**Date**: 19/11/2024

**DSA Practice Problems**

1. **Next Permutation**

class Solution {

public void nextPermutation(int[] nums) {

int index1=-1;

int index2=-1;

for(int i=nums.length-2;i>=0;i--){

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

index1=i;

break;

}

}

if(index1==-1){

reverse(nums,0);

}

else{

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

if(nums[i]>nums[index1]){

index2=i;

break;

}

}

swap(nums,index1,index2);

reverse(nums,index1+1);

}

}

void swap(int[] nums,int i,int j){

int temp=nums[i];

nums[i]=nums[j];

nums[j]=temp;

}

void reverse(int[] nums,int start){

int i=start;

int j=nums.length-1;

while(i<j){

swap(nums,i,j);

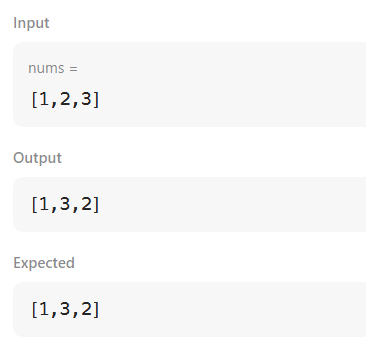
i++;

j--;

}

}

}



**Time Complexity**: O(n)

1. **Spiral Matrix**

class Solution {

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

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

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

return result;

}

int m = matrix.length;

int n = matrix[0].length;

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

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

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

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

}

top++;

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

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

}

right--;

if (top <= bottom) {

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

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

}

bottom--;

}

if (left <= right) {

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

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

}

left++;

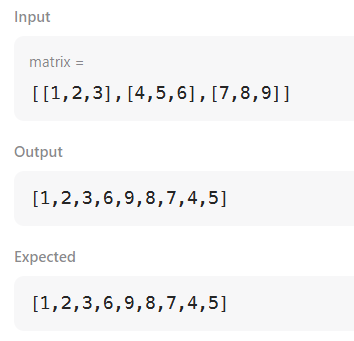
}

}

return result;

}

}



**Time Complexity**: O(m\*n)

1. **Longest substring without repeating characters**

class Solution {

public int lengthOfLongestSubstring(String s) {

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

HashMap<Character, Integer> charIndexMap = new HashMap<>();

int maxLength = 0;

int start = 0;

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

char currentChar = s.charAt(end);

if (charIndexMap.containsKey(currentChar) && charIndexMap.get(currentChar) >= start) {

start = charIndexMap.get(currentChar) + 1;

}

charIndexMap.put(currentChar, end);

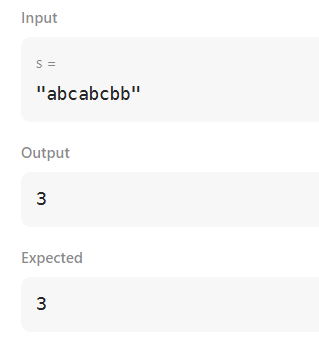
maxLength = Math.max(maxLength, end - start + 1);

}

return maxLength;

}

}



**Time Complexity:** O(n)

1. **Remove linked list elements**

class Solution {

public ListNode removeElements(ListNode head, int val) {

ListNode sol=new ListNode(0,head);

ListNode pointer=sol;

while(pointer!=null){

while(pointer.next!=null && pointer.next.val==val){

pointer.next=pointer.next.next;

}

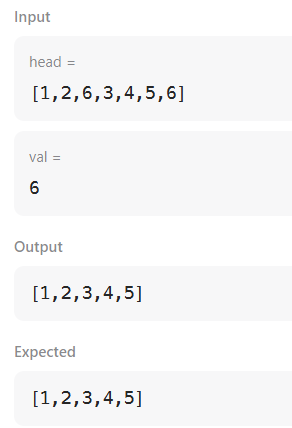
pointer=pointer.next;

}

return sol.next;

}

}



**Time Complexity:** O(n)

1. **Palindrome linked list**

private ListNode reverseList(ListNode head) {

ListNode prev = null;

while (head != null) {

ListNode next = head.next;

head.next = prev;

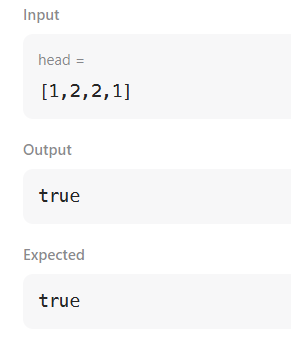
prev = head;

head = next;

}

return prev;

}



**Time Complexity:** O(n)

1. **Minimum path sum**

class Solution {

public int minPathSum(int[][] grid) {

int m = grid.length;

int n = grid[0].length;

int[][] dp = new int[m][n];

dp[0][0] = grid[0][0];

for (int j = 1; j < n; j++) {

dp[0][j] = dp[0][j - 1] + grid[0][j];

}

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

dp[i][0] = dp[i - 1][0] + grid[i][0];

}

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

for (int j = 1; j < n; j++) {

dp[i][j] = grid[i][j] + Math.min(dp[i - 1][j], dp[i][j - 1]);

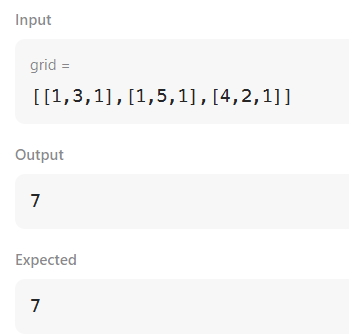
}

}

return dp[m - 1][n - 1];

}

}



**Time Complexity:** O(m\*n)

1. **Validate binary search tree**

class Solution {

public boolean isValidBST(TreeNode root) {

if (root == null){

return true;

}

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

TreeNode prev = null;

while (root != null || !stack1.isEmpty()) {

while (root != null) {

stack1.push(root);

root = root.left;

}

root = stack1.pop();

if(prev != null && root.val <= prev.val){

return false;

}

prev = root;

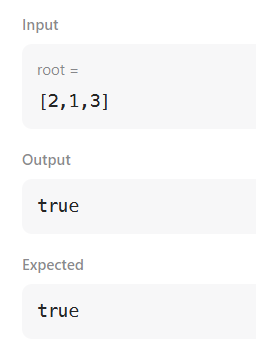
root = root.right;

}

return true;

}

}



**Time Complexity:** O(m\*n)