**1. MAXIMUM SUBARRAY SUM-KADANES ALGORITHM**

import java.util.\*;

public class Main {

public static int maxSum(int[] nums) {

int currentSum = Integer.MIN\_VALUE;

int prevSum = 0;

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

prevSum += nums[i];

if (prevSum > currentSum) {

currentSum = prevSum;

}

if (prevSum < 0) {

prevSum = 0;

}

}

return currentSum;

}

public static void main(String[] args) {

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

System.out.println(maxSum(arr));

}

}

TIME COMPLEXITY : O(N)

**2.MAXIMUM PRODUCT SUBARRAY**

import java.util.\*;

public class Main {

public static int maxproduct(int[] nums) {

int MAX = nums[0];

int MIN = nums[0];

int RES = nums[0];

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

int current = nums[i];

if (current < 0) {

int temp = MAX;

MAX = MIN;

MIN = temp;

}

MAX = Math.max(current, MAX \* current);

MIN = Math.min(current, MIN \* current);

RES = Math.max(MAX, RES);

}

return RES;

}

public static void main(String[] args) {

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

System.out.println(maxproduct(arr));

}

}

TIME COMPLEXITY: O(N)

**3.INDEX OF KEY**

public class Main {

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

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

if (arr[i] == key) {

return i;

}

}

return -1;

}

public static void main(String[] args) {

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

int key = 0;

int result = searchInRotatedArray(arr, key);

System.out.println( result);

}

}

TIME COMPLEXITY: O(N)

**4. CONTAINER WITH MOST WATER**

public class Solution {

public int maxArea(int[] height) {

int left = 0, right = height.length - 1, maxArea = 0;

while (left < right) {

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

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

else right--;

}

return maxArea;

}

public static void main(String[] args) {

Solution solution = new Solution();

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

System.out.println(solution.maxArea(height));

}

}

TIME COMPLEXITY: O(N)

**5.FACTORIAL OF BIG NUMBER**

import java.math.BigInteger;

public class Main {

public static BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

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

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

}

return result;

}

public static void main(String[] args) {

int number = 100;

System.out.println(factorial(number));

}

}

TIME COMPLEXITY : 0(N)

**6. TRAPPING RAINWATER PROBLEM**

public class Solution {

public int trap(int[] height) {

int left = 0, right = height.length - 1, leftMax = 0, rightMax = 0, water = 0;

while (left < right) {

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

if (height[left] >= leftMax) leftMax = height[left];

else water += leftMax - height[left];

left++;

} else {

if (height[right] >= rightMax) rightMax = height[right];

else water += rightMax - height[right];

right--;

}

}

return water;

}

public static void main(String[] args) {

Solution solution = new Solution();

int[] height1 = {3, 0, 1, 0, 4, 0, 2};

System.out.println(solution.trap(height1));

int[] height2 = {3, 0, 2, 0, 4};

System.out.println(solution.trap(height2));

int[] height3 = {1, 2, 3, 4};

System.out.println(solution.trap(height3));

int[] height4 = {10, 9, 0, 5};

System.out.println(solution.trap(height4));

}

}

TIME COMPLEXITY: O(N)

**7. CHOCOLATE DISTRIBUTION PROBLEM**

import java.util.Arrays;

public class Solution {

public int findMinDiff(int[] arr, int m) {

if (m == 0 || arr.length == 0) return 0;

if (arr.length < m) 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) {

Solution solution = new Solution();

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

int m = 3;

System.out.println(solution.findMinDiff(arr, m));

}

}

TIME COMPLEXITY: O(N)

8. MERGE OVERLOGGING INTERVALS

import java.util.Arrays;

import java.util.ArrayList;

public class Solution {

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

if (intervals.length == 0) return new int[0][0];

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

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

int[] current = intervals[0];

result.add(current);

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

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

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

} else {

current = intervals[i];

result.add(current);

}

}

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

}

public static void main(String[] args) {

Solution solution = new Solution();

int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

int[][] result1 = solution.merge(intervals1);

for (int[] interval : result1) {

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

}

int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

int[][] result2 = solution.merge(intervals2);

for (int[] interval : result2) {

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

}

}

TIME COMPLEXITY:O(N)

9. A BOOLEAN MATRIX QUESTION

public class Solution {

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

if (matrix == null || matrix.length == 0) return;

int m = matrix.length, 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) {

Solution solution = new Solution();

int[][] matrix1 = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

solution.printSpiral(matrix1);

System.out.println();

int[][] matrix2 = {

{1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18}

};

solution.printSpiral(matrix2);

}

}

TIME COMPLEXITY: O(N)

**10. PRINT A GIVEN MATRIX IN SPRIAL FORM**

public class Solution {

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

if (matrix == null || matrix.length == 0) return;

int m = matrix.length, 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) {

Solution solution = new Solution();

int[][] matrix1 = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

solution.printSpiral(matrix1);

System.out.println();

int[][] matrix2 = {

{1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18}

};

solution.printSpiral(matrix2);

}

}

TIME COMPLEXITY: O(N)

**13. CHECK IF GIVEN PARANTHESES EXPRESSION IS BALANCES OR NOT**

public class Solution {

public String isBalanced(String str) {

int balance = 0;

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

if (str.charAt(i) == '(') {

balance++;

} else if (str.charAt(i) == ')') {

balance--;

}

if (balance < 0) {

return "Not Balanced";

}

}

return balance == 0 ? "Balanced" : "Not Balanced";

}

public static void main(String[] args) {

Solution solution = new Solution();

String str1 = "((()))()()";

System.out.println(solution.isBalanced(str1));

String str2 = "())((())";

System.out.println(solution.isBalanced(str2));

}

}

TIME COMPLEXITY:O(N)

**14. CHECK IF TWO STRINGS ARE ANAGRAMS OF EACH OTHER**

import java.util.Arrays;

public class Solution {

public 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) {

Solution solution = new Solution();

String s1 = "geeks";

String s2 = "kseeg";

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

s1 = "allergy";

s2 = "allergic";

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

s1 = "g";

s2 = "g";

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

}

}

TIME COMPLEXITY: O(N)

**15. LONGEST PALINDROMIC SUBSTRING**

public class Solution {

public String longestPalindrome(String str) {

if (str == null || str.length() < 1) return "";

int start = 0, end = 0;

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

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

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

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

if (len > end - start) {

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

end = i + len / 2;

}

}

return str.substring(start, end + 1);

}

private int expandAroundCenter(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) {

Solution solution = new Solution();

String str1 = "forgeeksskeegfor";

System.out.println(solution.longestPalindrome(str1));

String str2 = "Geeks";

System.out.println(solution.longestPalindrome(str2));

String str3 = "abc";

System.out.println(solution.longestPalindrome(str3));

String str4 = "";

System.out.println(solution.longestPalindrome(str4));

}

}

TIME COMPLEXITY:O(N)

**16 LARGEST COMMOM PREFIX**

import java.util.Arrays;

public class Main {

public static String longestCommonPrefix(String[] arr) {

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

return "-1";

}

String first = arr[0];

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

int i = 0;

while (i < first.length() && i < last.length()) {

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

break;

}

i++;

}

if (i == 0) {

return "-1";

}

return first.substring(0, i);

}

public static void main(String[] args) {

String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

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

}

}

TIME COMPLEXITY: O(N)

**17. REMOVE MID ELEMENT OF STACK**

import java.util.Stack;

public class Main {

public static void removeMiddle(Stack<Integer> stack) {

int size = stack.size();

int mid = size/2;

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

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

int temp = stack.pop();

if(i != mid){

tempStack.push(temp);

}

}

while(!tempStack.isEmpty()) {

stack.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);

removeMiddle(stack);

System.out.println(stack);

}

}

TIME COMPLEXITY:O(N)

**18. NEXT GREATER ELEMENT (NGE) FOR EVERY ELEMENT IN GIVEN ARRAY**

import java.util.Stack;

public class Solution {

public void nextGreaterElement(int[] arr) {

int n = arr.length;

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

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

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

int index = stack.pop();

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

}

stack.push(i);

}

while (!stack.isEmpty()) {

int index = stack.pop();

System.out.println(arr[index] + " --> " + -1);

}

}

public static void main(String[] args) {

Solution solution = new Solution();

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

solution.nextGreaterElement(arr1);

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

solution.nextGreaterElement(arr2);

}

}

TIME COMPLEXITY: O(N)

**19. PRINT RIGHT VIEW OF A BINARY TREE**

import java.util.\*;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class Solution {

public void rightView(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 temp = queue.poll();

if (i == n) {

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

}

if (temp.left != null) {

queue.add(temp.left);

}

if (temp.right != null) {

queue.add(temp.right);

}

}

}

}

public static void main(String[] args) {

Solution solution = new Solution();

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

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

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

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

root.left.right.left = new Node(7);

solution.rightView(root);

}

}

TIME COMPLEXITY: O(N)

**20. MAXINUM DEPTH OR HEIGHT OF BINARY TREE**

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class Solution {

public 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) {

Solution solution = new Solution();

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

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

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

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

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

}

}

TIME COMPLEXITY: O(N)