(n^2)

Space Complexity: O(1)

**16. Longest Common Prefix using Sorting**

import java.util.Arrays;

public class LongestCommonPrefix {

public static String longestCommonPrefix(String[] arr) {

if (arr == null || arr.length == 0) {

return "-1";

}

Arrays.sort(arr);

String first = arr[0];

String last = arr[arr.length - 1];

int minLength = Math.min(first.length(), last.length());

int i = 0;

while (i < minLength && first.charAt(i) == last.charAt(i)) {

i++;

}

String commonPrefix = first.substring(0, i);

return commonPrefix.isEmpty() ? "-1" : commonPrefix;

}

public static void main(String[] args) {

String[] testCase1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

String[] testCase2 = {"hello", "world"};

String[] testCase3 = {"apple", "ape", "apricot", "appliance"};

String[] testCase4 = {"abcd", "abef", "ab"};

String[] testCase5 = {"cat", "catalog", "caterpillar"}; // Hidden test case 1

String[] testCase6 = {"dog", "race", "car"}; // Hidden test case 2

String[] testCase7 = {"king", "kind", "kiss", "kid"}; // Hidden test case 3

String[] testCase8 = {"java", "jazz", "jupiter"}; // Hidden test case 4

System.out.println(longestCommonPrefix(testCase1));

System.out.println(longestCommonPrefix(testCase2));

System.out.println(longestCommonPrefix(testCase3));

System.out.println(longestCommonPrefix(testCase4));

System.out.println(longestCommonPrefix(testCase5));

System.out.println(longestCommonPrefix(testCase6));

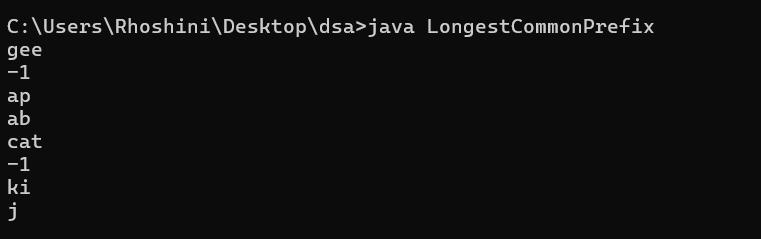
System.out.println(longestCommonPrefix(testCase7));

System.out.println(longestCommonPrefix(testCase8));

}

}

OUTPUT:



Time Complexitty: O(n lod n+ m)

Space Complexity: O(1)

**17. Delete middle element of a stack**

import java.util.Stack;

public class DeleteMiddleElement {

public static void deleteMiddle(Stack<Integer> stack, int size, int currentIndex) {

if (stack.isEmpty() || currentIndex == size / 2) {

stack.pop();

return;

}

int top = stack.pop();

deleteMiddle(stack, size, currentIndex + 1);

stack.push(top);

}

public static void main(String[] args) {

Stack<Integer> stack1 = new Stack<>();

stack1.push(1);

stack1.push(2);

stack1.push(3);

stack1.push(4);

stack1.push(5);

deleteMiddle(stack1, stack1.size(), 0);

System.out.println(stack1);

Stack<Integer> stack2 = new Stack<>();

stack2.push(1);

stack2.push(2);

stack2.push(3);

stack2.push(4);

stack2.push(5);

stack2.push(6);

deleteMiddle(stack2, stack2.size(), 0);

System.out.println(stack2);

// Hidden test cases

Stack<Integer> stack3 = new Stack<>();

stack3.push(10);

stack3.push(20);

stack3.push(30);

stack3.push(40);

stack3.push(50);

deleteMiddle(stack3, stack3.size(), 0);

System.out.println(stack3);

Stack<Integer> stack4 = new Stack<>();

stack4.push(1);

stack4.push(2);

stack4.push(3);

stack4.push(4);

stack4.push(5);

stack4.push(6);

stack4.push(7);

deleteMiddle(stack4, stack4.size(), 0);

System.out.println(stack4);

Stack<Integer> stack5 = new Stack<>();

stack5.push(100);

stack5.push(200);

stack5.push(300);

deleteMiddle(stack5, stack5.size(), 0);

System.out.println(stack5);

Stack<Integer> stack6 = new Stack<>();

stack6.push(10);

stack6.push(20);

stack6.push(30);

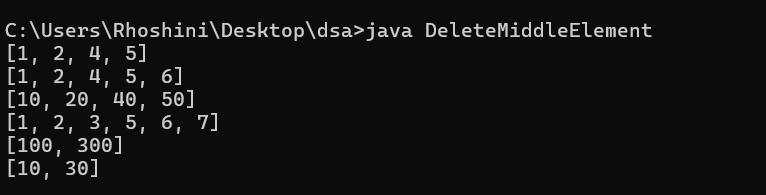
deleteMiddle(stack6, stack6.size(), 0);

System.out.println(stack6);

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(n)

**18. Next Greater Element (NGE) for every element in given Array**

import java.util.Stack;

public class NextGreaterElement {

public static void printNextGreater(int[] arr) {

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

int[] nge = new int[arr.length];

for (int i = 0; i < arr.length; i++) {

nge[i] = -1;

}

for (int i = 0; i < arr.length; i++) {

while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {

int index = stack.pop();

nge[index] = arr[i];

}

stack.push(i);

}

for (int i = 0; i < arr.length; i++) {

System.out.println(nge[i]);

}

}

public static void main(String[] args) {

int[] arr1 = {4, 5, 2, 25};

printNextGreater(arr1);

int[] arr2 = {13, 7, 6, 12};

printNextGreater(arr2);

// Hidden test cases

int[] arr3 = {10, 5, 3, 7, 8};

printNextGreater(arr3);

int[] arr4 = {15, 8, 4, 2, 1};

printNextGreater(arr4);

int[] arr5 = {3, 4, 2, 1, 6};

printNextGreater(arr5);

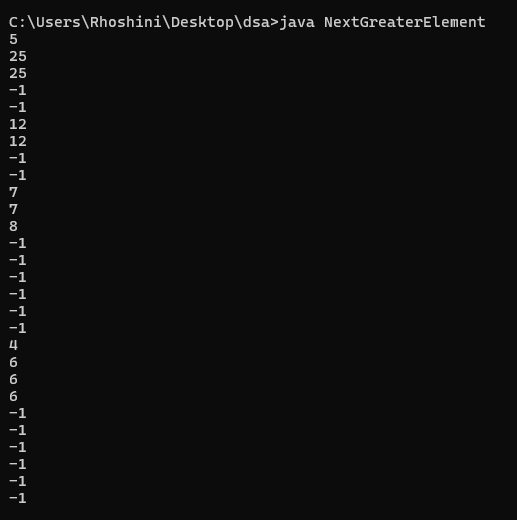
int[] arr6 = {5, 4, 3, 2, 1};

printNextGreater(arr6);

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(n)

**19. Print Right View of a Binary Tree**

import java.util.\*;

class BinaryTree {

static class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public static void printRightView(Node root) {

if (root == null) return;

Queue<Node> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int n = queue.size();

for (int i = 1; i <= n; i++) {

Node node = queue.poll();

if (i == n) {

System.out.print(node.data + " ");

}

if (node.left != null) queue.add(node.left);

if (node.right != null) queue.add(node.right);

}

}

}

public static void main(String[] args) {

Node root1 = new Node(1);

root1.left = new Node(2);

root1.right = new Node(3);

root1.left.left = new Node(4);

root1.left.right = new Node(5);

root1.right.right = new Node(6);

root1.left.left.left = new Node(7);

System.out.println("Right View of Binary Tree 1:");

printRightView(root1);

Node root2 = new Node(10);

root2.left = new Node(20);

root2.right = new Node(30);

root2.left.left = new Node(40);

root2.right.left = new Node(60);

root2.right.right = new Node(70);

System.out.println("\nRight View of Binary Tree 2:");

printRightView(root2);

// Hidden test cases

Node root3 = new Node(1);

root3.left = new Node(2);

root3.right = new Node(3);

root3.left.right = new Node(4);

root3.right.left = new Node(5);

root3.right.right = new Node(6);

System.out.println("\nRight View of Binary Tree 3:");

printRightView(root3);

Node root4 = new Node(1);

root4.left = new Node(2);

root4.right = new Node(3);

root4.left.left = new Node(4);

root4.left.right = new Node(5);

root4.right.left = new Node(6);

System.out.println("\nRight View of Binary Tree 4:");

printRightView(root4);

Node root5 = new Node(1);

root5.left = new Node(2);

root5.left.left = new Node(3);

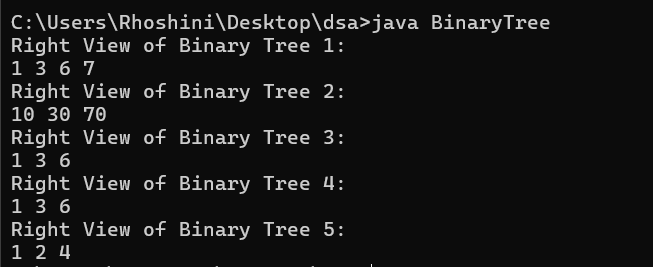
root5.left.right = new Node(4);

System.out.println("\nRight View of Binary Tree 5:");

printRightView(root5); }

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(h)

**20. Maximum Depth or Height of Binary Tree**

class MaxDepthBinaryTree {

static class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public static int maxDepth(Node root) {

if (root == null) {

return 0;

}

int leftDepth = maxDepth(root.left);

int rightDepth = maxDepth(root.right);

return Math.max(leftDepth, rightDepth) + 1;

}

public static void main(String[] args) {

MaxDepthBinaryTree tree = new MaxDepthBinaryTree();

Node root1 = new Node(1);

root1.left = new Node(2);

root1.right = new Node(3);

root1.left.left = new Node(4);

root1.left.right = new Node(5);

root1.right.right = new Node(6);

root1.left.left.left = new Node(7);

System.out.println("Maximum Depth of Binary Tree 1: " + tree.maxDepth(root1));

Node root2 = new Node(10);

root2.left = new Node(20);

root2.right = new Node(30);

root2.left.left = new Node(40);

root2.right.left = new Node(60);

root2.right.right = new Node(70);

System.out.println("Maximum Depth of Binary Tree 2: " + tree.maxDepth(root2));

// Hidden test cases

Node root3 = new Node(1);

root3.left = new Node(2);

root3.left.left = new Node(3);

System.out.println("Maximum Depth of Binary Tree 3: " + tree.maxDepth(root3));

Node root4 = new Node(1);

root4.left = new Node(2);

root4.left.left = new Node(3);

root4.left.left.left = new Node(4);

System.out.println("Maximum Depth of Binary Tree 4: " + tree.maxDepth(root4));

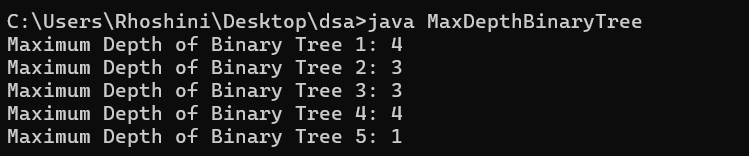
Node root5 = new Node(1);

System.out.println("Maximum Depth of Binary Tree 5: " + tree.maxDepth(root5));

}

}

OUTPUT:



Time Complexitty: O(n)

Space Complexity: O(h)