GATE Day – 6 Practice

Coding Questions

**1. Maximum Sum subarray**

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.

**Code:**

import java.util.Scanner;

public class MaximumSubarray {

public static int maxsubarray(int[] nums){

int cursum =0, maxsubsum = nums[0];

for(int n : nums){

if (cursum < 0) cursum = 0;

cursum += n;

maxsubsum = Math.max(cursum, maxsubsum);

}

return maxsubsum;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

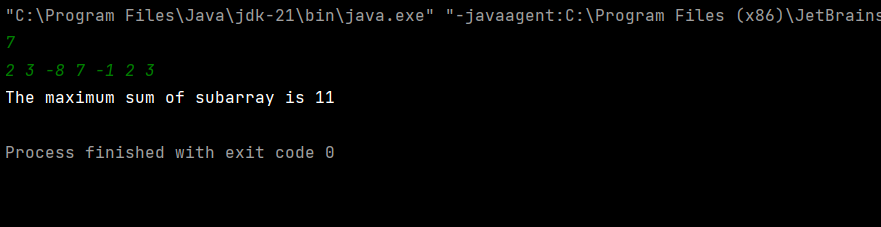
nums[i] = sc.nextInt();

}

System.out.println("The maximum sum of subarray is " + maxsubarray(nums));

}

}



Time Complexity: O(n)

we are looking into every element of the array in the sequence and by being greedy eliminating the possiblilites of negitive values to maximise the sum

**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

**Code:**

import java.util.Scanner;

public class MaximumProductSubarray {

public static int maximumProduct(int[] nums){

// initialize the variables

int maxSoFar = nums[0], minSoFar = nums[0];

int result = maxSoFar;

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

int n = nums[i];

int tempSoFar = Math.max(n, Math.max(maxSoFar\* n, minSoFar\*n));

minSoFar = Math.min(n, Math.min(maxSoFar\*n, minSoFar\*n));

maxSoFar = tempSoFar;

result = Math.max(result, maxSoFar);

}

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

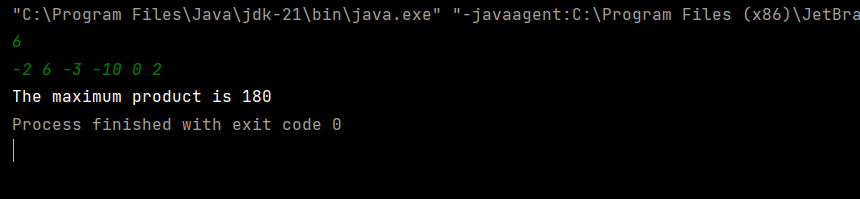
nums[i] = sc.nextInt();

}

System.out.printf("The maximum product is %d", maximumProduct(nums));

}

}



**Time Complexity:** O (n)

The numbers here contains both positive as well as negative so we have to be mindful while using greedy approach of checking the products of every element simultaneously.

3. Search in Rotated Sorted 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

**Code:**

import java.util.Scanner;

public class SearchRotatedSortedArray {

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

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

while(left <= right){

int mid = (left + right) /2;

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

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

if (nums[left] <= target && target <= nums[mid]) right = mid -1;

else left = mid + 1;

}

else {

if (nums[mid] <= target && target <= nums[right]) left = mid+1;

else right = mid -1;

}

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

nums[i] = sc.nextInt();

}

int result = search(nums, 0);

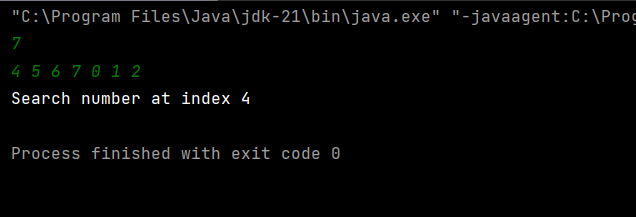
if (result == -1) System.out.println("Search number not found");

else System.out.println("Search number at index " + result);

sc.close();

}

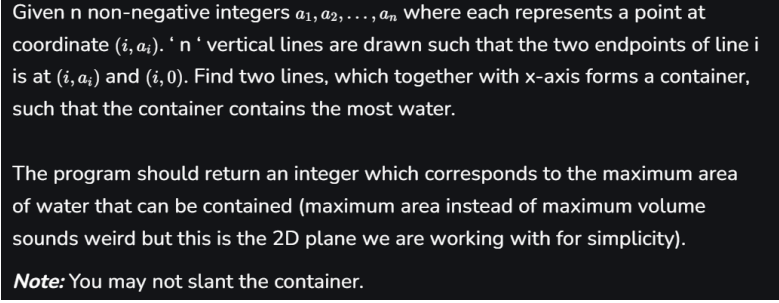
}



Time Complexity: O(logn)

Modified Binary Search

4. Container With the most 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

**Code:**

import java.util.Scanner;

public class ContainerWithMostWater {

public static int maxArea(int[] nums){

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

int maxarea = 0;

while(left < right){

int area = Math.min(nums[left], nums[right]) \* (right - left);

maxarea = Math.max(maxarea, area);

if (nums[left] < nums[right]) left++;

else right--;

}

return maxarea;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

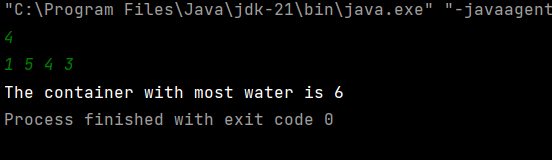
nums[i] = sc.nextInt();

}

System.out.printf("The container with most water is %d", maxArea(nums));

}

}



Time Complexity: O(n)

Two Pointer approach and maximising area minimising the height and still comparing left and right for the greater height.

5. Find the factorial of the large number

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

**Code:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Scanner;

public class LargeFactorials {

public static ArrayList<Integer> factorial(int n) {

ArrayList<Integer> res = new ArrayList<>();

res.add(1);

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

multiply(i, res);

}

// Reverse the result to have the most significant digit at the start

Collections.reverse(res);

return res;

}

static void multiply(int x, ArrayList<Integer> res) {

int carry = 0;

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

int prod = res.get(i) \* x + carry;

res.set(i, prod % 10);

carry = prod / 10;

}

while (carry != 0) {

res.add(carry % 10);

carry = carry / 10;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); // You can change this to any number to test other factorials

// Call the factorial method

ArrayList<Integer> result = factorial(n);

// Print the factorial result

System.out.print("Factorial of " + n + " is: ");

for (int digit : result) {

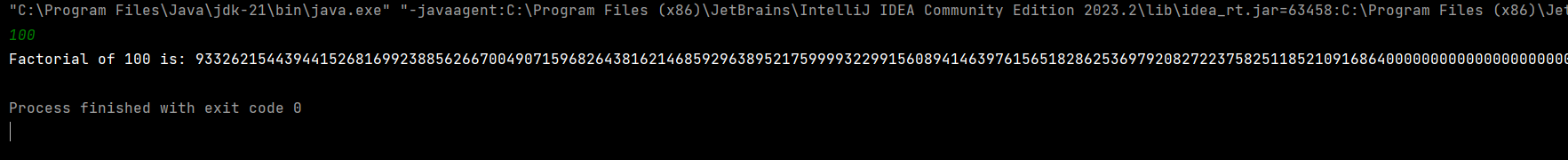
System.out.print(digit);

}

System.out.println();

}

}



Time Complexity: O(Nlog(N!))

where the n is for the for loop and n! Is for the nested while loop

6.Trapping Rain Water

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.

**Code:**

import java.util.Scanner;

public class TrappingRainWater {

public static int trapWater(int[] heights){

int i = 0, leftMax = heights[0], sum = 0;

int j = heights.length -1 , rightMax = heights[j];

while(i < j){

if (leftMax < rightMax){

sum += leftMax - heights[i];

i++;

leftMax = Math.max(leftMax, heights[i]);

}

else {

sum += rightMax - heights[j];

j--;

rightMax = Math.max(rightMax, heights[i]);

}

}

return sum;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

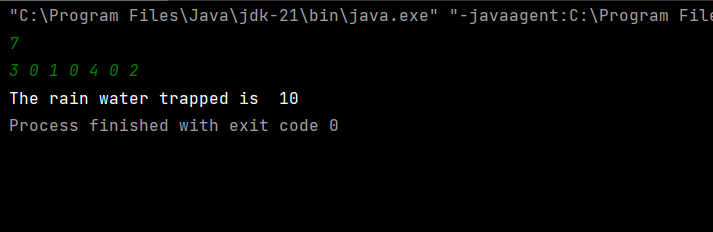
nums[i] = sc.nextInt();

}

System.out.printf("The rain water trapped is %d", trapWater(nums));

}

}



Time Complexity O(n)

Two pointer approach

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

**Code:**

import java.util.Arrays;

import java.util.Scanner;

public class ChocolateDistribution {

public static int findMinDifference(int[] arr, int m) {

if (m > arr.length) {

return -1;

}

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

for (int i = 0; i + m - 1 < arr.length; i++) {

int diff = arr[i + m - 1] - arr[i];

minDiff = Math.min(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int m = sc.nextInt();

int[] nums = new int[n];

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

nums[i] = sc.nextInt();

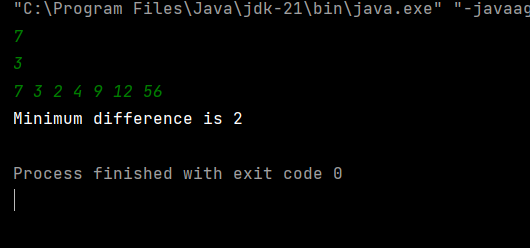
}

int result = findMinDifference(nums, m);

System.out.println("Minimum difference is " + result);

}

}



Time Complexity : O(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]].

**Code:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class MergeInterval {

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

if (intervals.length <= 1) {

return intervals;

}

// Sort intervals based on the start time

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

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

// Start with the first interval

int[] currentInterval = intervals[0];

mergedIntervals.add(currentInterval);

for (int[] interval : intervals) {

int currentEnd = currentInterval[1];

// If the current interval overlaps with the next one, merge them

if (interval[0] <= currentEnd) {

currentInterval[1] = Math.max(currentEnd, interval[1]);

} else {

// No overlap, add the current interval to the list and move to the next

currentInterval = interval;

mergedIntervals.add(currentInterval);

}

}

// Convert the list to a 2D array

return mergedIntervals.toArray(new int[mergedIntervals.size()][]);

}

public static void main(String[] args) {

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

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

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

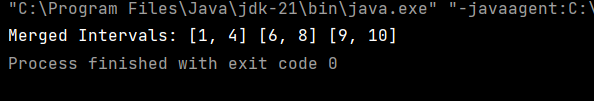
for (int[] interval : result) {

System.out.print(Arrays.toString(interval) + " ");

}

}

}



Time Complexity : O(nlogn + n) = O(nlogn)

nlogn is for sorting

n is for loop traversal

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}}

Input: {{0, 0, 0},

{0, 0, 1}}

Output: {{0, 0, 1},

{1, 1, 1}}

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

{0, 0, 1, 0},

{0, 0, 0, 0}}

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

{1, 1, 1, 1},

{1, 0, 1, 1}}

**Code:**

import java.util.Scanner;

public class BooleanMatrix{

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

int rows = matrix.length;

int cols = matrix[0].length;

// Arrays to keep track of rows and columns to be updated

boolean[] rowFlags = new boolean[rows];

boolean[] colFlags = new boolean[cols];

// First pass: mark the rows and columns that need to be updated

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

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

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

rowFlags[i] = true;

colFlags[j] = true;

}

}

}

// Second pass: update the matrix based on the flags

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

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

if (rowFlags[i] || colFlags[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) {

Scanner sc = new Scanner(System.in);

// Get the dimensions of the matrix from the user

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

int rows = sc.nextInt();

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

int cols = sc.nextInt();

// Initialize the matrix

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

// Fill the matrix with user input

System.out.println("Enter the matrix elements (0 or 1):");

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

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

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

}

}

// Print the original matrix

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

printMatrix(matrix);

// Modify the matrix based on the problem requirements

modifyMatrix(matrix);

// Print the modified matrix

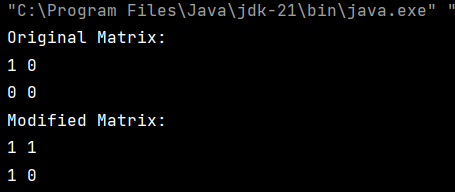
System.out.println("Modified Matrix:");

printMatrix(matrix);

sc.close();

}

}



Time Complexity : O(n) + O(n) with of O(2n) space

usage of two two for loops and with the help of two arrays

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

Code:

import java.util.\*;

public class SpiralMatrix {

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

ArrayList<Integer> res = new ArrayList<>();

int n = matrix.length;

int m = matrix[0].length;

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

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

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

res.add(matrix[top][i]);

top++;

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

res.add(matrix[i][right]);

right--;

if (top <= bottom) {

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

res.add(matrix[bottom][i]);

bottom--;

}

if (left <= right) {

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

res.add(matrix[i][left]);

left++;

}

}

return res;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input matrix dimensions

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

int rows = scanner.nextInt();

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

int cols = scanner.nextInt();

// Input matrix elements

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

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

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

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

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

}

}

List<Integer> result = spiralOrder(matrix);

// Output the result

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

for (int num : result) {

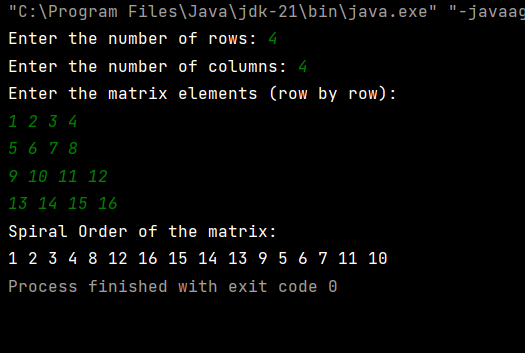
System.out.print(num + " ");

}

scanner.close();

}

}



Time Complxity: O(n\*m) due to nexted for loops

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

import java.util.\*;

public class ValidParenthesis {

// Optimized function to check if parentheses are balanced

public static boolean isBalanced(String s) {

Deque<Character> stack = new ArrayDeque<>(); // Use Deque instead of Stack for better performance

for (char ch : s.toCharArray()) {

// If it's an opening parenthesis, push it onto the stack

if (ch == '(') {

stack.push(ch);

}

// If it's a closing parenthesis

else if (ch == ')') {

// Check if the stack is empty (unbalanced) or the top element is not matching

if (stack.isEmpty()) {

return false; // No matching opening parenthesis

}

stack.pop(); // Pop the matching opening parenthesis

}

}

// If the stack is empty, all parentheses matched, otherwise unbalanced

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str = scanner.nextLine();

// Check balance and print result

if (isBalanced(str)) {

System.out.println("Balanced");

} else {

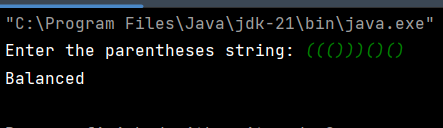
System.out.println("Not Balanced");

}

scanner.close();

}

}



Time complexity : O(n) traversing the entire string

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.

Code:

import java.util.Arrays;

import java.util.Scanner;

public class ValidAnagram {

public static boolean isAnagram(String s, String t) {

if (s.length() != t.length()) {

return false;

}

// Convert strings to char arrays

char[] sArray = s.toCharArray();

char[] tArray = t.toCharArray();

// Sort the char arrays

Arrays.sort(sArray);

Arrays.sort(tArray);

// Compare sorted arrays

return Arrays.equals(sArray, tArray);

}

public static void main(String[] args) {

// Initialize Scanner to take input

Scanner scanner = new Scanner(System.in);

// Prompt user for input

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

String s = scanner.nextLine();

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

String t = scanner.nextLine();

// Check if the two strings are anagrams

if (isAnagram(s, t)) {

System.out.println("true");

} else {

System.out.println("false");

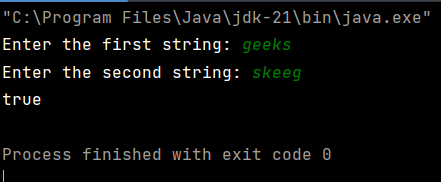
}

// Close the scanner

scanner.close();

}

}



Time Complexity: O(nlogn) – due to sorting

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”

Code:  
  
import java.util.Scanner;

public class LongestPalindromicSubstring {

public static String longestPalindrome(String s) {

if (s.length() <= 1) {

return s;

}

String maxStr = s.substring(0, 1);

for (int i = 0; i < s.length() - 1; i++) {

String odd = expandFromCenter(s, i, i);

String even = expandFromCenter(s, i, i + 1);

if (odd.length() > maxStr.length()) {

maxStr = odd;

}

if (even.length() > maxStr.length()) {

maxStr = even;

}

}

return maxStr;

}

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

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

left--;

right++;

}

return s.substring(left + 1, right);

}

public static void main(String[] args) {

// Initialize Scanner to take input

Scanner scanner = new Scanner(System.in);

// Prompt user for input

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

String s = scanner.nextLine();

// Find the longest palindrome

String result = longestPalindrome(s);

// Output the result

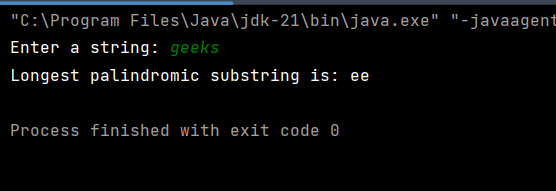
System.out.println("Longest palindromic substring is: " + result);

// Close the scanner

scanner.close();

}

}



Time Complexity is O(n)

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.

Code:

import java.util.\*;

public class LongestPrefix{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of Strings : ");

int n=sc.nextInt();

String[] arr=new String[n];

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

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

{

arr[i]=sc.next();

}

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

System.out.println(-1);

}

Arrays.sort(arr);

String first = arr[0];

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

StringBuilder commonPrefix = new StringBuilder();

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

if (first.charAt(i) == last.charAt(i)) {

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

} else {

break;

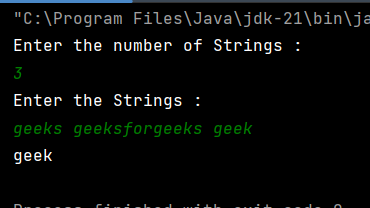
}

}

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

}

}



Time Complexity: O(m+n)

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]

Input : Stack[] = [1, 2, 3, 4, 5, 6]

import java.util.\*;

public class DeleteMiddleElementStack {

// Helper method to delete the middle element from the stack

private void deleteMiddleHelper(Stack<Integer> stack, int currentIndex, int middleIndex) {

// Base case: if we've reached the middle element, pop it

if (currentIndex == middleIndex) {

stack.pop();

return;

}

// Pop the top element and recursively delete the middle element

int top = stack.pop();

deleteMiddleHelper(stack, currentIndex + 1, middleIndex);

// Push the popped element back to the stack after the middle element is deleted

stack.push(top);

}

// Method to delete the middle element of the stack

public void deleteMiddleElement(Stack<Integer> stack) {

int size = stack.size();

if (size == 0) return; // If the stack is empty, there's nothing to delete

int middleIndex = size / 2;

deleteMiddleHelper(stack, 0, middleIndex);

}

public static void main(String[] args) {

// Create a scanner to take user input

Scanner scanner = new Scanner(System.in);

// Create an object of the class to access methods

DeleteMiddleElementStack solution = new DeleteMiddleElementStack();

// Get the number of elements in the stack

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

int n = scanner.nextInt();

// Initialize the stack and get user input to fill the stack

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

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

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

stack.push(scanner.nextInt());

}

// Call the method to delete the middle element

solution.deleteMiddleElement(stack);

// Print the stack after deleting the middle element

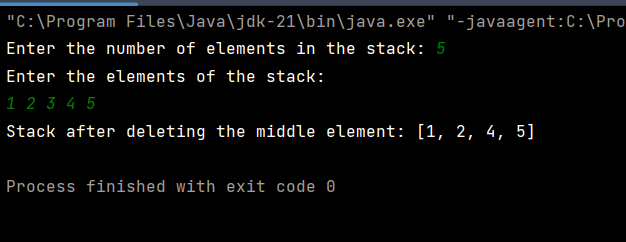
System.out.println("Stack after deleting the middle element: " + stack);

// Close the scanner

scanner.close();

}

}



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

import java.util.Stack;

import java.util.\*;

public class NextGreaterElement {

public static void printNGE(int[] arr) {

int n = arr.length;

int[] nge = 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();

}

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

stack.push(arr[i]);

}

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

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

}

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

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

int n=sc.nextInt();

int[] arr = new int[n];

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

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

{

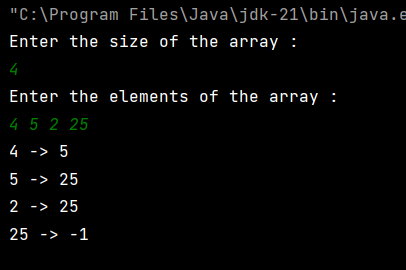
arr[i]=sc.nextInt();

}

printNGE(arr);

}

}



Time Complextiy : O(n)

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.\*;

public class RightMostView {

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

left = right = null;

}

}

public TreeNode insert(TreeNode root, int val) {

if (root == null) {

return new TreeNode(val);

}

if (val < root.val) {

root.left = insert(root.left, val);

} else {

root.right = insert(root.right, val);

}

return root;

}

public List<Integer> rightSideView(TreeNode root) {

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

if (root == null) {

return rightSide;

}

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

queue.offer(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

TreeNode rightmostNode = null;

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);

}

rightmostNode = node;

}

rightSide.add(rightmostNode.val);

}

return rightSide;

}

public static void main(String[] args) {

// Create an instance of the RightMostView class

RightMostView solution = new RightMostView();

// Initialize the Scanner for user input

Scanner scanner = new Scanner(System.in);

// Create a binary tree

TreeNode root = null;

// Get the number of nodes in the binary tree

System.out.print("Enter the number of nodes in the binary tree: ");

int n = scanner.nextInt();

System.out.println("Enter the values of the nodes (for binary search tree property):");

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

int val = scanner.nextInt();

root = solution.insert(root, val); // Insert each value into the binary tree

}

// Call the rightSideView method and get the result

List<Integer> rightSide = solution.rightSideView(root);

// Print the result (rightmost nodes at each level)

System.out.println("Right side view of the binary tree:");

for (int val : rightSide) {

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

}

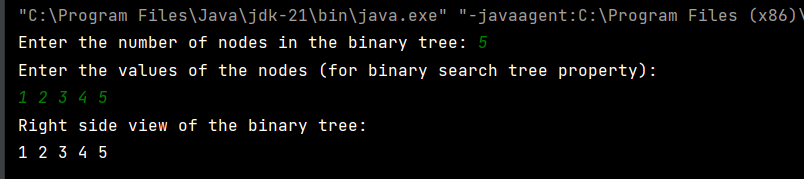
System.out.println();

// Close the scanner

scanner.close();

}

}



Time Complexity : O(n)

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 val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

class maximumDepthBinaryTree {

public int maxDepth(TreeNode root) {

if (root == null) {

return 0;

}

int lh = maxDepth(root.left);

int rh = maxDepth(root.right);

return 1 + Math.max(lh, rh);

}

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);

maximumDepthBinaryTree solution = new maximumDepthBinaryTree ();

int maxDepth = solution.maxDepth(root);

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

}

}

Time Complexity : O(n)

