**Date:** 21/11/24

**DSA Practice Problems**

1. **Valid Palindrome**

class Solution {

public boolean isPalindrome(String s) {

StringBuilder str=new StringBuilder();

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

if(Character.isLetterOrDigit(c)){

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

}

}

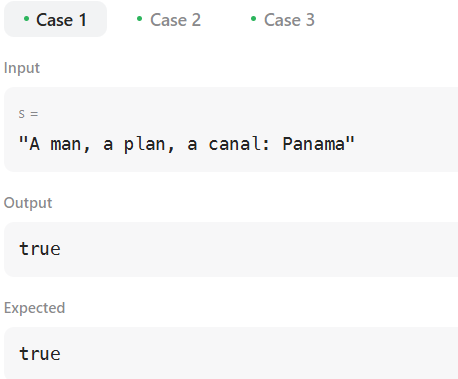
String front=str.toString();

String end=str.reverse().toString();

return front.equals(end);

}

}



**Time Complexity:** O(n)

1. **Is Subsequence**

class Solution {

    public boolean isSubsequence(String s, String t) {

        int scount=0;

        int tcount=0;

        while(scount<s.length() && tcount<t.length()){

            if(s.charAt(scount)==t.charAt(tcount)){

                scount++;

            }

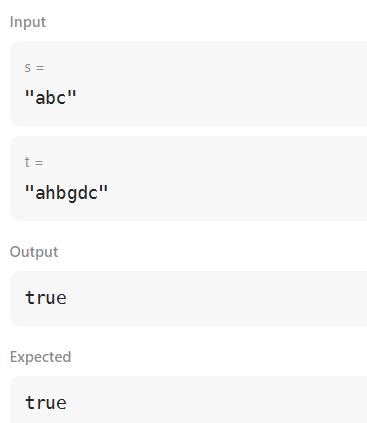
            tcount++;

        }

        return scount==s.length();

    }

}



**Time Complexity:** O(n)

1. [**Two Sum II - Input Array Is Sorted**](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/)

class Solution {

  public int[] twoSum(int[] numbers, int target) {

   int left=0;

    int right=numbers.length-1;

    while(numbers[left]+numbers[right]!=target){

        if(numbers[left]+numbers[right]<target){

            ++left;

        }

        else{

            --right;

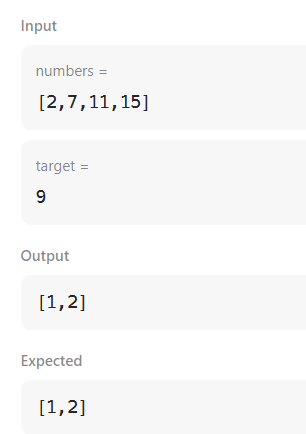
        }

    }

    return new int[] {left+1,right+1};

  }

}



**Time Complexity:** O(n)

1. **Container with most water**

class Solution {

public int maxArea(int[] height) {

int left=0;

int right=height.length-1;

int maxarea=0;

while(left<right){

int base=right-left;

int length=Math.min(height[left],height[right]);

int area=base\*length;

maxarea=Math.max(area,maxarea);

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

left++;

}

else{

right--;

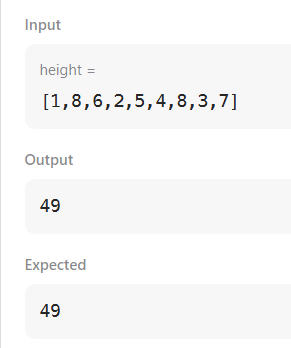
}

}

return maxarea;

}

}



**Time Complexity:** O(n)

1. **3Sum**

class Solution {

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

int target = 0;

Arrays.sort(nums);

Set<List<Integer>> set1 = new HashSet<>();

List<List<Integer>> list1= new ArrayList<>();

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

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (sum == target) {

set1.add(Arrays.asList(nums[i], nums[j], nums[k]));

j++;

k--;

} else if (sum < target) {

j++;

} else {

k--;

}

}

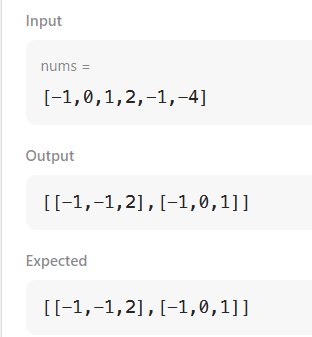
}

list1.addAll(set1);

return list1;

}

}



**Time Complexity:** O(n^2)

1. **Valid Parantheses**

class Solution {

public boolean isValid(String s) {

Stack<Character> stack1 = new Stack<>();

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

if (s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[') {

stack1.push(s.charAt(i));

} else {

if (!stack1.empty() &&

((stack1.peek() == '(' && s.charAt(i) == ')') ||

(stack1.peek() == '{' && s.charAt(i) == '}') ||

(stack1.peek() == '[' && s.charAt(i) == ']'))) {

stack1.pop();

}

}

}

if (stack1.empty()) {

return true;

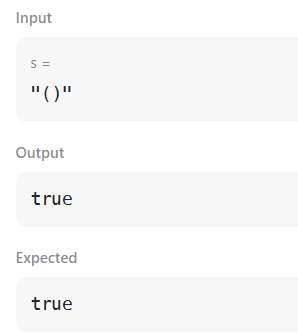
} else {

return false;

}

}

}



**Time Complexity:** O(n)

1. **Simplify Path**

class Solution {

public String simplifyPath(String path) {

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

Stack<String> stack1=new Stack<>();

for(final String str:newstr){

if(str.isEmpty() || str.equals(".")){

continue;

}

if(str.equals("..")){

if(!stack1.isEmpty()){

stack1.pop();

}

}

else{

stack1.push(str);

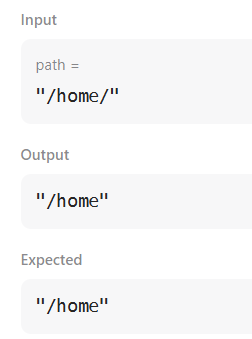
}

}

return "/"+String.join("/",stack1);

}

}



**Time Complexity:** O(n)

1. **MinStack**

class MinStack {

Stack<Integer> stack;

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.pop().equals(minStack.peek())) {

minStack.pop();

}

}

public int top() {

return stack.peek();

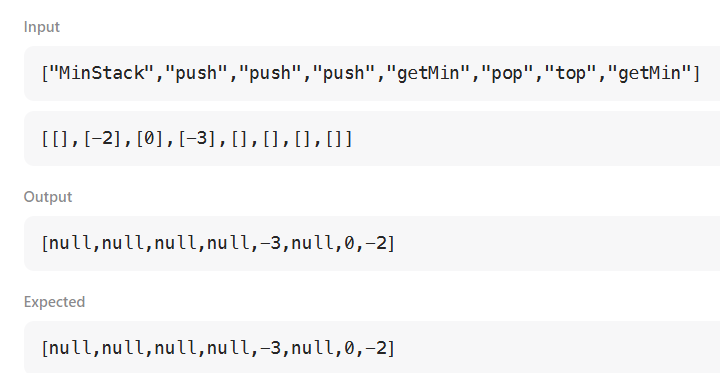
}

public int getMin() {

return minStack.peek();

}

}



**Time Complexity:** O (1)

1. **Evaluate reverse polish notation**

class Solution {

public int evalRPN(String[] tokens) {

Stack<Integer> stack1=new Stack<>();

for(String token:tokens){

switch(token){

case "+" ->{

int val1=stack1.pop();

int val2=stack1.pop();

stack1.add(val1+val2);

}

case "-" ->{

int val1=stack1.pop();

int val2=stack1.pop();

stack1.add(val2-val1);

}

case "\*" ->{

int val1=stack1.pop();

int val2=stack1.pop();

stack1.add(val2\*val1);

}

case "/" ->{

int val1=stack1.pop();

int val2=stack1.pop();

stack1.add(val2/val1);

}

default ->{

stack1.add(Integer.parseInt(token));

}

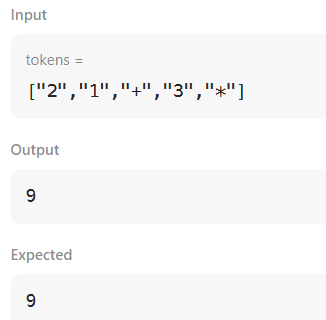
}

}

return stack1.pop();

}

}



**Time Complexity:** O(n)

1. **Basic Calculator**

class Solution {

public int calculate(String s) {

if(s == null) return 0;

int result = 0;

int sign = 1;

int num = 0;

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

stack.push(sign);

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

char c = s.charAt(i);

if(c >= '0' && c <= '9') {

num = num \* 10 + (c - '0');

} else if(c == '+' || c == '-') {

result += sign \* num;

sign = stack.peek() \* (c == '+' ? 1: -1);

num = 0;

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

stack.push(sign);

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

stack.pop();

}

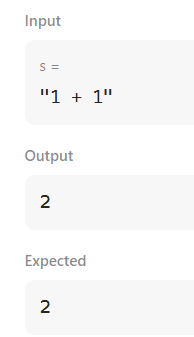
}

result += sign \* num;

return result;

}

}



**Time Complexity:** O(n)