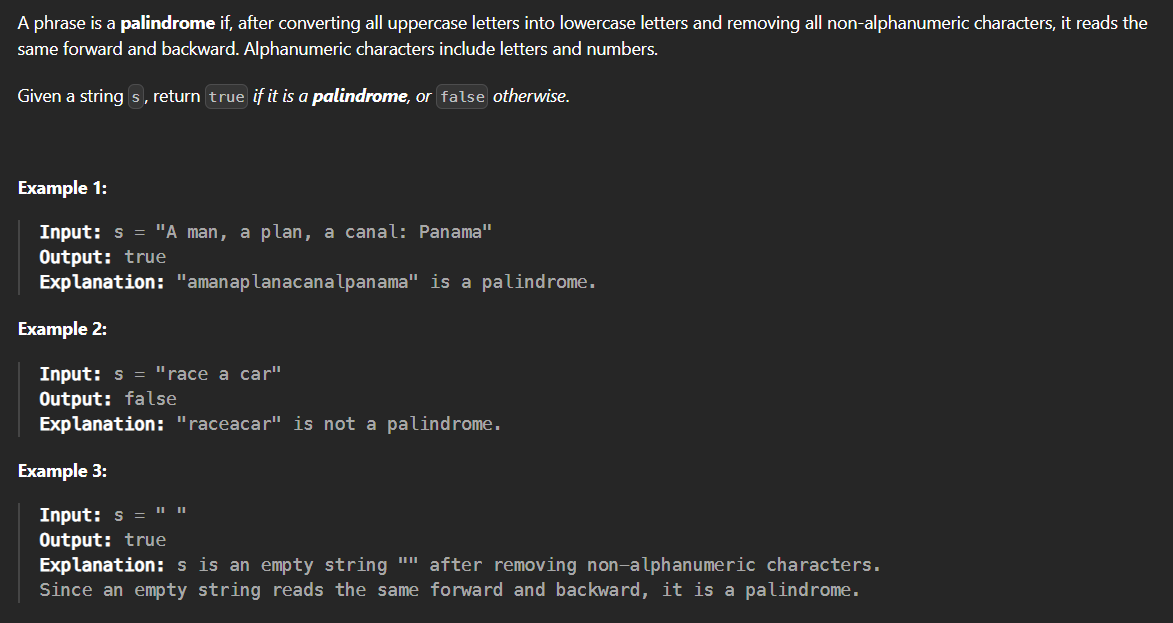
**DSA Practice Test – 9** 21st Nov 2024

**1. Valid Palindrome**



**Code:**

class ValidPalindrome {

static boolean isPalindrome(String s) {

StringBuilder filtered = new StringBuilder();

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

if (Character.isLetterOrDigit(c)) {

filtered.append(Character.toLowerCase(c));

}

}

int l = 0, r = filtered.length() - 1;

while (l < r) {

if (filtered.charAt(l) != filtered.charAt(r)) {

return false;

}

l++;

r--;

}

return true;

}

public static void main(String ar[]){

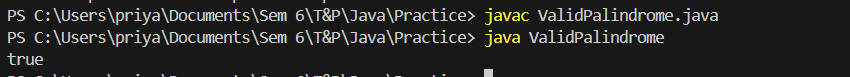
String s="malayalam";

System.out.println(isPalindrome(s));

}

}

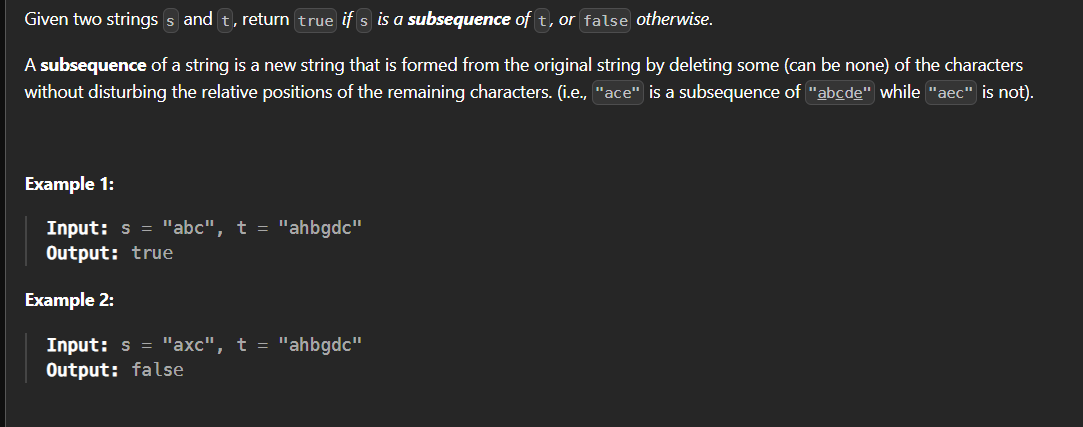
**Output**:



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**2. Is subsequence**

****

**Code:**

class IsSubsequence {

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

int sp=0,tp=0;

while(sp<s.length() && tp<t.length()){

if(s.charAt(sp)==t.charAt(tp)){

sp++;

}

tp++;

}

return sp==s.length();

}

public static void main(String main[]){

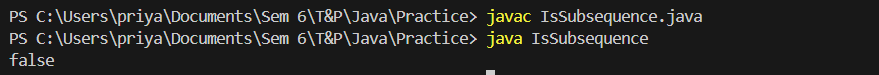
String s="axc",t="ahbgdc";

System.out.print(isSubsequence(s, t));

}

}

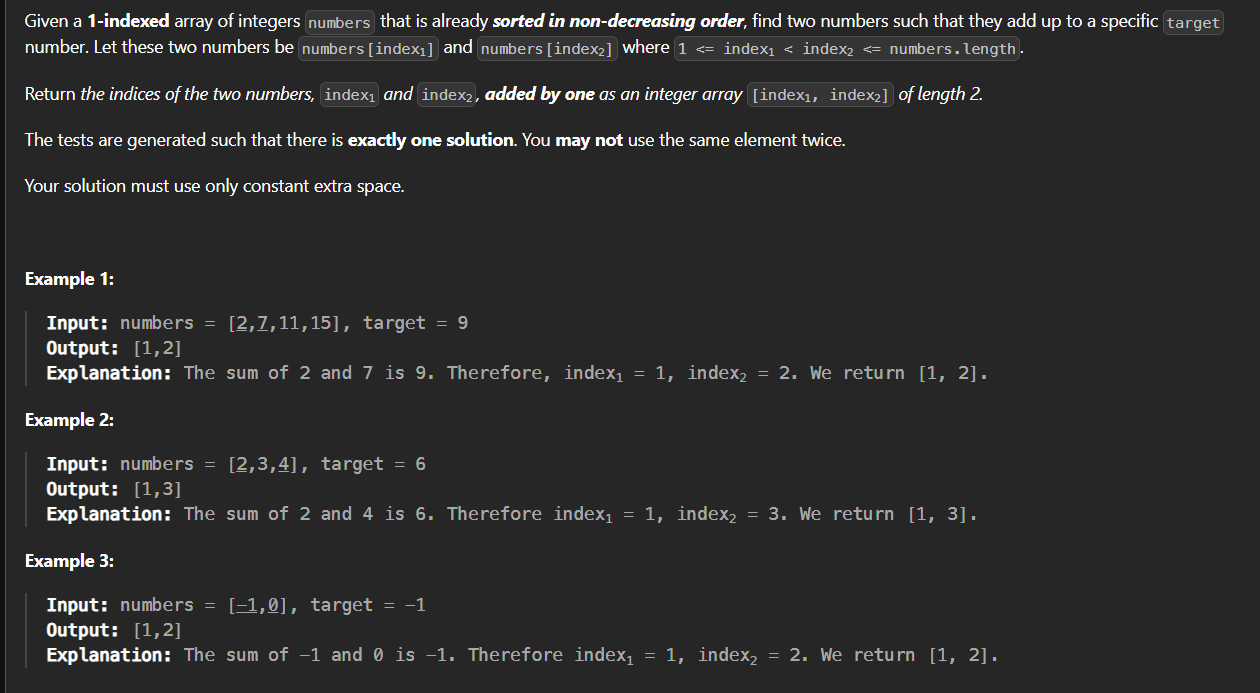
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**3. Two Sum II – Input Array is Sorted**



**Code:**

class TwoSum {

public static int[] twoSum(int[] n, int t) {

int l=0,r=n.length-1;

int[] res=new int[2];

while(l<r){

if((n[l]+n[r])==t){

res[0]=l+1;

res[1]=r+1;

break;

}

else if((n[l]+n[r])<t){

l++;

}

else r--;

}

return res;

}

public static void main(String ar[]){

int[] n={2,7,11,15};

int t=9;

int[] res=twoSum(n, t);

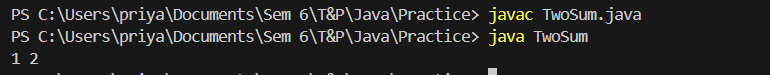
for(int i : res)

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

}

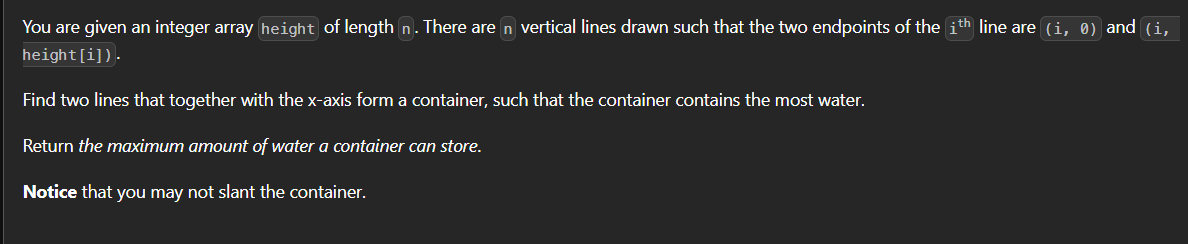
}

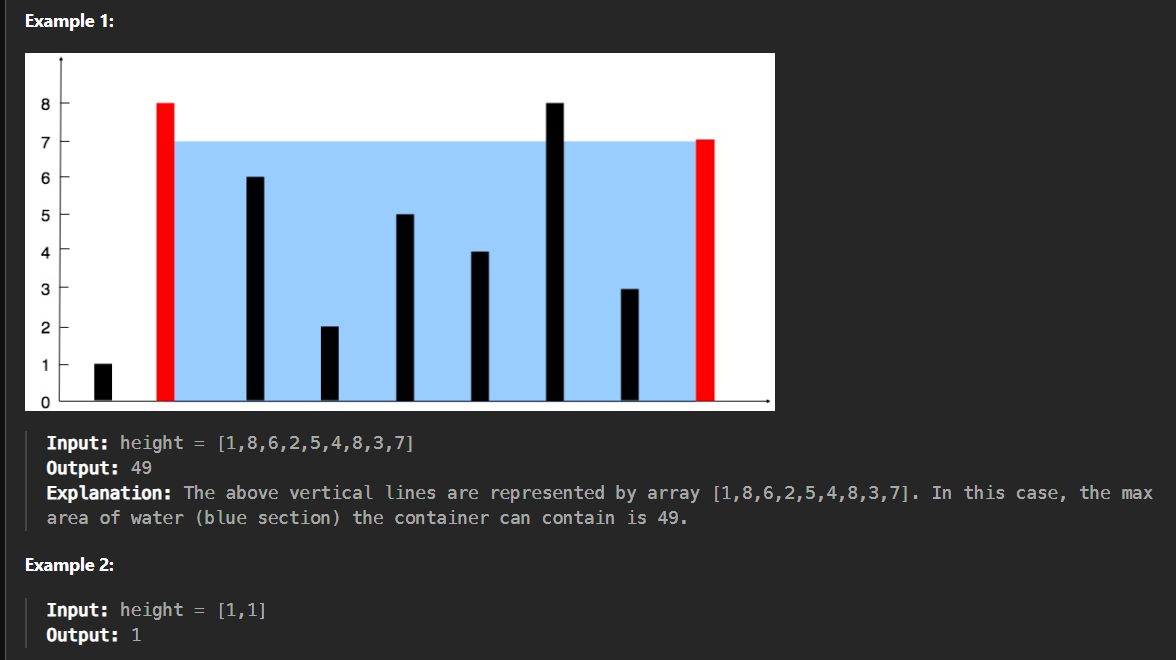
**Output:**



**Time Complexity:** O (n)  
**Space Complexity:** O (1)

**4. Container with Most Water**





**Code:**

public class ContainerWithMostWater {

public static int maxArea(int[] height) {

int a=0;

int b=height.length-1;

int area=0;

while(a<b){

int curr\_a=Math.min(height[a],height[b])\*(b-a);

area=Math.max(area,curr\_a);

if(height[a]<height[b]){

a++;

}

else{

b--;

}

}

return area;

}

public static void main(String ar[]){

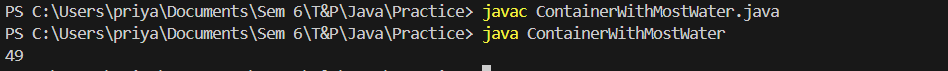
int[] h={1,8,6,2,5,4,8,3,7};

System.out.print(maxArea(h));

}

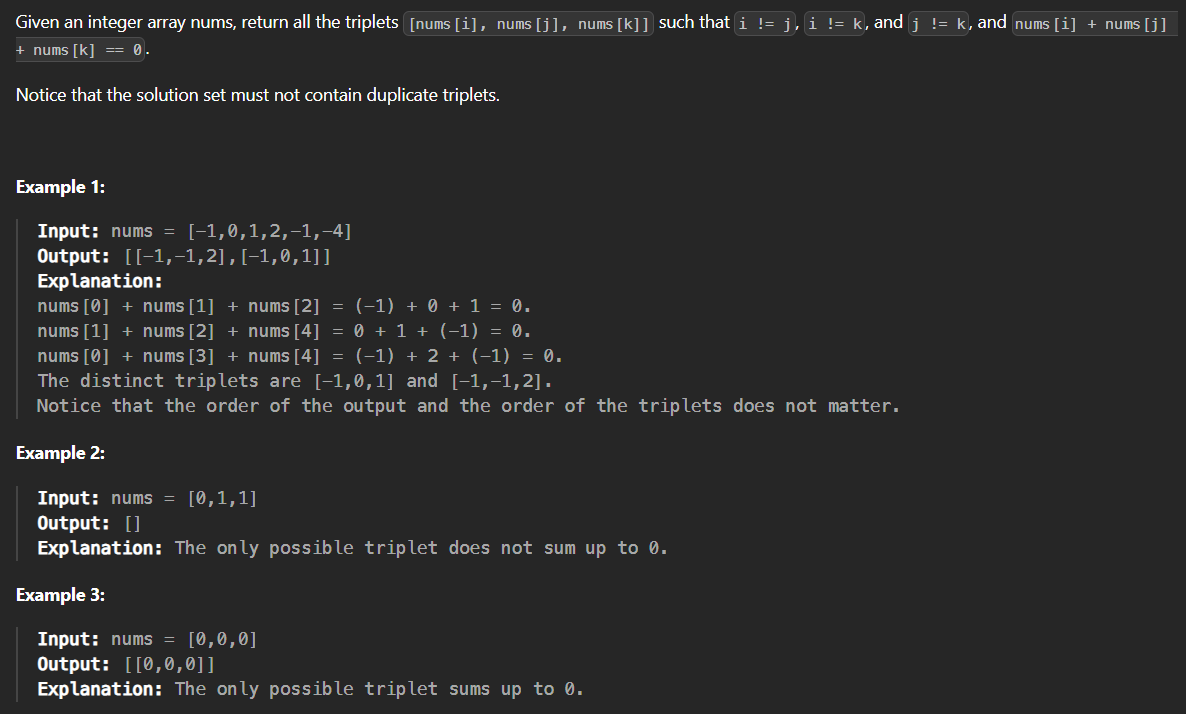
}

**Output:**



**Time Complexity:** O (n)  
**Space Complexity:** O (1)

**5. 3 Sum**



**Code:**import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class Sum3 {

public static List<List<Integer>> threeSum(int[] n) {

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

Arrays.sort(n);

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

if(i>0 && n[i]==n[i-1]) continue;

int j=i+1,k=n.length-1;

while(j<k){

int t=n[i]+n[j]+n[k];

if(t>0) k--;

else if(t<0) j++;

else{

res.add(Arrays.asList(n[i],n[j],n[k]));

j++;

while(n[j]==n[j-1] && j<k) j++;

}

}

}

return res;

}

public static void main(String ar[]){

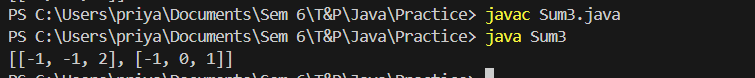
int[] n={-1,0,1,2,-1,-4};

System.out.print(threeSum(n));

}

}

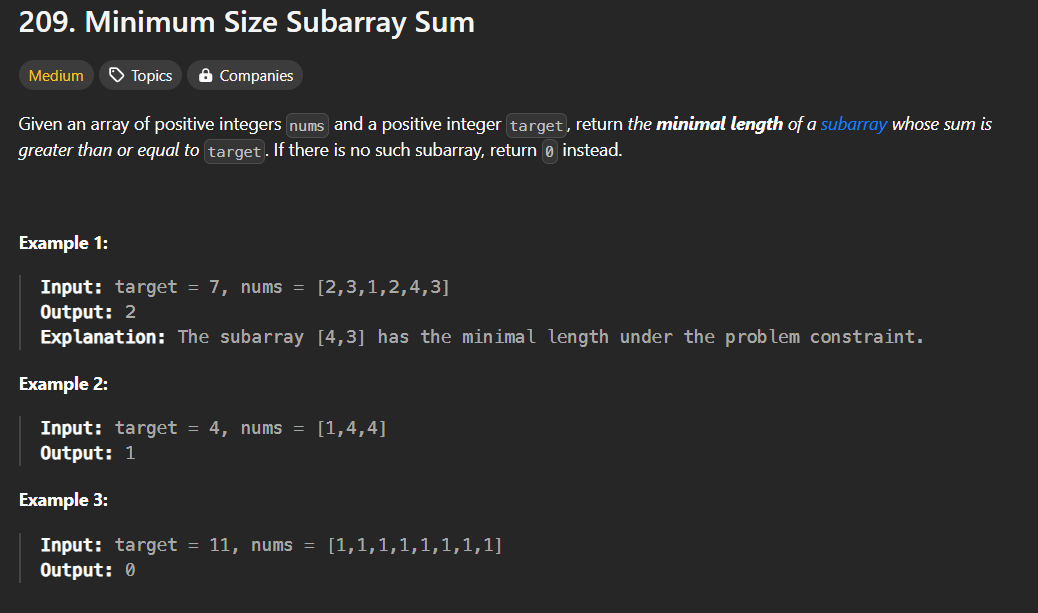
**Output:**

****

**Time Complexity:** O (n2)

**Space Complexity:** O (1)

**6. Minimum Size Subarray Sum**



**Code:**

class MinimumSubArray {

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

int n = nums.length;

int l = 0;

int sum = 0;

int m = Integer.MAX\_VALUE;

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

sum += nums[r];

while (sum >= target) {

m = Math.min(m, r - l + 1);

sum -= nums[l];

l++;

}

}

return m == Integer.MAX\_VALUE ? 0 : m;

}

public static void main(String ar[]){

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

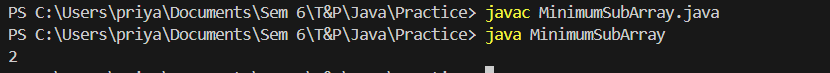
int t=7;

System.out.print(minSubArrayLen(t, g));

}

}

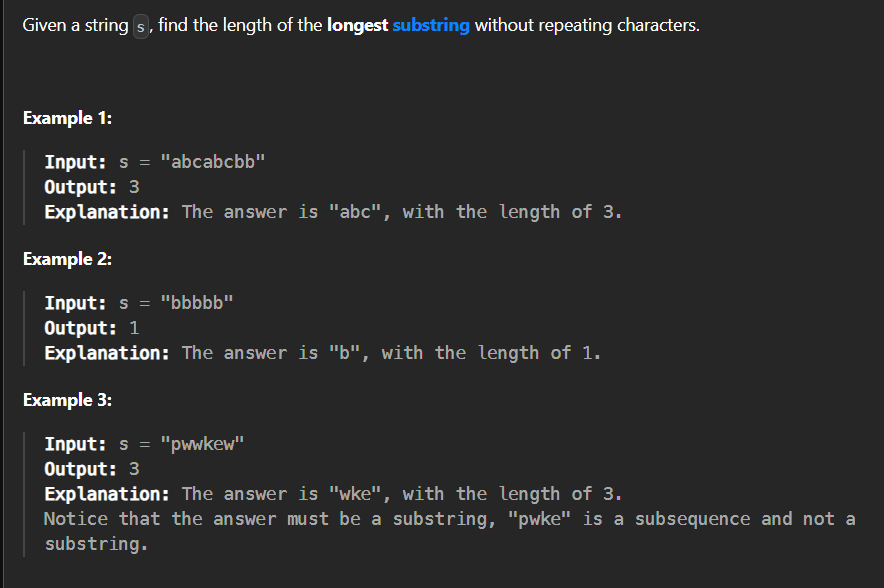
**Output:**

****

**Time Complexity:** O (n)

**Space Complexity:** O (1)

**7. Longest Substring Without Repeating Character**



**Code:**

import java.util.HashSet;

import java.util.Set;

class LongestSubstring {

public static int lengthOfLongestSubstring(String s) {

Set<Character> set=new HashSet<>();

int m=0,l=0;

for(int r=0;r<s.length();r++){

if(!set.contains(s.charAt(r))){

set.add(s.charAt(r));

m=Math.max(m,r-l+1);

}

else{

while(s.charAt(l)!=s.charAt(r)){

set.remove(s.charAt(l));

l++;

}

set.remove(s.charAt(l));

l++;

set.add(s.charAt(r));

}

}

return m;

}

public static void main (String ar[]){

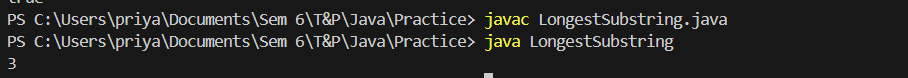
String s="gwwkew";

System.out.print(lengthOfLongestSubstring(s));

}

}

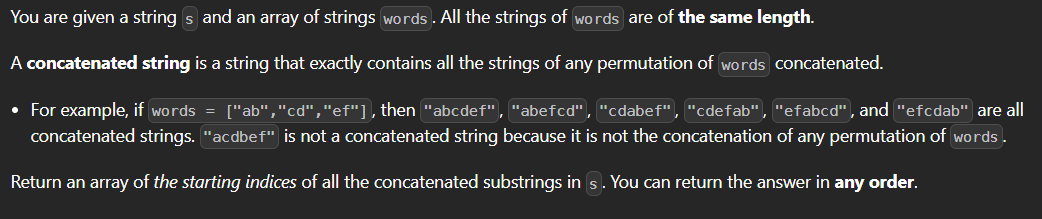
**Output:**

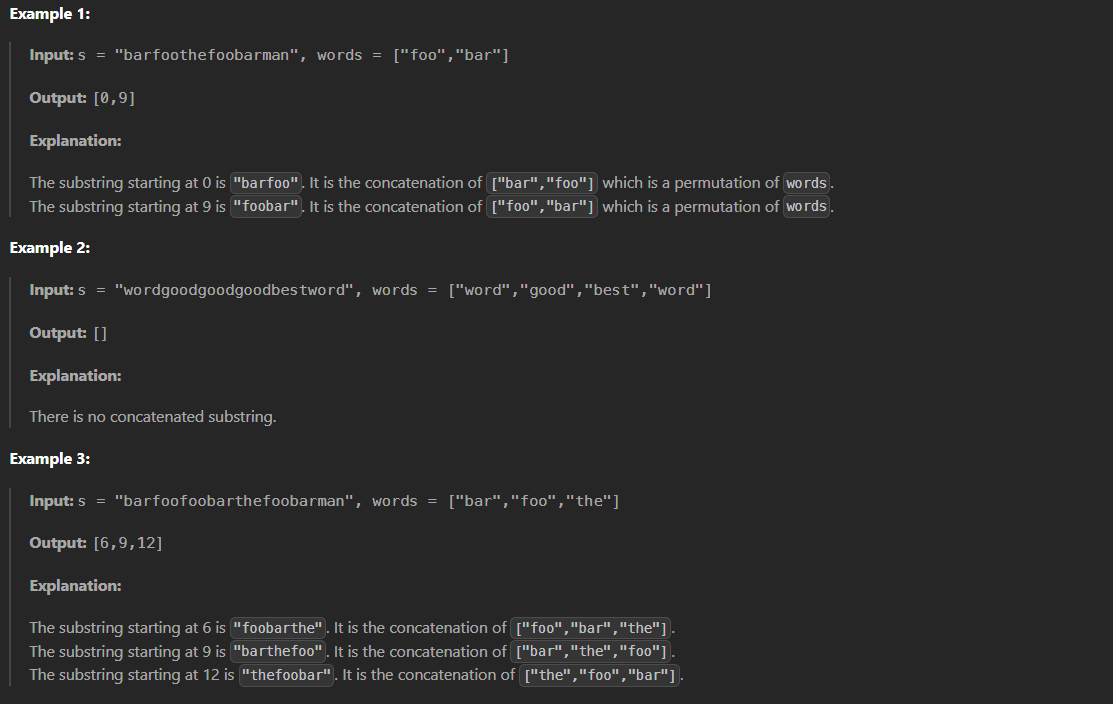


**Time Complexity:** O (n)

**Space Complexity:** O (1)

**8. Substring With Concatenation of All Words**





**Code:**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

class SubstringCon {

public static List<Integer> findSubstring(String s, String[] words) {

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

int n = s.length();

int m = words.length;

int w = words[0].length();

HashMap<String,Integer> map = new HashMap<>();

for(String x : words)

map.put(x, map.getOrDefault(x,0)+1);

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

HashMap<String,Integer> t = new HashMap<>();

int c = 0;

for(int j=i,k=i; j+w <= n; j=j+w){

String word = s.substring(j,j+w);

t.put(word,t.getOrDefault(word,0)+1);

c++;

if(c==m){

if(map.equals(t)){

ans.add(k);

}

String remove = s.substring(k,k+w);

t.computeIfPresent(remove, (a, b) -> (b > 1) ? b - 1 : null);

c--;

k=k+w;

}

}

}

return ans;

}

public static void main(String ar[]){

String[] str={"foo","bar"};

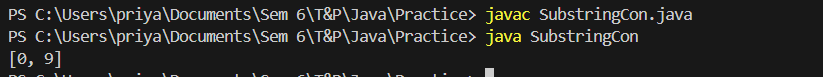
String s="barfoothefoobarman";

System.out.print(findSubstring(s, str));

}

}

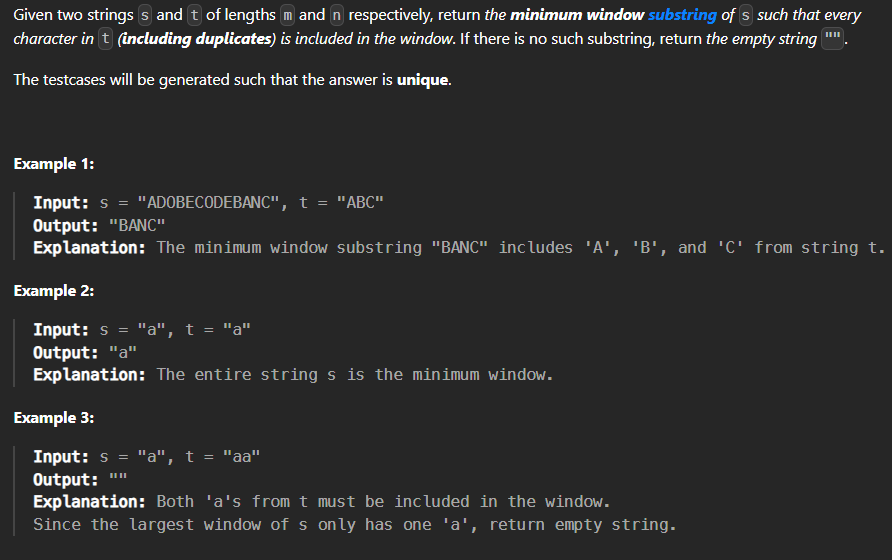
**Output:**



**Time Complexity:** O (n2)

**Space Complexity:** O (n)

**9. Minimum Window Substring**



**Code:**

class MinWinSub {

public static String minWindow(String s, String t) {

if (s == null || t == null || s.length() == 0 || t.length() == 0 || s.length() < t.length()) {

return "";

}

int[] targetCount = new int[128];

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

targetCount[c]++;

}

int left = 0, right = 0, start = 0, minLength = Integer.MAX\_VALUE;

int requiredCount = t.length();

char[] chars = s.toCharArray();

while (right < chars.length) {

if (targetCount[chars[right]] > 0) {

requiredCount--;

}

targetCount[chars[right]]--;

right++;

while (requiredCount == 0) {

if (right - left < minLength) {

minLength = right - left;

start = left;

}

targetCount[chars[left]]++;

if (targetCount[chars[left]] > 0) {

requiredCount++;

}

left++;

}

}

return minLength == Integer.MAX\_VALUE ? "" : s.substring(start, start + minLength);

}

public static void main(String[] args) {

String s = "ADOBECODEBANC";

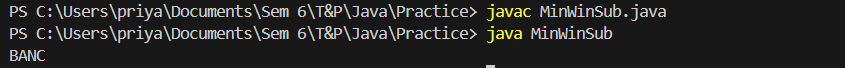
String t = "ABC";

System.out.println(minWindow(s, t));

}

}

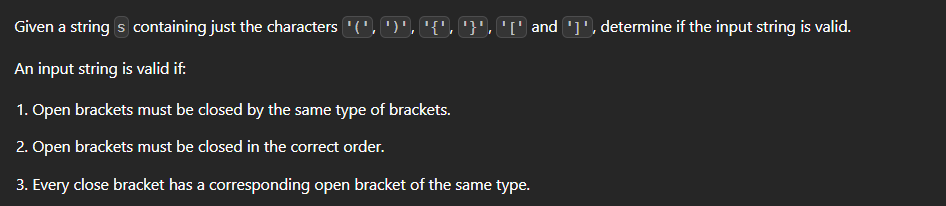
**Output:**

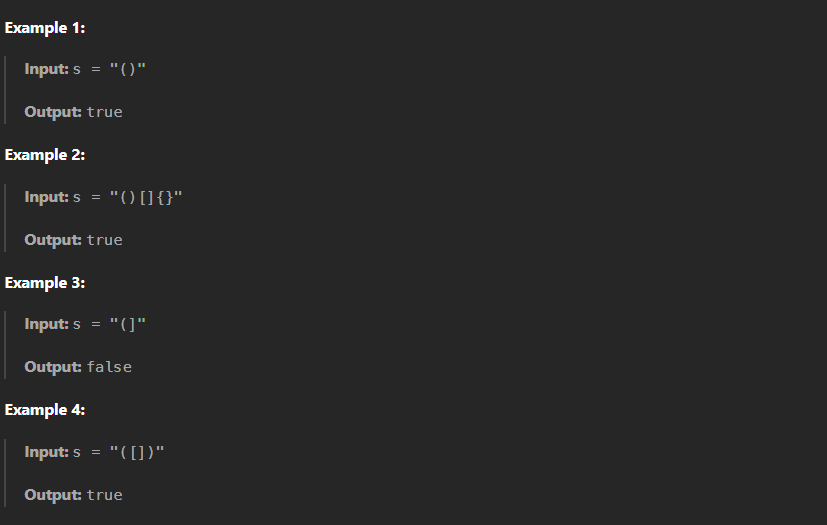
****

**Time Complexity:** O(n2)

**Space Complexity:** O (1)

**10. Valid Parentheses**





**Code:**

import java.util.Stack;

import java.util.Map;

import java.util.HashMap;

class ValidParentheses {

public static boolean isValid(String s) {

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

Map<Character, Character> m = new HashMap<>();

m.put(')', '(');

m.put(']', '[');

m.put('}', '{');

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

if (m.containsValue(c)) {

stack.push(c);

} else if (m.containsKey(c)) {

if (!stack.isEmpty() && stack.peek() == m.get(c)) {

stack.pop();

} else {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String priya[]){

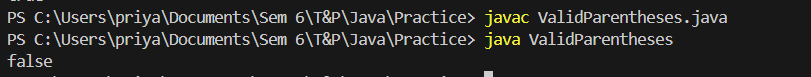
String s="(]";

System.out.print(isValid(s));

}

}

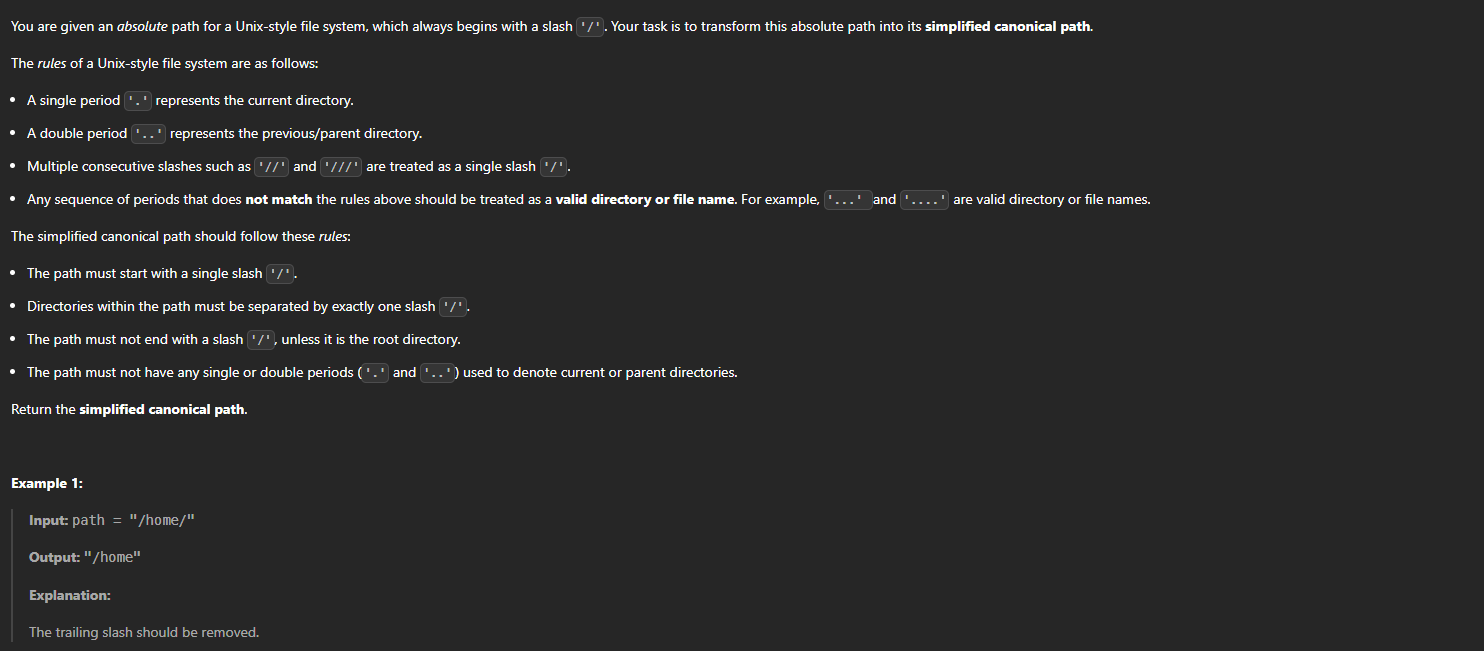
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**11. Simplify Path**





**Code:**

import java.util.Stack;

class SimplifyPath {

public static String simplifyPath(String path) {

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

String[] components = path.split("/");

for (String component : components) {

if (component.equals("") || component.equals(".")) {

continue;

} else if (component.equals("..")) {

if (!stack.isEmpty()) {

stack.pop();

}

} else {

stack.push(component);

}

}

StringBuilder canonicalPath = new StringBuilder();

for (String dir : stack) {

canonicalPath.append("/").append(dir);

}

return canonicalPath.length() > 0 ? canonicalPath.toString() : "/";

}

public static void main(String[] args) {

System.out.println(simplifyPath("/home/"));

System.out.println(simplifyPath("/home//foo/"));

System.out.println(simplifyPath("/home/user/Documents/../Pictures"));

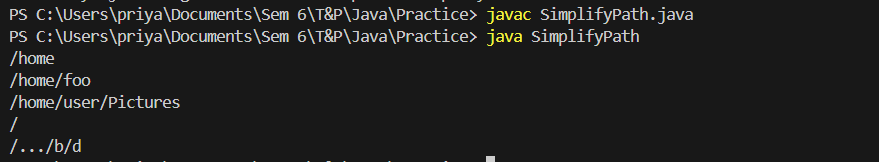
System.out.println(simplifyPath("/../"));

System.out.println(simplifyPath("/.../a/../b/c/../d/./"));

}

}

**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**12. Min Stack**



**Code:**

import java.util.Stack;

class MinStack {

private Stack<Integer> stack;

private Stack<Integer> minStack;

public MinStack() {

stack = new Stack<>();

minStack = new Stack<>();

}

public void push(int val) {

stack.push(val);

if (minStack.isEmpty() || val <= minStack.peek()) {

minStack.push(val);

}

}

public void pop() {

if (stack.peek().equals(minStack.peek())) {

minStack.pop();

}

stack.pop();

}

public int top() {

return stack.peek();

}

public int getMin() {

return minStack.peek();

}

public static void main(String[] args) {

MinStack obj = new MinStack();

obj.push(-2);

obj.push(0);

obj.push(-3);

System.out.println(obj.getMin());

obj.pop();

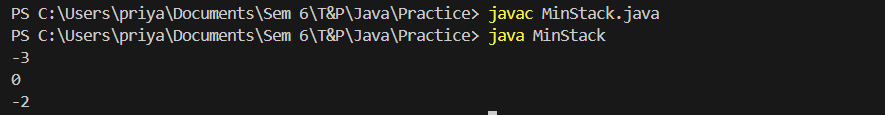
System.out.println(obj.top());

System.out.println(obj.getMin());

}

}

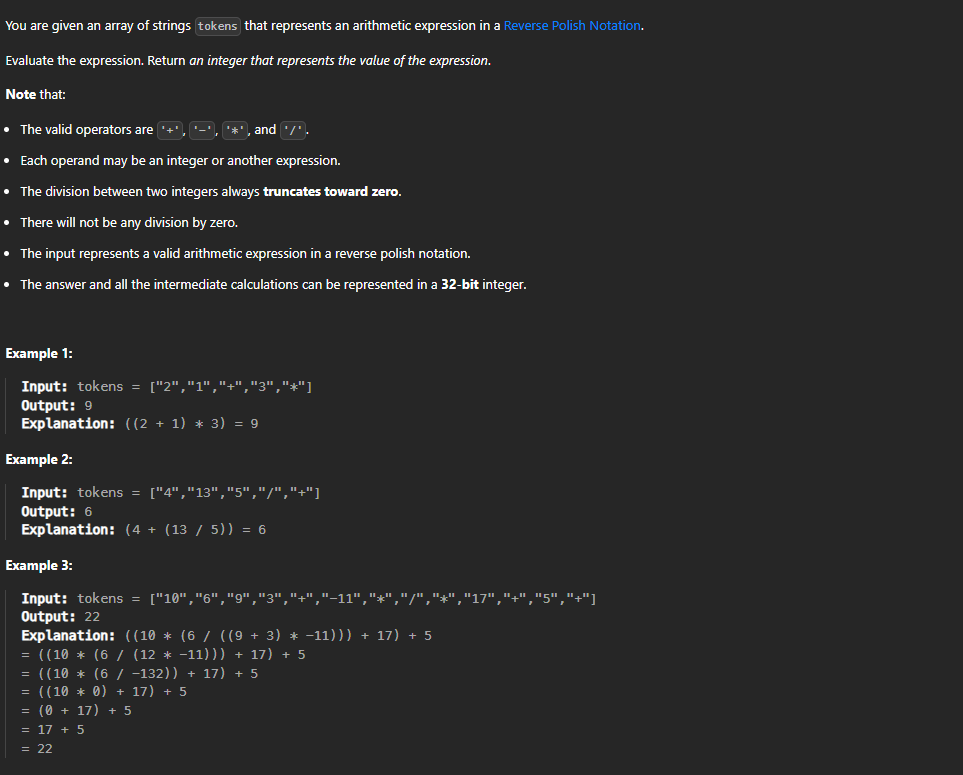
**Output:**



**Time Complexity:** O (1)

**Space Complexity:** O (n)

**13. Evaluate Reverse Polish Notation**



**Code:**

import java.util.Stack;

class ReversePolish {

public static int evalRPN(String[] tokens) {

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

for (String token : tokens) {

if (isOperator(token)) {

int b = stack.pop();

int a = stack.pop();

stack.push(applyOperator(a, b, token));

} else {

stack.push(Integer.parseInt(token));

}

}

return stack.pop();

}

private static boolean isOperator(String token) {

return "+".equals(token) || "-".equals(token) || "\*".equals(token) || "/".equals(token);

}

private static int applyOperator(int a, int b, String operator) {

switch (operator) {

case "+": return a + b;

case "-": return a - b;

case "\*": return a \* b;

case "/": return a / b;

default: throw new IllegalArgumentException("Invalid operator: " + operator);

}

}

public static void main(String[] args) {

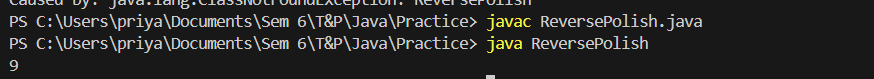
String[] tokens1 = {"2", "1", "+", "3", "\*"};

System.out.println(evalRPN(tokens1));

}

}

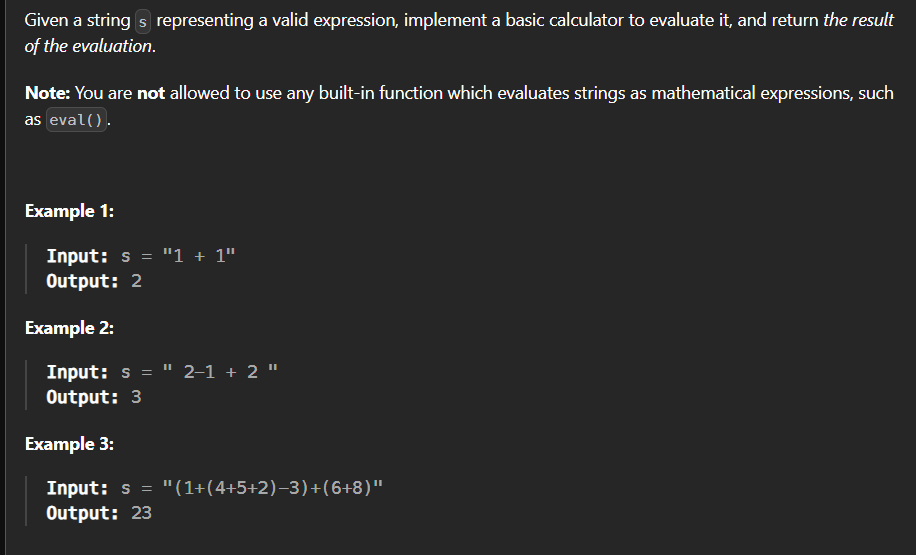
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**14. Basic Calculator**



**Code:**

import java.util.Stack;

class Calculator {

public int calculate(String s) {

int result = 0;

int sign = 1;

int number = 0;

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

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

char ch = s.charAt(i);

if (ch == ' ') {

continue;

}

if (Character.isDigit(ch)) {

number = number \* 10 + (ch - '0');

}

else if (ch == '+') {

result += sign \* number;

number = 0;

sign = 1;

}

else if (ch == '-') {

result += sign \* number;

number = 0;

sign = -1;

}

else if (ch == '(') {

stack.push(result);

stack.push(sign);

result = 0;

sign = 1;

}

else if (ch == ')') {

result += sign \* number;

number = 0;

result \*= stack.pop();

result += stack.pop();

}

}

result += sign \* number;

return result;

}

public static void main(String[] args) {

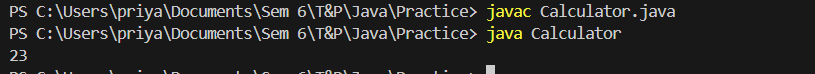
Calculator solution = new Calculator();

System.out.println(solution.calculate("(1+(4+5+2)-3)+(6+8)"));

}

}

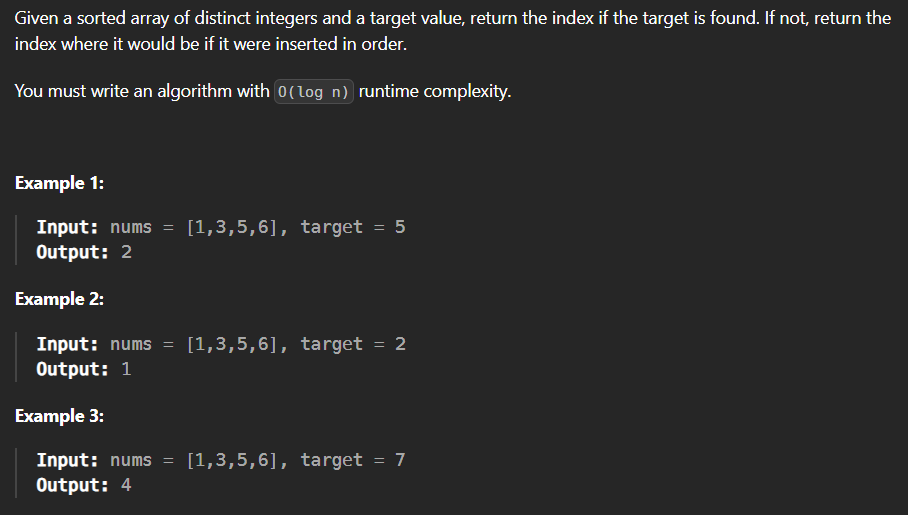
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**15. Search Insert Position**



**Code:**

class SearchInsertPos {

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

int index=0;

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

{

if(nums[i]==target)

index=i;

}

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

if(nums[i]<target && nums[i+1]>target)

index=i+1;

}

if(nums[nums.length-1]<target)

index=nums.length;

return index;

}

public static void main(String sr[]){

int[] num={1,3,5,6};

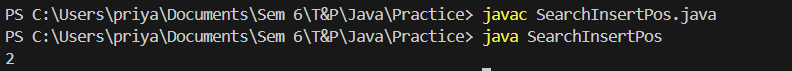
int t=5;

System.out.print(searchInsert(num, t));

}

}

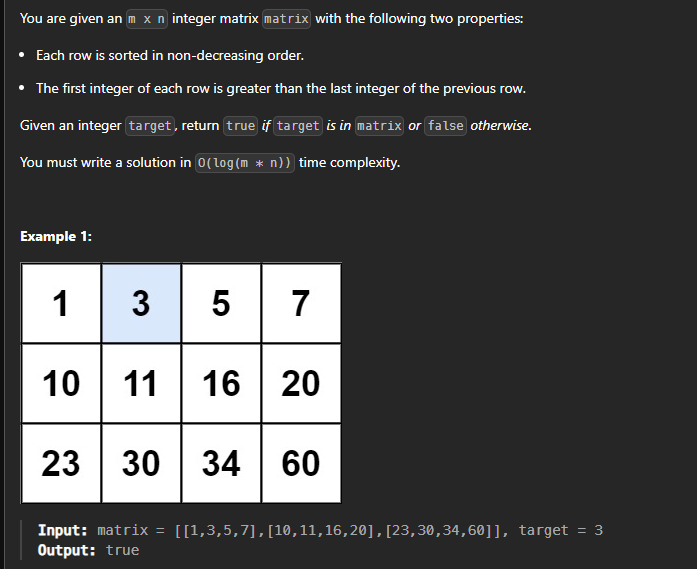
**Output:**

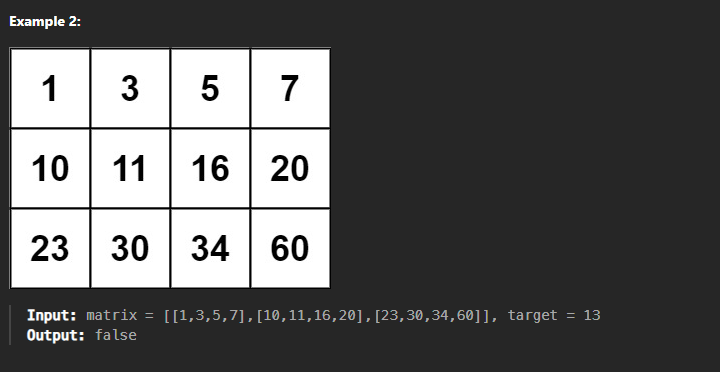


**Time Complexity:** O(n)

**Space Complexity:** O(1)

**16. Search a 2D Matrix**





**Code:**

class Search2DArr {

public static boolean searchMatrix(int[][] m, int t) {

int x=m.length;

int y=m[0].length;

int i=0,j=y-1;

while(i<x && j>=0){

if(m[i][j]==t) return true;

if(m[i][j]>t) j--;

else i++;

}

return false;

}

public static void main(String ar[]){

int[][] m={

{1,3,5,7},

{10,11,16,20},

{23,30,34,60}

};

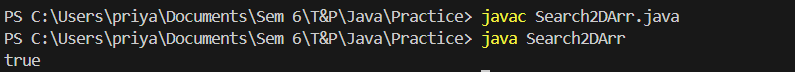
int t=3;

System.out.print(searchMatrix(m, t));

}

}

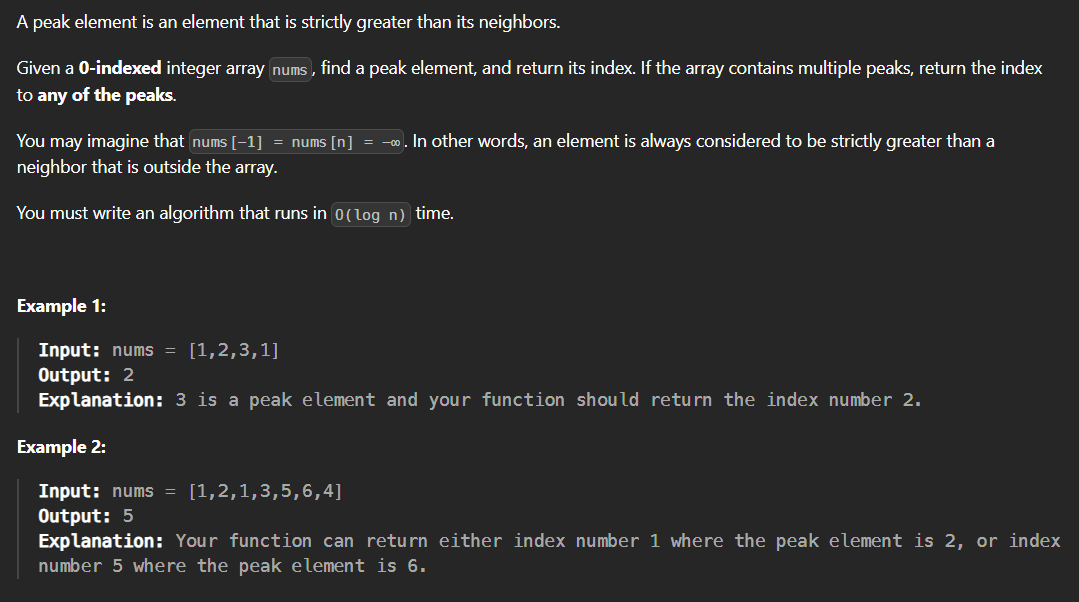
**Output:**



**Time Complexity: O (n)**

**Space Complexity: O (n)**

**17. Find Peak Element**



**Code:**

class FindPeak {

public int findPeakElement(int[] nums) {

int l = 0, r = nums.length - 1;

while (l < r) {

int m = (l + r) / 2;

if (nums[m] < nums[m + 1]) {

l = m + 1;

} else {

r = m;

}

}

return l;

}

public static void main(String[] args) {

FindPeak s = new FindPeak();

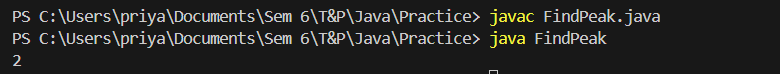
int[] nums = {1, 2, 3, 1};

System.out.println(s.findPeakElement(nums));

}

}

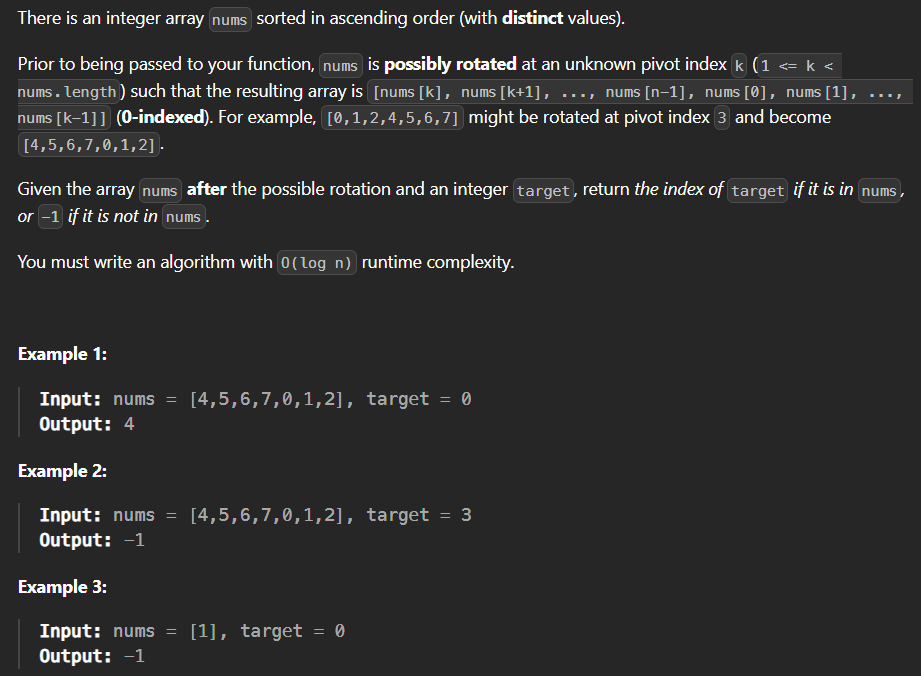
**Output:**



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

**Space Complexity:** O(1)

**18. Search in Rotated Sorted Array**



**Code:**

class SearchRotatedArr {

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

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

if(target==nums[i])

return i;

}

return -1;

}

public static void main(String main[]){

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

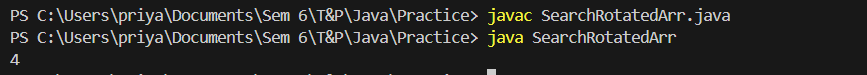
int t=0;

System.out.print(search(n, t));

}

}

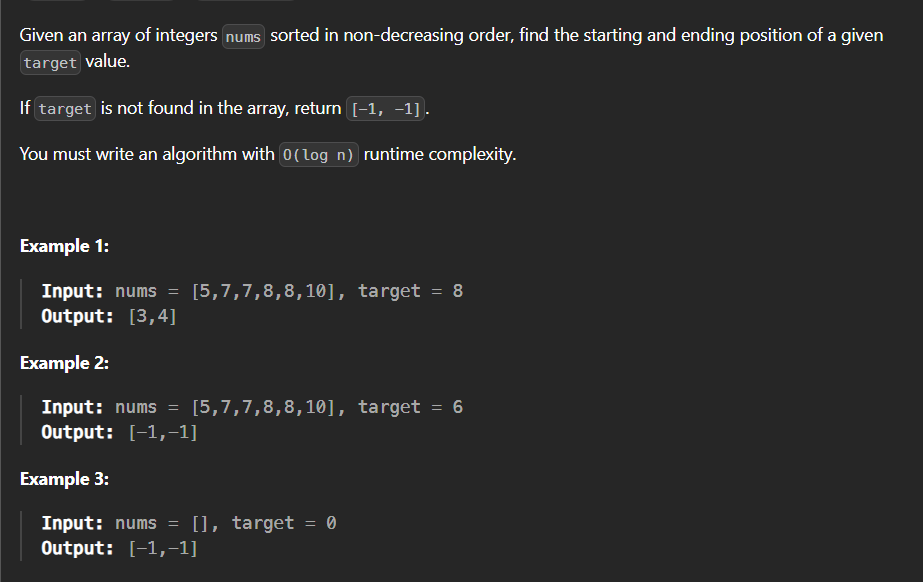
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**19. Find First and Last Position of an Element**



**Code:**

import java.util.\*;

class FindFirstLast {

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

int l = 0, r = nums.length - 1;

int first = -1;

while (l <= r) {

int mid = l + (r - l) / 2;

if (nums[mid] == target) {

first = mid;

r = mid - 1;

} else if (nums[mid] < target) {

l = mid + 1;

} else {

r = mid - 1;

}

}

return first;

}

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

int l = 0, r = nums.length - 1;

int last = -1;

while (l <= r) {

int mid = l + (r - l) / 2;

if (nums[mid] == target) {

last = mid;

l = mid + 1;

} else if (nums[mid] < target) {

l = mid + 1;

} else {

r = mid - 1;

}

}

return last;

}

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

int first = findFirst(nums, target);

int last = findLast(nums, target);

return new int[] {first, last};

}

public static void main(String[] args) {

FindFirstLast solution = new FindFirstLast();

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

int target = 8;

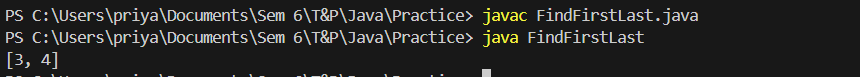
int[] result = solution.searchRange(nums, target);

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

}

}

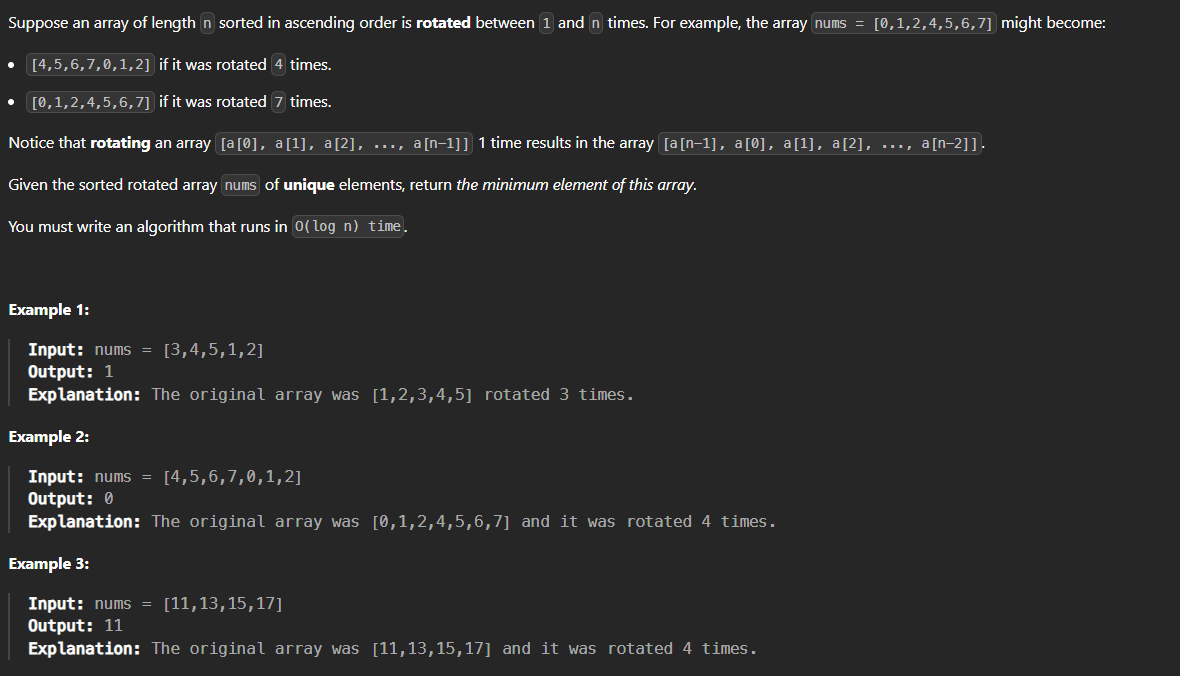
**Output:**



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

**Space Complexity:** O (log n)

**20. Find Minimum in Rotated Array**



**Code:**

class Solution {

    public int findMin(int[] nums) {

        int s=0;

        int e=nums.length-1;

        if(nums[e] > nums[s]){

            return nums[s];

        }

        return minElement(nums, s, e);

    }

    public static int minElement(int[] nums, int s, int e) {

        while (s < e) {

            int mid = s + (e - s) / 2;

            if (nums[mid] > nums[mid + 1]) {

                return nums[mid+1];

            }

            if (nums[mid] >= nums[s]) {

                s = mid + 1;

            } else {

                e = mid;

            }

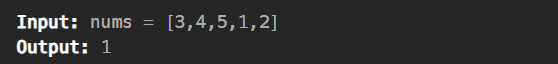
        }

        return nums[s];

    }

}

**Output:**



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

**Space Complexity:** O (1)