PRIYANKA S

22AM041

AIML

**KADANE ALGORITHM**

1. Maximum Subarray Sum

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.

import java.util.Scanner;

public class KadaneSolution {

    public static int kadane(int[] nums) {

        int n = nums.length;

        int maxi = Integer.MIN\_VALUE, sum = 0;

        for (int i = 0; i < n; i++) {

            sum += nums[i];

            maxi = Math.max(sum, maxi);

            if (sum < 0) sum = 0;

        }

        return maxi;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter n");

        int n = sc.nextInt();

        int nums[] = new int[n];

        for (int i = 0; i < n; i++) {

            nums[i] = sc.nextInt();

        }

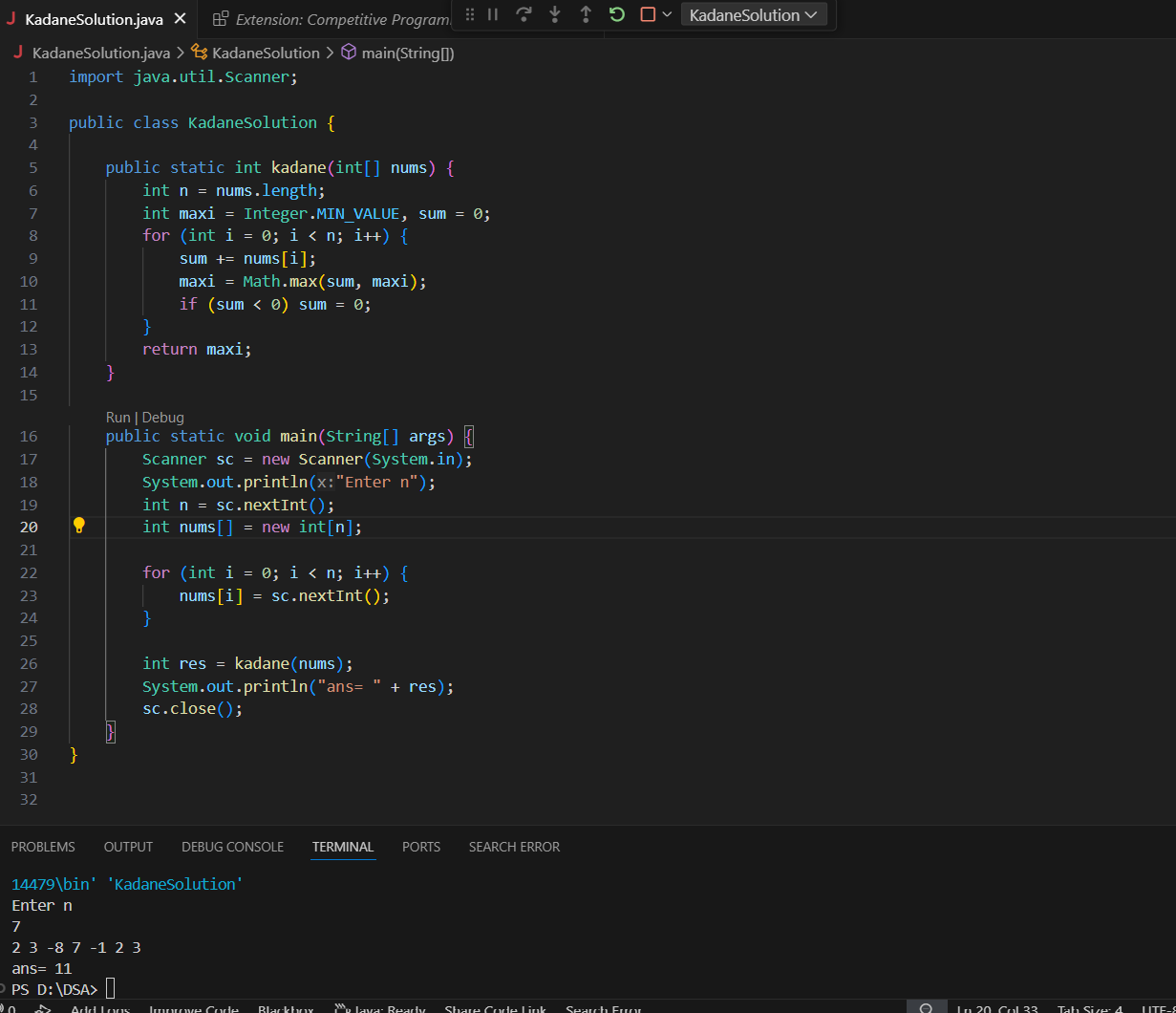
        int res = kadane(nums);

        System.out.println("ans= " + res);

        sc.close();

    }

}



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

class Solution {

    public int maxProduct(int[] nums) {

        int n= nums.length;

        int maxi = nums[0],mini=nums[0],res=nums[0];

        for(int i=1;i<n;i++){

            int curr=nums[i];

            if (curr<0){

                int temp =mini;

                mini=maxi;

                maxi=temp;

            }

            maxi=Math.max(curr,maxi\*curr);

            mini=Math.min(curr,mini\*curr);

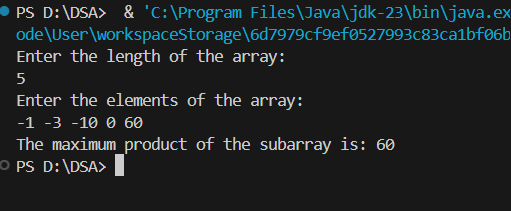
            res= Math.max(res,maxi);

        }

        return res;

    }

}



3) Search in a sorted and rotated Array

import java.util.Scanner;

class Solution {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the rotated array size: ");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

System.out.print("Enter the Target: ");

int x = s.nextInt();

int l = 0;

int r = n - 1;

int mid;

boolean found = false;

while (l <= r) {

mid = (l + r) / 2;

if (arr[mid] == x) {

System.out.println(mid);

found = true;

break;

}

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

if (arr[l] <= x && x < arr[mid]) {

r = mid - 1;

} else {

l = mid + 1;

}

} else {

if (arr[mid] < x && x <= arr[r]) {

l = mid + 1;

} else {

r = mid - 1;

}

}

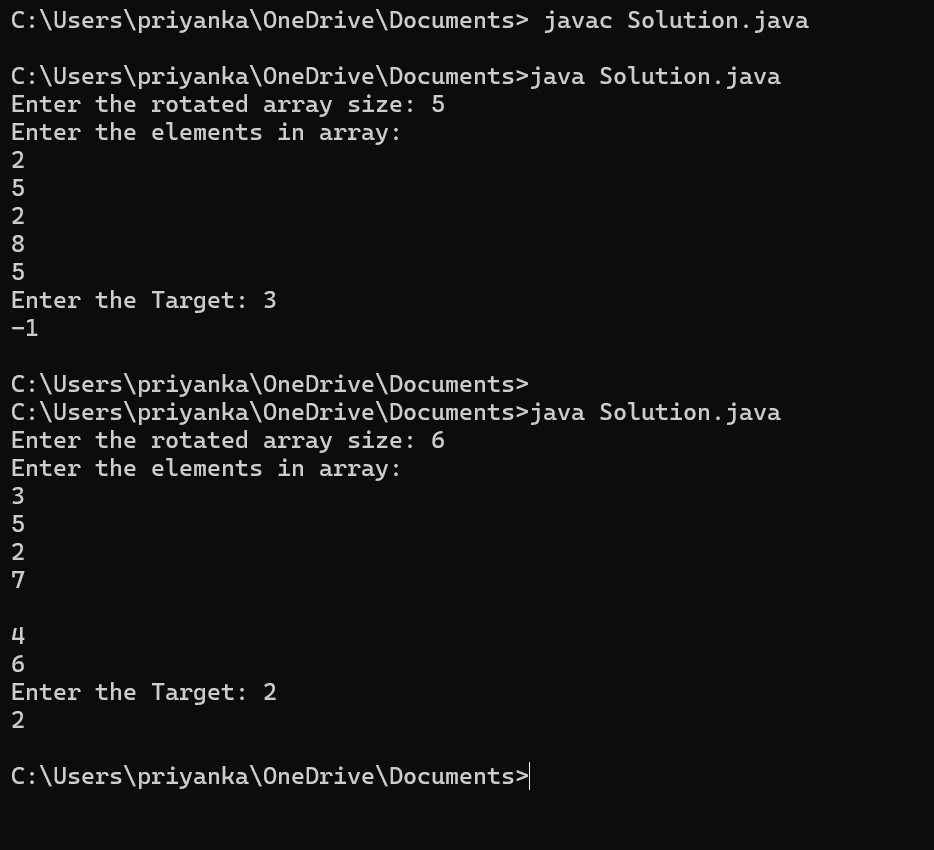
}

if (!found) {

System.out.println(-1);

}

}

}

Time complexity : O(log n)

4) Container with Most Water

CODE

import java.util.Scanner;

class Solution {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

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

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

int left = 0;

int right = n - 1;

int area = 0;

while (left < right) {

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

if (arr[left] < arr[right])

left += 1;

else

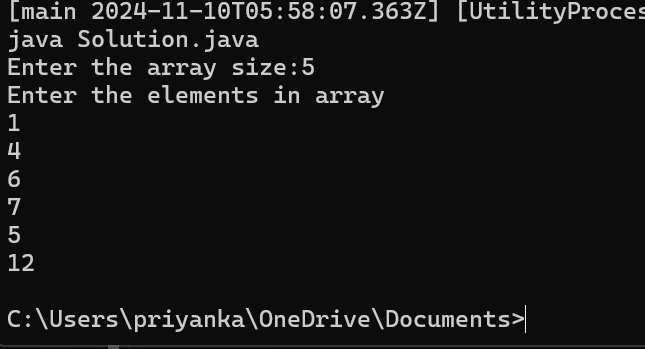
right -= 1;

}

System.out.println(area);

}

}



TIME COMPLEXITY : O(n)

5. Find the Factorial of a large number

Input: 100 Output:

933262154439441526816992388562667004907159682643816214685929638952175999932299

156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50 Output: 30414093201713378043612608166064768844377641568960512000000000000

class Solution {

static void factorial(int n)

{

int res[] = new int[500];

res[0] = 1;

int res\_size = 1;

for (int x = 2; x <= n; x++)

res\_size = multiply(x, res, res\_size);

System.out.println("Factorial of given number is ");

for (int i = res\_size - 1; i >= 0; i--)

System.out.print(res[i]);

}

static int multiply(int x, int res[], int res\_size)

{

int carry = 0;

for (int i = 0; i < res\_size; i++) {

int prod = res[i] \* x + carry;

res[i] = prod % 10;

carry = prod / 10;

}

while (carry != 0) {

res[res\_size] = carry % 10;

carry = carry / 10;

res\_size++;

}

return res\_size;

}

public static void main(String args[])

{

factorial(100);

}

}

Hidden TestCases:

Input: 134

C:\Users\priyanka\OneDrive\Documents>java Solution.java

Factorial of given number is

93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

C:\Users\priyanka\OneDrive\Documents>

Time Complexity: O(N^2.logn)

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

import java.util.Scanner;

public class Solution {

public static int trap(int[] height) {

int h = height.length;

if (height == null || h == 0) {

return 0;

}

int left = 0;

int right = h - 1;

int max\_left = 0;

int max\_right = 0;

int tot\_area = 0;

while (left < right) {

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

if (height[left] >= max\_left) {

max\_left = height[left];

} else {

tot\_area += max\_left - height[left];

}

left++;

} else {

if (height[right] >= max\_right) {

max\_right = height[right];

} else {

tot\_area += max\_right - height[right];

}

right--;

}

}

return tot\_area;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input length of the height array

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

int n = scanner.nextInt();

// Input elements of the height array

int[] height = new int[n];

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

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

height[i] = scanner.nextInt();

}

// Calculate and print the total trapped water

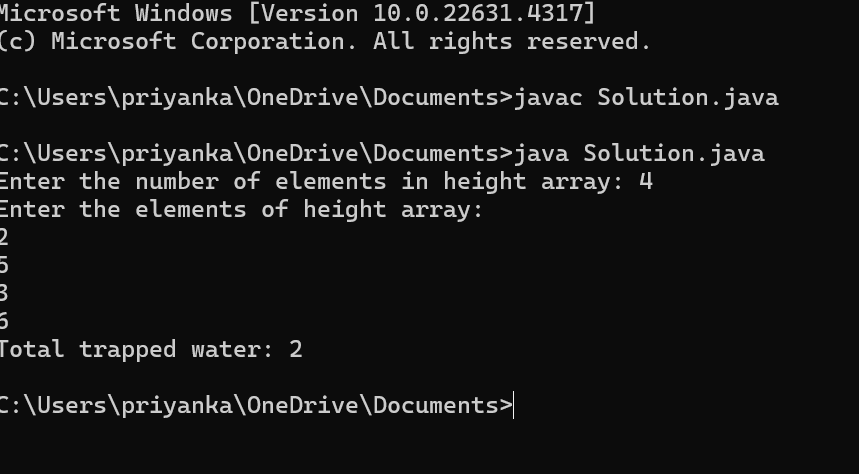
int result = trap(height);

System.out.println("Total trapped water: " + result);

scanner.close();

}

}



7) Chocolate Distribution Problem

import java.util.\*;

class Solution{

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the rotated array size: ");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

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

int m = s.nextInt();

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

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

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

if (diff < minDiff)

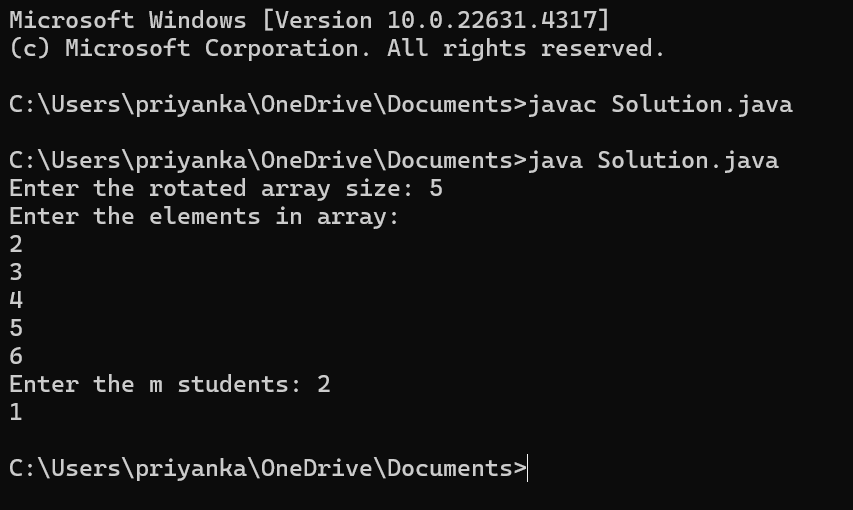
minDiff = diff;

}

System.out.println(minDiff);

}

}



TIME COMPLEXITY: nlog(n)

8) Merge Overlapping Intervals

CODE

import java.util.\*;

public classSolution {

static List<int[]> mergeOverlap(int[][] arr) {

int n = arr.length;

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

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

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

int start = arr[i][0];

int end = arr[i][1];

if (!res.isEmpty() && res.get(res.size() - 1)[1] >= start) {

res.get(res.size() - 1)[1] = Math.max(res.get(res.size() - 1)[1], end);

} else {

res.add(new int[] { start, end });

}

}

return res;

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

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

int n = s.nextInt();

int[][] arr = new int[n][2];

System.out.println("Enter the intervals (start and end): ");

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

arr[i][0] = s.nextInt();

arr[i][1] = s.nextInt();

}

List<int[]> res = mergeOverlap(arr);

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

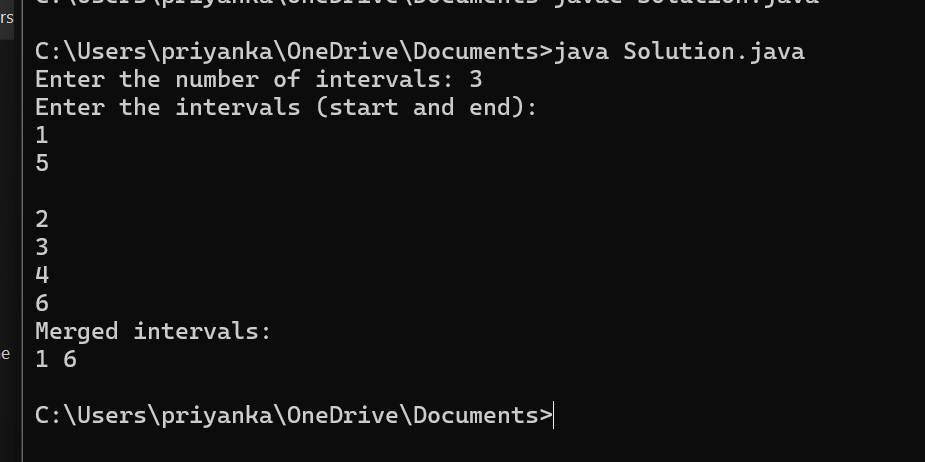
for (int[] interval : res) {

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

}

}

}



9) Boolean matrix

import java.util.Arrays;

public class Solution {

public static void main(String args[]){

int[][] matrix={{1, 0, 0, 1},

{0, 0, 1, 0},

{0, 0, 0, 0}};

int row=matrix.length;

int col=matrix[0].length;

boolean[] rowFlag=new boolean[row];

boolean[] colFlag=new boolean[col];

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

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

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

rowFlag[i]=true;

colFlag[j]=true;

}

}

}

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

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

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

matrix[i][j]=1;

}

}

}

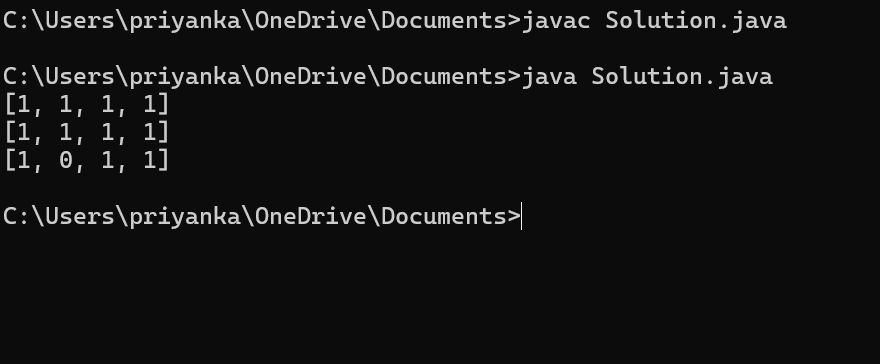
for(int[] entry:matrix) {

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

}

}

}



Time complexity : O(n \* m)

10. Print a given matrix in spiral form

import java.util.\*;

public class Solution {

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

int m = matrix.length;

int n = matrix[0].length;

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

if (m == 0)

return result;

boolean[][] seen = new boolean[m][n];

int[] dr = { 0, 1, 0, -1 };

int[] dc = { 1, 0, -1, 0 };

int r = 0, c = 0;

int di = 0;

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

result.add(matrix[r][c]);

seen[r][c] = true;

int newR = r + dr[di];

int newC = c + dc[di];

if (0 <= newR && newR < m && 0 <= newC && newC < n

&& !seen[newR][newC]) {

r = newR;

c = newC;

} else {

di = (di + 1) % 4;

r += dr[di];

c += dc[di];

}

}

return result;

}

public static void main(String[] args) {

int[][] matrix = {

{ 1, 2, 3, 4 },

{ 5, 6, 7, 8 },

{ 9, 10, 11, 12 },

{ 13, 14, 15, 16 }

};

List<Integer> result = spiralOrder(matrix);

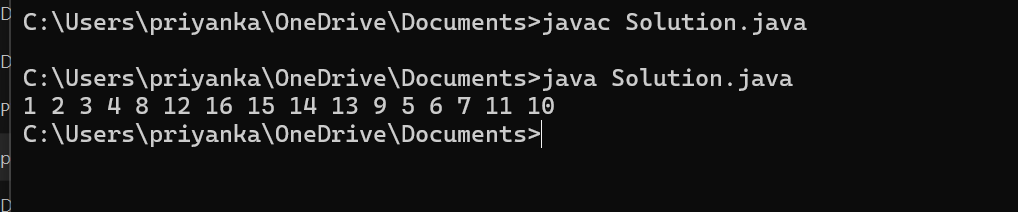
for (int num : result) {

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

}

}

}



13. Check if given Parentheses expression is balanced or not

import java.util.\*;

public class Solution {

public static void main(String[] args) {

String str="())((())";

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

for (char c : str.toCharArray()) {

if (!st.isEmpty() && c == ')' && st.peek() == '(') {

st.pop();

} else {

st.push(c);

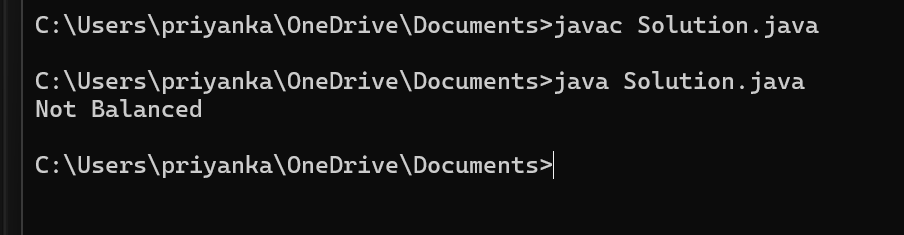
}

}

System.out.println(st.isEmpty()?"Balanced":"Not Balanced");

}

}



14. Check if two Strings are Anagrams of each other

public class Solution{

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

// Your code here

int[] freq=new int[26];

for(char c:s1.toCharArray()){

freq[c-'a']++;

}

for(char c:s2.toCharArray()){

freq[c-'a']--;

}

for(int i=0;i<26;i++){

if(freq[i]!=0){

return false;

}

}

return true;

}

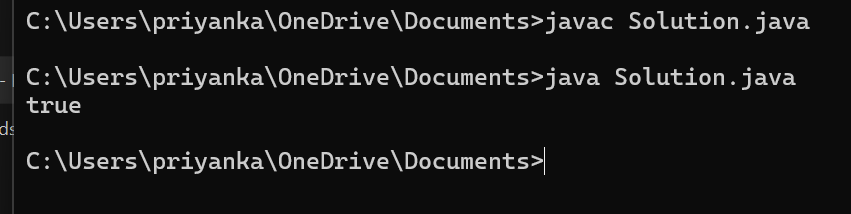
public static void main(String args[]) {

String s1="geeks";

String s2="kseeg";

System.out.println(areAnagrams(s1,s2));

}

}

15. Longest Palindromic Substring

public class Solution {

public static String expandAroundCenter(String str, int left, int right) {

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

left--;

right++;

}

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

}

public static String longestPalindrome(String str) {

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

return "";

}

String longest = "";

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

String oddPalindrome = expandAroundCenter(str, i, i);

String evenPalindrome = expandAroundCenter(str, i, i + 1);

if (oddPalindrome.length() > longest.length()) {

longest = oddPalindrome;

}

if (evenPalindrome.length() > longest.length()) {

longest = evenPalindrome;

}

}

return longest;

}

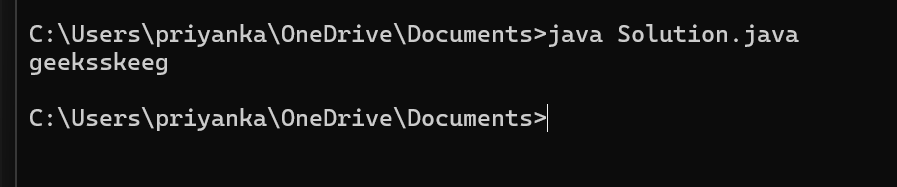
public static void main(String[] args) {

String str = "forgeeksskeegfor";

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

}

}



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]

Output : Stack[] = [1, 2, 4, 5, 6]

import java.util.Stack;

public class DeleteMiddleElement {

public static void deleteMid(Stack<Integer> s, int sizeOfStack) {

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

int mid = sizeOfStack % 2 == 0 ? sizeOfStack / 2 : sizeOfStack / 2 + 1;

while (s.size() > mid) {

tempStack.push(s.pop());

}

s.pop();

while (!tempStack.isEmpty()) {

s.push(tempStack.pop());

}

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

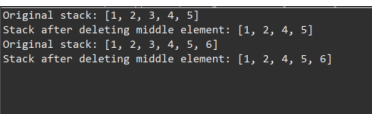
stack.push(5);

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

deleteMid(stack, stack.size());

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

}



18. Next Greater Element

public class NextGreat {

static void Greater(int[] arr) {

int n = arr.length;

int[] nge = new int[n];

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

Arrays.fill(nge, -1);

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

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

stack.pop();

}

if (!stack.isEmpty()) {

nge[i] = 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) {

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

System.out.println("Next Greater Elements for the array " + Arrays.toString(arr1) + ":");

Greater(arr1);

System.out.println();

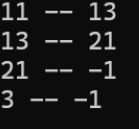
int[] arr2 = {13, 7, 6, 12};

System.out.println("Next Greater Elements for the array " + Arrays.toString(arr2) + ":");

Greater(arr2);

}

}



19. Right view of the binary tree

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

public class RightView {

public List<Integer> rightSideView(TreeNode root) {

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

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

if (root == null)

return rightView;

queue.add(root);

while (!queue.isEmpty()) {

int size = queue.size();

TreeNode rightNode = null;

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

TreeNode node = queue.poll();

rightNode = node;

if (node.left != null)

queue.add(node.left);

if (node.right != null)

queue.add(node.right);

}

rightView.add(rightNode.val);

}

return rightView;

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

RightView solution = new RightView();

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

System.out.println(rightView);

}

}



20.Maximum height or depth of the binary tree

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

public class MaxHeight {

public int maxDepth(TreeNode root) {

return findHeight(root);

}

public int findHeight(TreeNode node) {

if (node == null) {

return 0;

}

int left = findHeight(node.left);

int right = findHeight(node.right);

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

}

public static void main(String[] args) {

TreeNode root = new TreeNode(12);

root.left = new TreeNode(8);

root.right = new TreeNode(18);

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

root.left.right = new TreeNode(11);

MaxHeight solution = new MaxHeight();

int maxDepth = solution.maxDepth(root);

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

}

}

