9.11.2024 Day-6 Practice test 1  
  
1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.   
Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11 Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.  
 Input: arr[] = {-2, -4} Output: –2 Explanation: The subarray {-2} has the largest sum -2. Input: arr[] = {5, 4, 1, 7, 8} Output: 25 Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.  
  
**solution:**  
package maxsubarraysum;

import java.util.Scanner;

import java.util.Arrays;

class GfG {

static int[] maxSubarraySum(int[] arr) {

int maxSum = arr[0];

int maxEnding = arr[0];

int start = 0;

int end = 0;

int tempStart = 0;

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

if (arr[i] > maxEnding + arr[i]) {

maxEnding = arr[i];

tempStart = i;

} else {

maxEnding += arr[i];

}

if (maxEnding > maxSum) {

maxSum = maxEnding;

start = tempStart;

end = i;

}

}

int[] maxSubarray = Arrays.copyOfRange(arr, start, end + 1);

System.out.println("The maximum subarray is: " + Arrays.toString(maxSubarray));

return new int[] { maxSum };

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of 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();

}

System.out.println("The input array is: " + Arrays.toString(arr));

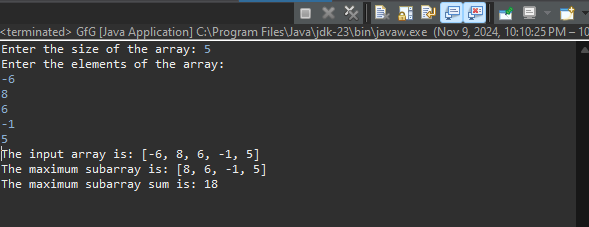
int[] result = maxSubarraySum(arr);

System.out.println("The maximum subarray sum is: " + result[0]);

scanner.close();

}

}

Output:  


Time complexity: O(n)  
  
2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.  
 Input: arr[] = {-2, 6, -3, -10, 0, 2}   
Output: 180 Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10) = 180   
Input: arr[] = {-1, -3, -10, 0, 60} Output: 60 Explanation: The subarray with maximum product is {60}.  
  
Solution:

package productarray;

import java.util.Scanner;

public class maxprodarray {

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[] 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 maxProduct = maxProduct(nums);

System.out.println("Maximum Product: " + maxProduct);

}

public static int maxProduct(int[] nums) {

int max = Integer.MIN\_VALUE;

int pro = 1;

int tempStart = 0, start = 0, end = 0;

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

pro \*= nums[i];

if (max < pro) {

max = pro;

start = tempStart;

end = i;

}

if (pro == 0) {

pro = 1;

tempStart = i + 1;

}

}

pro = 1;

tempStart = nums.length - 1;

for (int i = nums.length - 1; i >= 0; i--) {

pro \*= nums[i];

if (max < pro) {

max = pro;

start = i;

end = tempStart;

}

if (pro == 0) {

pro = 1;

tempStart = i - 1;

}

}

System.out.print("Subarray with maximum product: ");

for (int i = start; i <= end; i++) {

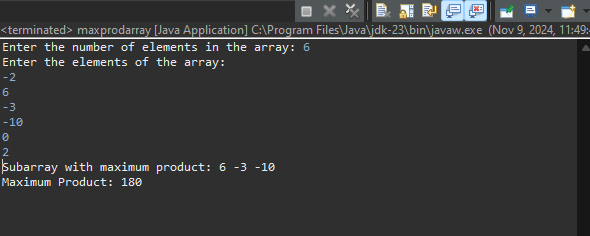
System.out.print(nums[i] + " ");

}

System.out.println();

return max;

}

}  


Time complexity O(n)

3. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.  
 Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0 Output : 4   
Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3 Output : -1  
 Input : arr[] = {50, 10, 20, 30, 40}, key = 10 Output : 1  
  
Solution  
  
package sortedandrotated;

import java.util.Scanner;

class sortrot{

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[] 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.print("Enter the target value: ");

int target = scanner.nextInt();

sortrot solution = new sortrot();

int result = solution .search(nums, target);

if (result != -1) {

System.out.println("Target found at index: " + result);

} else {

System.out.println("Target not found in the array.");

}

}

public int search(int[] nums, int target) {

int low = 0, high = nums.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (nums[mid] == target) {

return mid;

}

if (nums[low] <= nums[mid]) {

if (nums[low] <= target && target < nums[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

}

else {

if (nums[mid] < target && target <= nums[high]) {

low = mid + 1;

} else {

high = mid - 1;

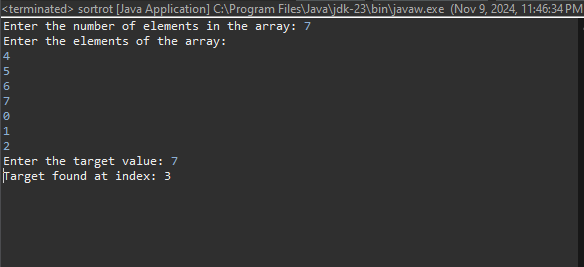
}

}

}

return -1;

}

}  


Time Complexity O(log n )  
  
  
  
  
4. Containg more water  
Input: arr = [1, 5, 4, 3] Output: 6 Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6 Input: arr = [3, 1, 2, 4, 5] Output: 12 Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12  
  
solution:

package sortedandrotated;

import java.util.Scanner;

class Containwater {

public int maxArea(int[] height) {

int maxArea = 0;

int left = 0;

int right = height.length - 1;

while (left < right) {

maxArea = Math.max(maxArea, (right - left) \* Math.min(height[left], height[right]));

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return maxArea;

}

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[] height = new int[n];

System.out.println("Enter the elements (heights) separated by space:");

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

height[i] = scanner.nextInt();

}

Containwater solution = new Containwater();

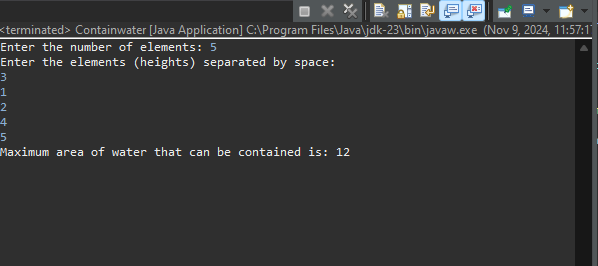
int maxWaterArea = solution.maxArea(height);

System.out.println("Maximum area of water that can be contained is: " + maxWaterArea);

scanner.close();

}

}

  
  
Time Complexity O(n)  
  
5. Find the Factorial of a large number   
Input: 100 Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00   
Input: 50 Output: 30414093201713378043612608166064768844377641568960512000000000000  
  
  
Solution  
package factorial;

import java.math.BigInteger;

import java.util.Scanner;

public class Factlarge

{

public static BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

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

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

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number to calculate its factorial: ");

int number = scanner.nextInt();

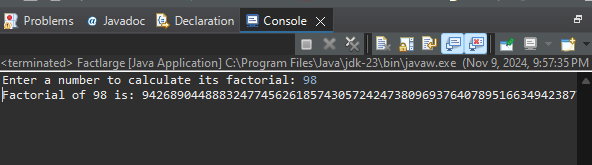
BigInteger factorialResult = factorial(number);

System.out.println("Factorial of " + number + " is: " + factorialResult);

scanner.close();

}

}

  
Time Complexity: O(n)

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.   
Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image.   
Input: arr[] = {3, 0, 2, 0, 4} Output: 7 Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.   
Input: arr[] = {1, 2, 3, 4} Output: 0 Explanation : We cannot trap water as there is no height bound on both sides   
Input: arr[] = {10, 9, 0, 5} Output: 5 Explanation : We trap 0 + 0 + 5 + 0 = 5  
  
package maxsubarraysum;

import java.util.Scanner;

class Traprainwater {

public int trap(int[] height) {

int left = 0;

int right = height.length - 1;

int leftMax = height[left];

int rightMax = height[right];

int water = 0;

while (left < right) {

if (leftMax < rightMax) {

left++;

leftMax = Math.max(leftMax, height[left]);

water += leftMax - height[left];

} else {

right--;

rightMax = Math.max(rightMax, height[right]);

water += rightMax - height[right];

}

}

return water;

}

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[] height = new int[n];

System.out.println("Enter the heights separated by space:");

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

height[i] = scanner.nextInt();

}

Traprainwater solution = new Traprainwater();

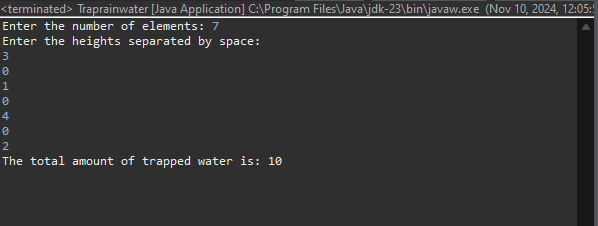
int trappedWater = solution.trap(height);

System.out.println("The total amount of trapped water is: " + trappedWater);

scanner.close();

}

}



Time complexity O(n)

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized. Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3 Output: 2 Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2. Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5 Output: 7 Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7  
  
package sortedandrotated;

import java.util.Arrays;

import java.util.Scanner;

class Chocolatedistri {

static int findMinimumDifference(int[] packets, int students) {

int numPackets = packets.length;

Arrays.sort(packets);

int minimumDifference = Integer.MAX\_VALUE;

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

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

if (difference < minimumDifference)

minimumDifference = difference;

}

return minimumDifference;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of packets: ");

int n = scanner.nextInt();

int[] packets = new int[n];

System.out.println("Enter the sizes of the packets:");

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

packets[i] = scanner.nextInt();

}

System.out.print("Enter the number of students: ");

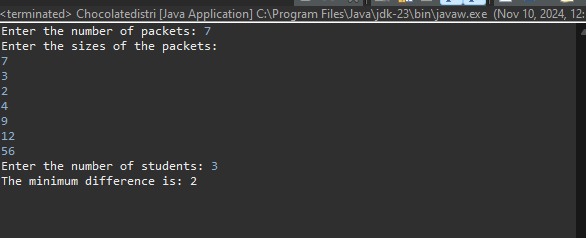
int students = scanner.nextInt();

System.out.println("The minimum difference is: " + findMinimumDifference(packets, students));

scanner.close();

}

}

  
  
time complexity O(n log n)

8.Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.  
 Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]] Output: [[1, 4], [6, 8], [9, 10]] Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].  
 Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]] Output: [[1, 6], [7, 8]] Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].  
  
package sortedandrotated;

import java.util.\*;

class Mergedintervals {

public int[][] merge(int[][] intervals) {

int numIntervals = intervals.length;

Arrays.sort(intervals, new Comparator<int[]>() {

public int compare(int[] interval1, int[] interval2) {

return interval1[0] - interval2[0];

}

});

List<List<Integer>> mergedList = new ArrayList<>();

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

if (mergedList.isEmpty() || intervals[i][0] > mergedList.get(mergedList.size() - 1).get(1)) {

mergedList.add(Arrays.asList(intervals[i][0], intervals[i][1]));

} else {

mergedList.get(mergedList.size() - 1).set(1, Math.max(mergedList.get(mergedList.size() - 1).get(1), intervals[i][1]));

}

}

int[][] mergedIntervals = new int[mergedList.size()][2];

for (int i = 0; i < mergedList.size(); i++) {

mergedIntervals[i][0] = mergedList.get(i).get(0);

mergedIntervals[i][1] = mergedList.get(i).get(1);

}

return mergedIntervals;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of intervals: ");

int intervalCount = scanner.nextInt();

int[][] intervals = new int[intervalCount][2];

System.out.println("Enter each interval in the format [start end]:");

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

intervals[i][0] = scanner.nextInt();

intervals[i][1] = scanner.nextInt();

}

Mergedintervals solution = new Mergedintervals ();

int[][] result = solution.merge(intervals);

System.out.println("Merged Intervals:");

for (int[] interval : result) {

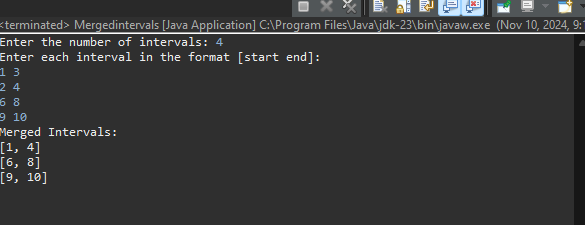
System.out.println(Arrays.toString(interval));

}

scanner.close();

}

}

  
  
**Time Complexity:** O(nlogn)

**9.** A Boolean Matrix Question

Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

Input: {{1, 0}, {0, 0}}

Output: {{1, 1} {1, 0}}

**Program:**

import java.util.\*;

public class Problem9{

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

int numberOfRows = matrix.length;

int numberOfCols = matrix[0].length;

int[] rows = new int[numberOfRows];

int[] cols = new int[numberOfCols];

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

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

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

rows[i] = 1;

cols[j] = 1;

}

}

}

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

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

if (rows[i] == 1 || cols[j] == 1) {

matrix[i][j] = 1;

}

}

}

}

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

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

for (int j = 0; j < matrix[0].length; j++) {

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

}

System.out.println();

}

}

public static void main(String[] args) {

int[][] matrix = {

{1, 0, 0, 1},

{0, 0, 1, 0},

{0, 0, 0, 0},

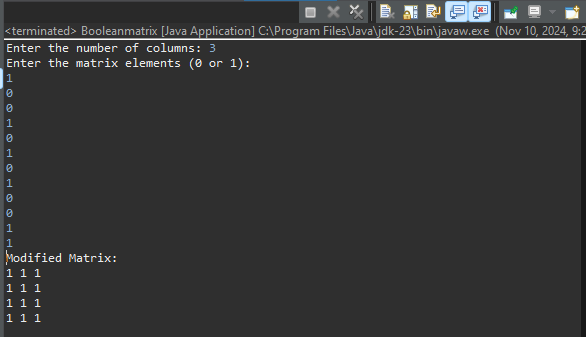
};

System.out.println("Modified Matrix :");

modifyMatrix(matrix);

printMatrix(matrix);

}

}  
  
  
**Time Complexity:** O(mxn)

10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form. Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}} Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11 Explanation: The output is matrix in spiral format.  
  
package sortedandrotated;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class SpiralMatrix {

public static List<Integer> getSpiralOrder(int[][] matrix) {

List<Integer> result = new ArrayList<>();

int rows = matrix.length;

int cols = matrix[0].length;

int top = 0, left = 0, bottom = rows - 1, right = cols - 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;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of rows: ");

int rows = scanner.nextInt();

System.out.print("Enter the number of columns: ");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

System.out.println("Enter the matrix elements:");

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

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

matrix[i][j] = scanner.nextInt();

}

}

List<Integer> result = getSpiralOrder(matrix);

System.out.println("Spiral order of the matrix:");

for (int num : result) {

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

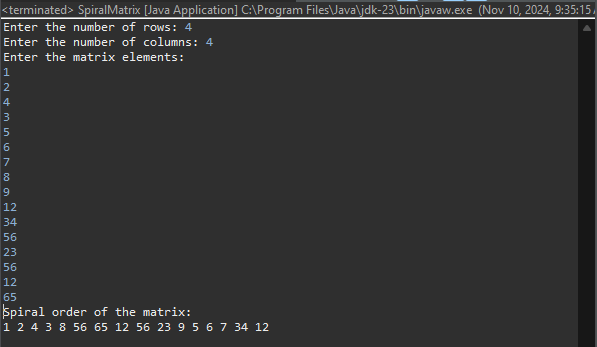
}

System.out.println();

scanner.close();

}

}

  
  
**Time Complexity:** O(n x m)

13. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not. Input: str = “((()))()()” Output: Balanced Input: str = “())((())” Output: Not Balanced  
  
package sortedandrotated;

import java.util.Scanner;

import java.util.Stack;

public class Parenexpress {

public static String checkBalance(String expression) {

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

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

char currentChar = expression.charAt(i);

if (currentChar == '(') {

stack.push(currentChar);

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

if (stack.isEmpty()) {

return "Not Balanced";

}

stack.pop();

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string of parentheses: ");

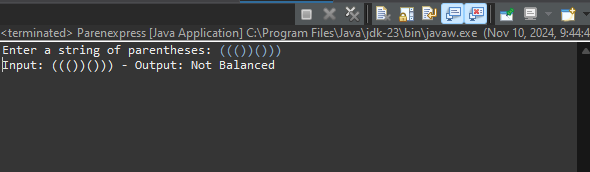
String input = scanner.next();

System.out.println("Input: " + input + " - Output: " + checkBalance(input));

scanner.close();

}

}

  
  
Time complexity O(n)  
  
14. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. Input: s1 = “geeks” s2 = “kseeg” Output: true Explanation: Both the string have same characters with same frequency. So, they are anagrams. Input: s1 = “allergy” s2 = “allergic” Output: false Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams. Input: s1 = “g”, s2 = “g” Output: true Explanation: Characters in both the strings are same, so they are anagrams  
  
import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class Anagram {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first string: ");

String firstString = scanner.nextLine();

System.out.print("Enter the second string: ");

String secondString = scanner.nextLine();

if (firstString.length() != secondString.length()) {

System.out.println("The strings are not anagrams.");

scanner.close();

return;

}

Map<Character, Integer> charCount = new HashMap<>();

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

char currentChar = firstString.charAt(i);

charCount.put(currentChar, charCount.getOrDefault(currentChar, 0) + 1);

}

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

char currentChar = secondString.charAt(i);

if (!charCount.containsKey(currentChar) || charCount.get(currentChar) == 0) {

System.out.println("The strings are not anagrams.");

scanner.close();

return;

}

charCount.put(currentChar, charCount.get(currentChar) - 1);

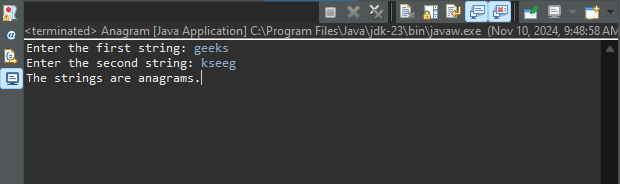
}

System.out.println("The strings are anagrams.");

scanner.close();

}

}

  
Time complexity O(n)  
  
15. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring. Input: str = “forgeeksskeegfor” Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all. Input: str = “Geeks” Output: “ee” Input: str = “abc” Output: “a” Input: str = “” Output: “”  
  
import java.util.Scanner;

public class PalindromeSubstring {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String inputString = scanner.nextLine();

String longestPalindrome = "";

int maxLength = 0;

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

int left = i, right = i;

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

if ((right - left + 1) > maxLength) {

longestPalindrome = inputString.substring(left, right + 1);

maxLength = right - left + 1;

}

left--;

right++;

}

left = i;

right = i + 1;

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

if ((right - left + 1) > maxLength) {

longestPalindrome = inputString.substring(left, right + 1);

maxLength = right - left + 1;

}

left--;

right++;

}

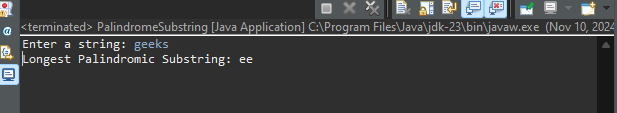
}

System.out.println("Longest Palindromic Substring: " + longestPalindrome);

scanner.close();

}

}



Time complexity O(n)  
  
16. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”. Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”] Output: gee Explanation: “gee” is the longest common prefix in all the given strings. Input: arr[] = [“hello”, “world”] Output: -1 Explanation: There‟s no common prefix in the given strings.  
  
import java.util.\*;

public class CommonPrefix {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of strings: ");

int numStrings = scanner.nextInt();

String[] strings = new String[numStrings];

System.out.println("Enter the strings: ");

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

strings[i] = scanner.next();

}

if (strings.length == 0) {

System.out.println("-1");

return;

}

Arrays.sort(strings);

String firstString = strings[0];

String lastString = strings[strings.length - 1];

StringBuilder commonPrefix = new StringBuilder();

for (int i = 0; i < Math.min(firstString.length(), lastString.length()); i++) {

if (firstString.charAt(i) == lastString.charAt(i)) {

commonPrefix.append(firstString.charAt(i));

} else {

break;

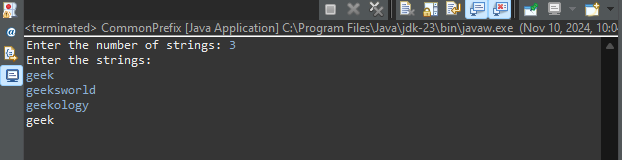
}

}

System.out.println(commonPrefix.length() > 0 ? commonPrefix.toString() : "-1");

}

}

  
**Time Complexity:** O(n log n+m)

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

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

import java.util.\*;

public class RemoveMiddleElement {

public static void removeMiddle(Stack<Integer> stack, int currentIndex, int middleIndex) {

if (currentIndex == middleIndex) {

stack.pop();

return;

}

int top = stack.pop();

removeMiddle(stack, currentIndex + 1, middleIndex);

stack.push(top);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter the number of elements in the stack: ");

int numElements = scanner.nextInt();

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

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

int element = scanner.nextInt();

stack.push(element);

}

System.out.println("Original Stack: " + stack);

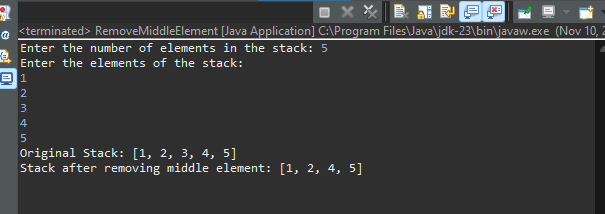
int middleIndex = numElements / 2;

removeMiddle(stack, 0, middleIndex);

System.out.println("Stack after removing middle element: " + stack);

}

}

  
  
Time complexity O(n)  
  
18. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1. Input: arr[] = [ 4 , 5 , 2 , 25 ] Output: 4 –> 5 5 –> 25 2 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side Input: arr[] = [ 13 , 7, 6 , 12 ] Output: 13 –> -1 7 –> 12 6 –> 12 12 –> -1  
  
import java.util.\*;

public class NextGreaterElement {

public static void findNextGreaterElements(int[] arr) {

int n = arr.length;

int[] nextGreater = new int[n];

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

for (int i = n - 1; i >= 0; i--) {

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

stack.pop();

}

nextGreater[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

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

System.out.println(arr[i] + " -> " + nextGreater[i]);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int arraySize = scanner.nextInt();

int[] array = new int[arraySize];

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

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

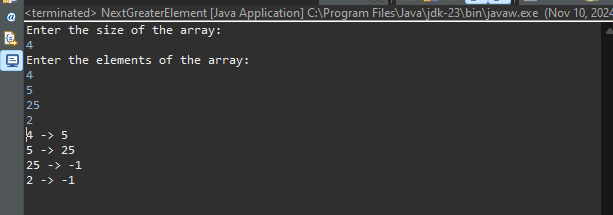
array[i] = scanner.nextInt();

}

findNextGreaterElements(array);

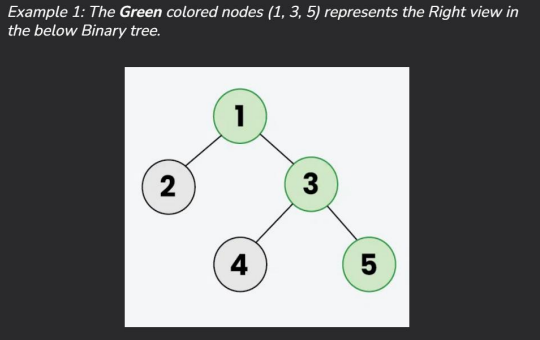
}

}

  
  
**Time Complexity:** O(n)

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

Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.



import java.util.\*;

class TreeNode {

int value;

TreeNode left;

TreeNode right;

TreeNode(int x) {

value = x;

left = null;

right = null;

}

}

class RightSideViewTree {

public List<Integer> getRightSideView(TreeNode root) {

List<Integer> rightView = new ArrayList<>();

viewRightSide(root, rightView, 0);

return rightView;

}

public void viewRightSide(TreeNode node, List<Integer> rightView, int depth) {

if (node == null) {

return;

}

if (depth == rightView.size()) {

rightView.add(node.value);

}

viewRightSide(node.right, rightView, depth + 1);

viewRightSide(node.left, rightView, depth + 1);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.right = new TreeNode(6);

root.left.left.left = new TreeNode(7);

RightSideViewTree solution = new RightSideViewTree();

List<Integer> rightViewList = solution.getRightSideView(root);

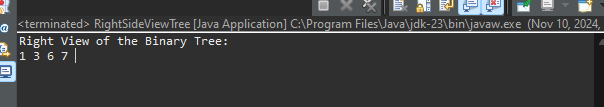
System.out.println("Right View of the Binary Tree:");

for (Integer value : rightViewList) {

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

}

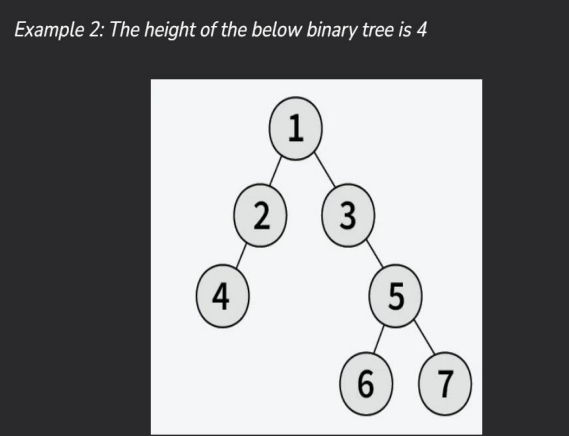
}

}  


**Time Complexity:** O(n)

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

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

  
  
class TreeNode {

int value;

TreeNode left;

TreeNode right;

TreeNode(int x) {

value = x;

left = null;

right = null;

}

}

class MaxDepthBinaryTree {

public int getMaxDepth(TreeNode node) {

if (node == null) {

return 0;

}

int leftDepth = getMaxDepth(node.left);

int rightDepth = getMaxDepth(node.right);

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

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.right = new TreeNode(6);

root.left.left.left = new TreeNode(7);

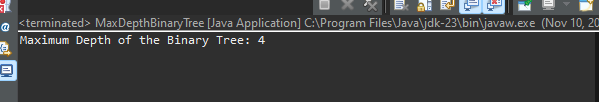
MaxDepthBinaryTree solution = new MaxDepthBinaryTree();

int maxDepth = solution.getMaxDepth(root);

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

}

}

  
Time complexity O(n)