

Assignment: Data Structures Mastery on LeetCode

**Personal Informa-on (First Page):**

* Javed AlI || Jitandar Kumar
* 023-22-0191 || 023-22-0362
* Data Structures and Algorithms
* Saif Hassan
* Computer Science
* Sukkur IBA University

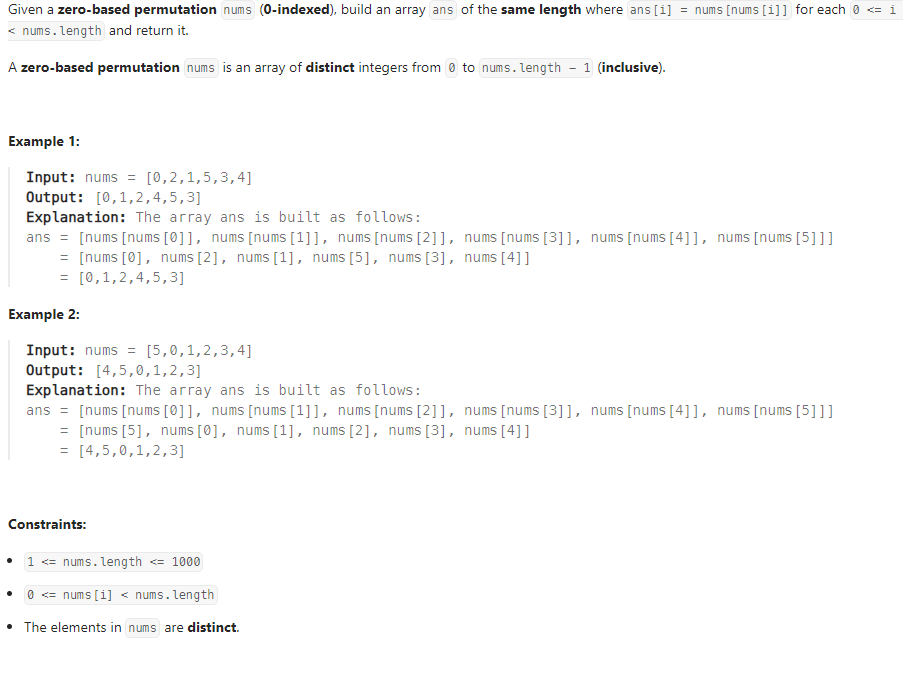
**Table of Contents (Second Page):**

* Contents with Page Numbers

**Task Presenta-on (Star-ng from Third Page Onwards):**

**Topic 1: Arrays**

* Task 1: Screenshot and Code



class Solution {

public int[] buildArray(int[] nums) {

int[] ans=new int[nums.length];

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

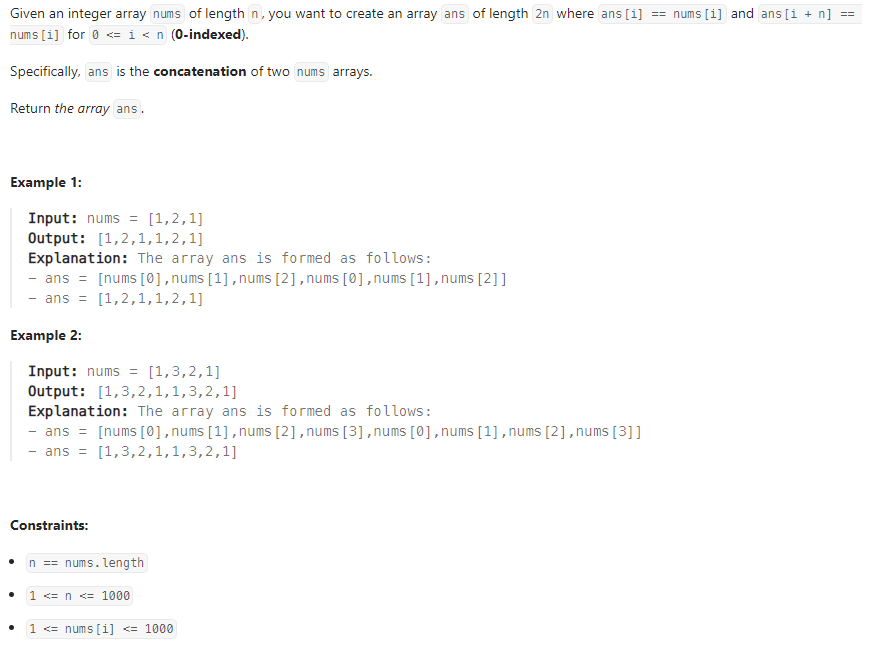
ans[i]=nums[nums[i]];

return ans;

}

}

* Task 2: Screenshot and Code



class Solution {

public int[] getConcatenation(int[] nums) {

int[] ans = new int[2\*(nums.length)];

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

ans[i] = nums[i];

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

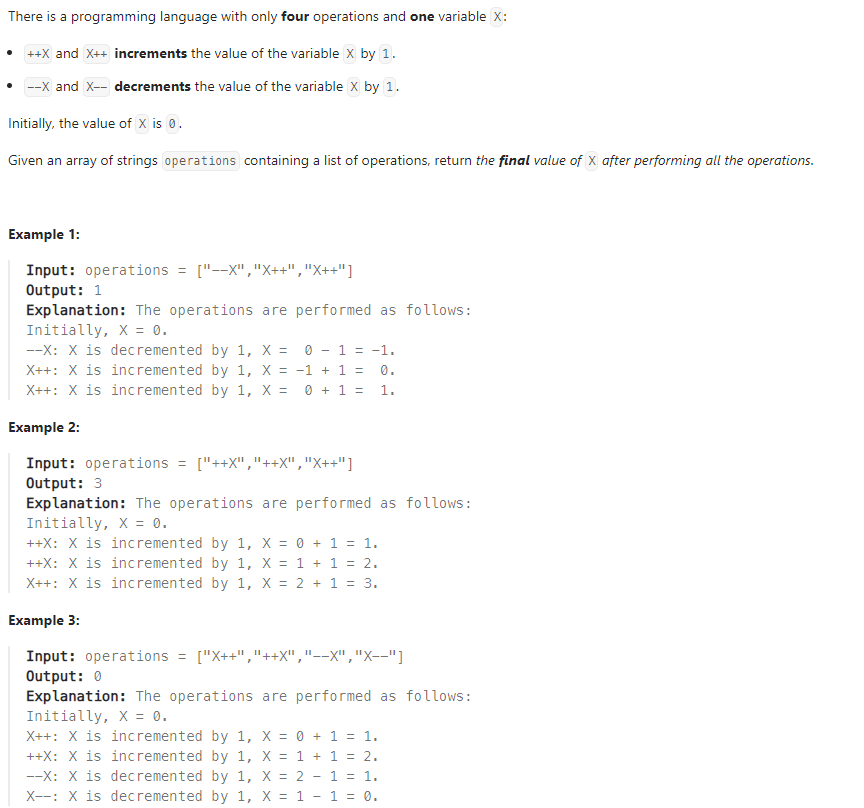
ans[i+nums.length] = nums[i];

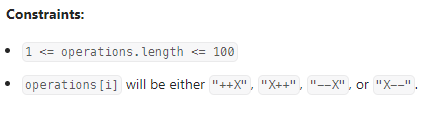
return ans;

}

}

* Task 3: Screenshot and Code





class Solution {

public int finalValueAfterOperations(String[] operations) {

int X=0;

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

if (operations[i].equals("X++"))

X++;

else if (operations[i].equals("++X"))

++X;

else if (operations[i].equals("X--"))

X--;

else if (operations[i].equals("--X"))

--X;

else

X=X;

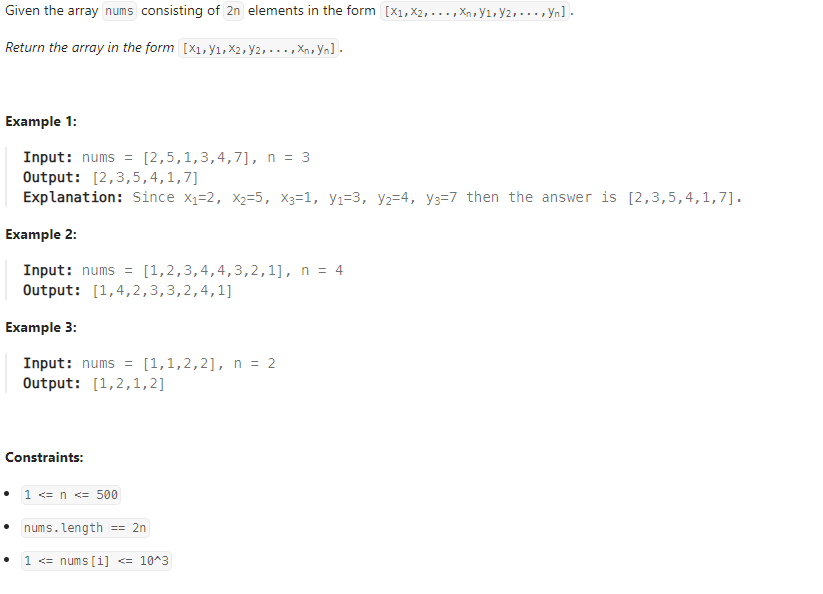
}

return X;

}

}

* Task 4: Screenshot and Code



class Solution {

public int[] shuffle(int[] nums, int n) {

int[] left = new int[n];

int [] right =new int[n];

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

left[i]=nums[i];

right[i]=nums[i+n];

}

int index= 0;

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

nums[index++]=left[i];

nums[index++]=right[i];

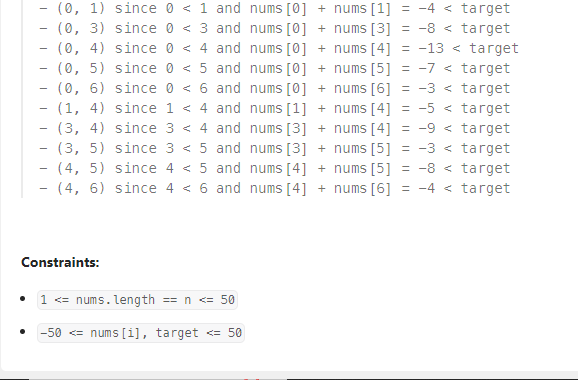
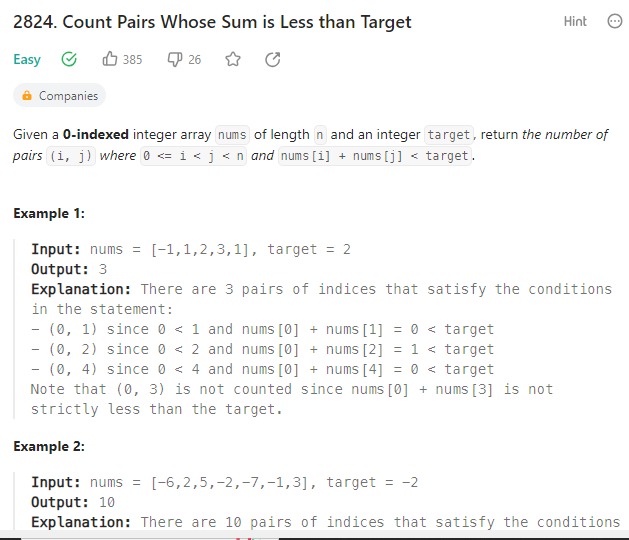
}

return nums;

}

}

* Task 5: Screenshot and Code



import java.util.\*;

public class Solution {

public int countPairs(List<Integer> nums, int target) {

Collections.sort(nums);

int count = 0;

int left = 0;

int right = nums.size() - 1;

while (left < right) {

int sum = nums.get(left) + nums.get(right);

if (sum < target) {

// If nums[left] + nums[right] < target,

// then all pairs (left, k) for k in (left+1, right) are also valid.

count += right - left;

left++;

} else {

// If nums[left] + nums[right] >= target,

// move the right pointer to the left.

right--;

}

}

return count;

}

public static void main(String[] args) {

Solution solution = new Solution();

// Example usage:

List<Integer> nums = Arrays.asList(-1, 1, 2, 3, 1);

int target = 2;

int result = solution.countPairs(nums, target);

System.out.println("Output: " + result);

}

}

**Topic 2: Strings**

* Task 6: Screenshot and Code



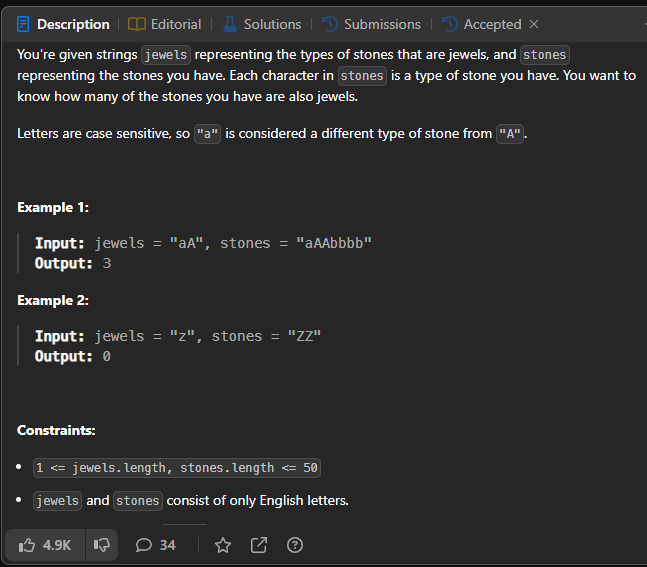
class Solution {

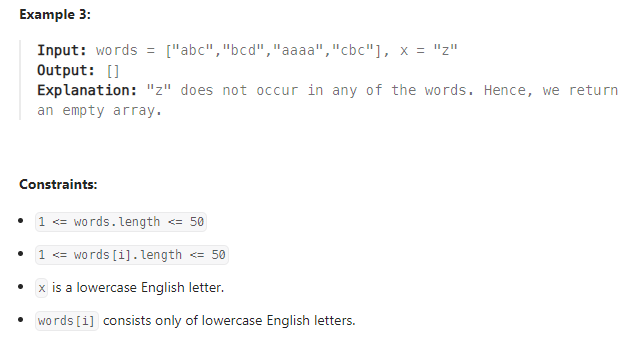
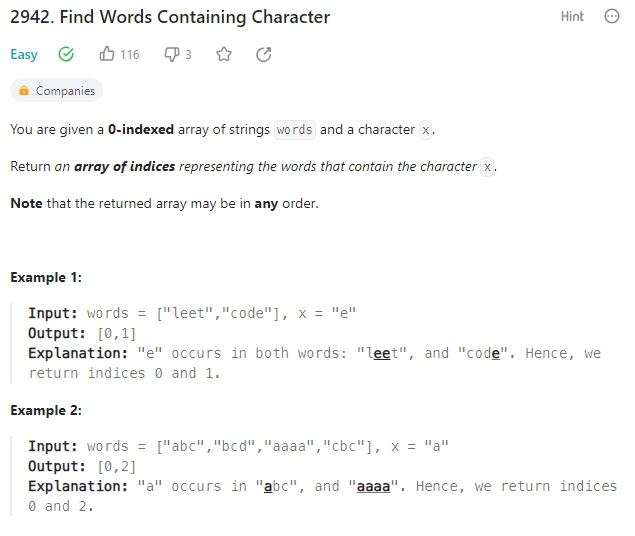
public String defangIPaddr(String address) {

return address.replace(".", "[.]");

}

}

* Task 7: Screenshot and Code
* 
* public int numJewelsInStones(String jewels, String stones) {
* int count = 0;
* for(int i=0; i<stones.length(); i++){
* if(jewels.contains(String.*valueOf*(stones.charAt(i))))
* count++;
* }
* return count;
* }
* Task 8: Screenshot and Code



import java.util.ArrayList;

import java.util.List;

public class Solution {

public List<Integer> findWordsContaining(String[] words, char x) {

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

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

if (words[i].indexOf(x) != -1) {

result.add(i);

}

}

return result;

}

public static void main(String[] args) {

Solution solution = new Solution();

// Example usage:

String[] words = {"apple", "banana", "cherry", "date"};

char characterX = 'a';

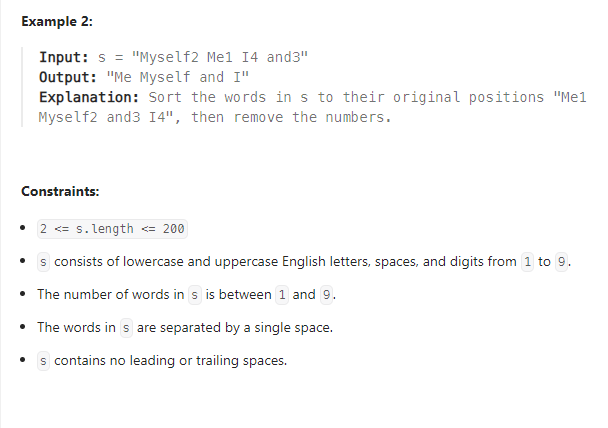
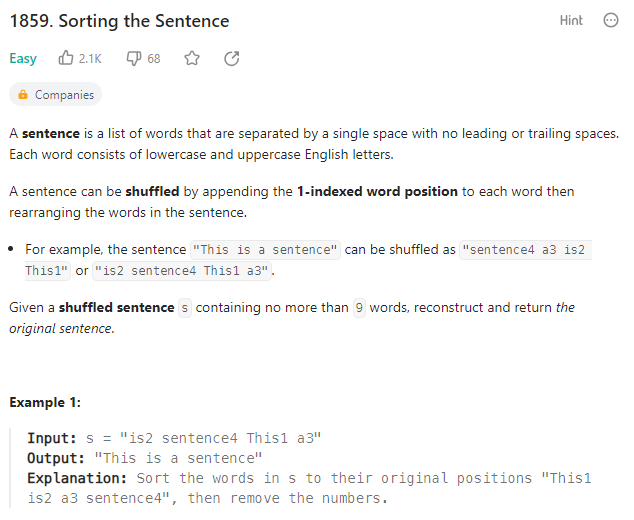
List<Integer> resultIndices = solution.findWordsContaining(words, characterX);

System.out.println(resultIndices);

}

}

* Task 9: Screenshot and Code

class Solution {

public String sortSentence(String s) {

String[] words = s.split(" ");

String[] sortedWords = new String[words.length];

for (String word : words) {

int lastCharIndex = word.length() - 1;

int wordIndex = word.charAt(lastCharIndex) - '0';

sortedWords[wordIndex - 1] = word.substring(0, lastCharIndex);

}

return String.join(" ", sortedWords);

}

public static void main(String[] args) {

// Example usage:

Solution solution = new Solution();

String shuffledSentence = "sentence4 a3 is2 This1";

String sortedSentence = solution.sortSentence(shuffledSentence);

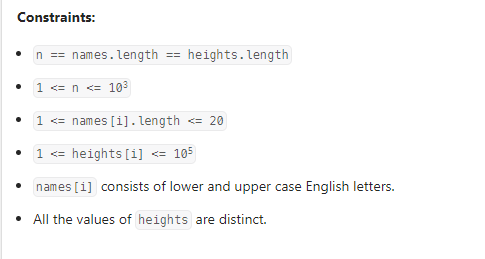
// Print the result

System.out.println("Sorted Sentence: " + sortedSentence);

}

}

* Task 10: Screenshot and Code

import java.util.\*;

class Solution {

public String[] sortPeople(String[] names, int[] heights) {

Comparator<Integer> heightComparator = (a, b) -> Integer.compare(b, a);

List<Pair> pairs = new ArrayList<>();

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

pairs.add(new Pair(names[i], heights[i]));

}

pairs.sort(Comparator.comparing(Pair::getHeight, heightComparator));

// Extract the sorted names

String[] sortedNames = new String[names.length];

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

sortedNames[i] = pairs.get(i).getName();

}

return sortedNames;

}

class Pair {

private String name;

private int height;

public Pair(String name, int height) {

this.name = name;

this.height = height;

}

public String getName() {

return name;

}

public int getHeight() {

return height;

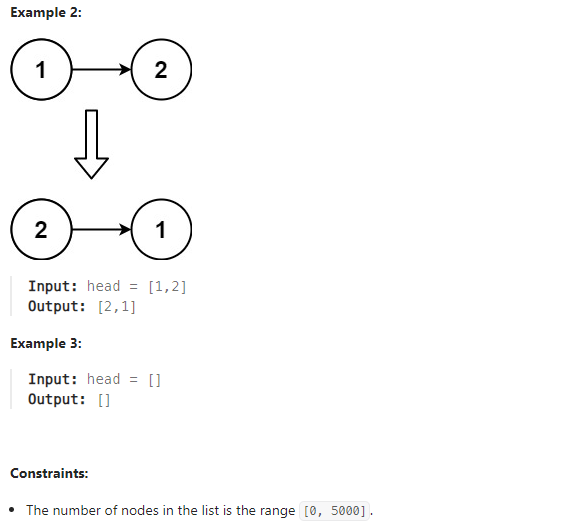
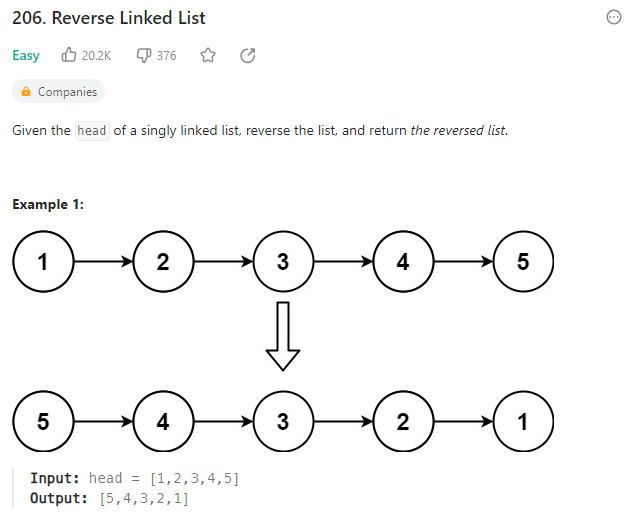
}

}

}

**Topic 3: Linked Lists**

* Task 11: Screenshot and Code

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseList(ListNode head) {

ListNode n=head;

ListNode temp=head;

ListNode prev=null;

while(temp!=null)

{

n=temp.next;

temp.next=prev;

prev=temp;

temp=n;

}

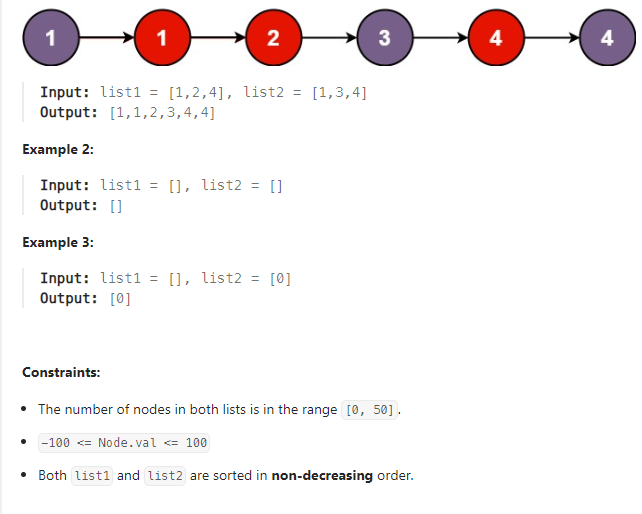
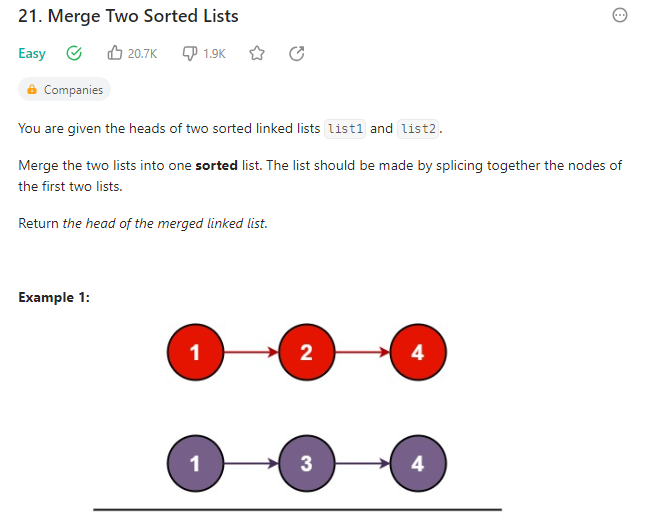
head=prev;

return head;

}

}

* Task 12: Screenshot and Code

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

if(list1!=null && list2!=null){

if(list1.val<list2.val){

list1.next=mergeTwoLists(list1.next,list2);

return list1;

}

else{

list2.next=mergeTwoLists(list1,list2.next);

return list2;

}

}

if(list1==null)

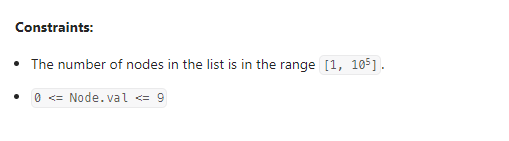
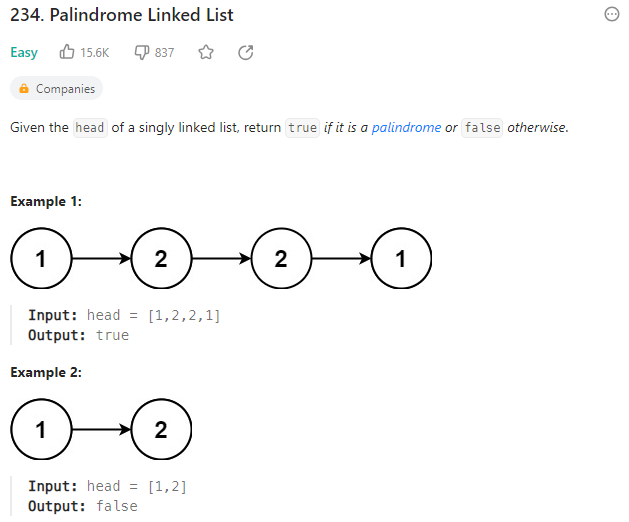
return list2;

return list1;

}

}

* Task 13: Screenshot and Code

class Solution {

public boolean isPalindrome(ListNode head) {

if (head == null || head.next == null) {

return true;

}

ListNode slow = head;

ListNode fast = head;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

}

ListNode reversedSecondHalf = reverseList(slow);

while (reversedSecondHalf != null) {

if (head.val != reversedSecondHalf.val) {

return false;

}

head = head.next;

reversedSecondHalf = reversedSecondHalf.next;

}

return true;

}

private ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode current = head;

while (current != null) {

ListNode next = current.next;

current.next = prev;

prev = current;

current = next;

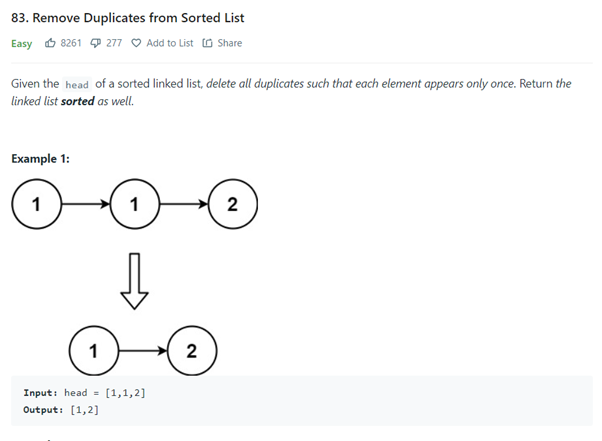
}

return prev;

}

}

* Task 14: Screenshot and Code

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head==null||head.next==null)

return head;

ListNode slow = head;

ListNode fast = head.next;

while(fast!=null){

if(slow.val==fast.val){

slow.next=fast.next;

}else{

slow=slow.next;

}

fast=fast.next;

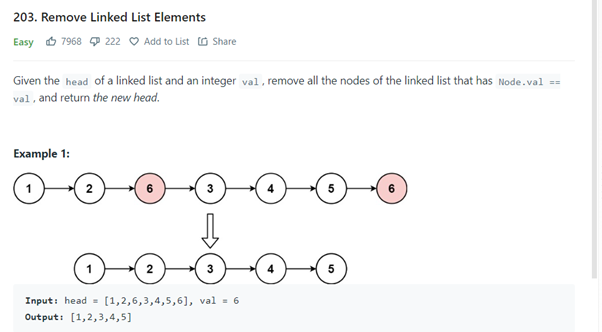
}

return head;

}

}

* Task 15: Screenshot and Code

class Solution {

public ListNode removeElements(ListNode head, int val) {

// Dummy node to simplify handling the head

ListNode dummy=new ListNode(0);

dummy.next=head;

ListNode temp=dummy;

while(temp.next!=null){

if(temp.next.val==val){

temp.next=temp.next.next;

}else{

temp=temp.next;

}

}

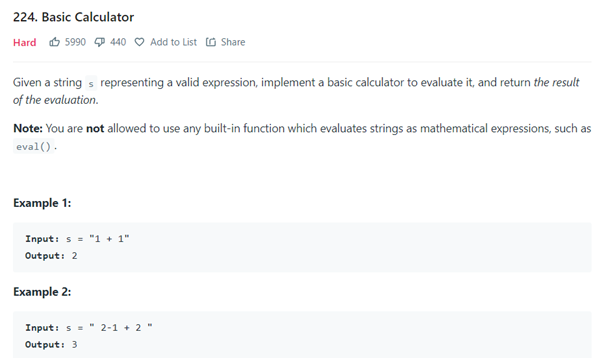
return dummy.next;

}

}

**Topic 4: Stacks**

* Task 16: Screenshot and Code



class Solution {

public:

int calculate(string s) {

int value = 0;

int res = 0;

int sign = 1;

stack<int> stk;

stack<int> stksing;

for(char c : s){

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

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

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

res += sign \* value;

value = 0;

sign = 1;

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

res += sign \* value;

value = 0;

sign = -1;

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

stk.push(res);

stksing.push(sign);

res = 0;

sign = 1;

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

res += sign \* value;

res \*= stksing.top(); stksing.pop();

res += stk.top(); stk.pop();

value = 0;

}

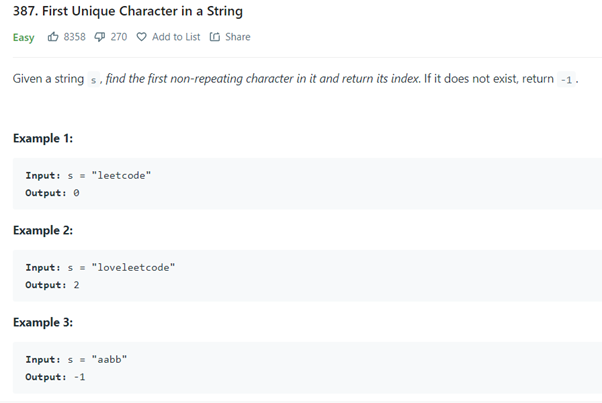
}

return res + sign \* value;

}

};

* Task 17: Screenshot and Code



**class Solution {**

**public int minLength(String s) {**

**StringBuilder s1=new StringBuilder(s);**

**while(true)**

**{**

**boolean checker=true;**

**for(int i=0;i<s1.length()-1;i++)**

**{**

**char a=s1.charAt(i);**

**char b=s1.charAt(i+1);**

**if((a=='A' && b=='B')||(a=='C' && b=='D')){**

**s1.deleteCharAt(i);**

**s1.deleteCharAt(i);**

**checker=false;**

**}**

**}**

**if(checker) break;**

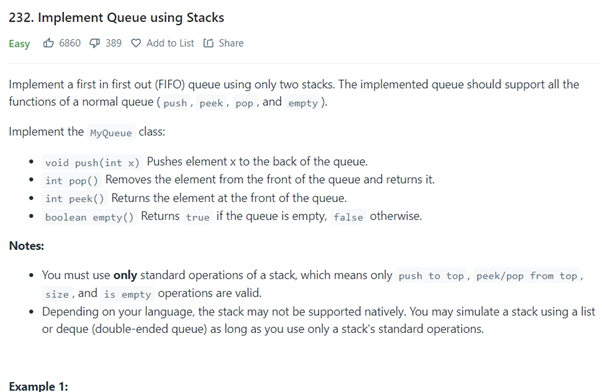
**}**

**return s1.length();**

**}**

**}**

* Task 18: Screenshot and Code



class Solution {

public int minOperations(String[] logs) {

List<String> folder = new ArrayList<>();

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

{

if(logs[i].equals("./"))

{

continue;

}

else if(logs[i].equals("../"))

{

if(folder.size()>1)

folder.remove(folder.size()-1);

else

folder.clear();

}

else

folder.add(logs[i]);

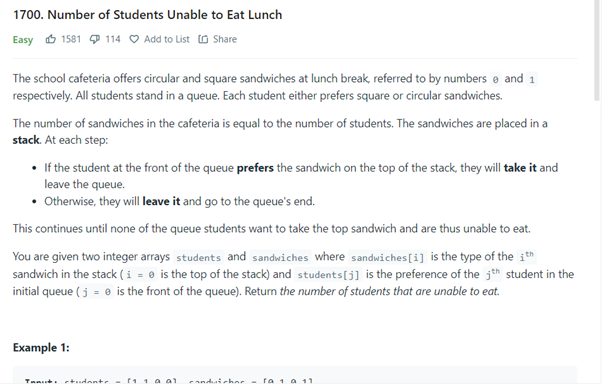
}

return folder.size();

}

}

* Task 19: Screenshot and Code



**class Solution {**

**public int[] finalPrices(int[] prices) {**

**int arr[] = new int[prices.length];**

**int i;**

**int j;**

**for(i = 0; i < prices.length; i++) {**

**for(j = i + 1; j < prices.length; j++) {**

**if(prices[j] <= prices[i]) {**

**arr[i] = prices[i] - prices[j];**

**break;**

**}**

**}**

**if(j == prices.length) {**

**arr[i] = prices[i];**

**}**

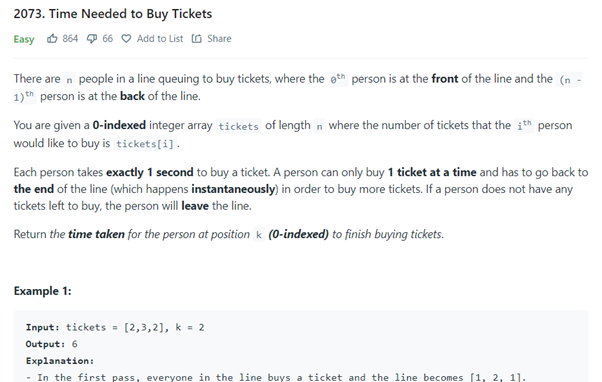
**}**

**return arr;**

**}**

**}**

* Task 20: Screenshot and Code

****

class Solution {

public static String removeDuplicates(String s) {

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

return s;

}

char[] stack = new char[s.length()];

int top = -1;

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

if (top != -1 && stack[top] == c) {

top--;

} else {

stack[++top] = c;

}

}

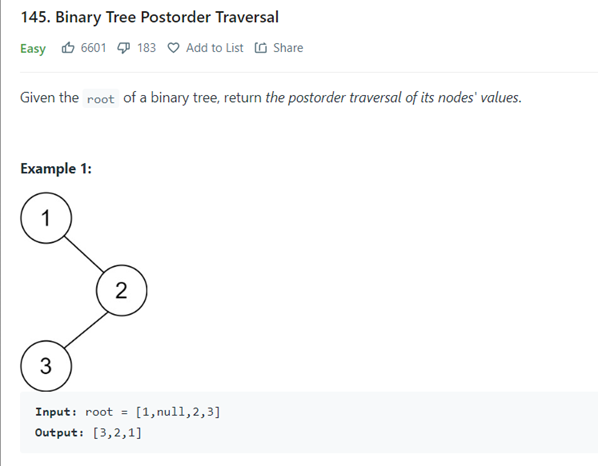
return new String(stack, 0, top + 1);

}

}

**Topic 5: Queues**

* Task 21: Screenshot and Code



**class RecentCounter {**

**ArrayList<Integer> lst;**

**public RecentCounter() {**

**lst=new ArrayList<>();**

**}**

**public int ping(int t) {**

**lst.add(t);**

**int min=t-3000;**

**int max=t;**

**int count=lst.size();**

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

**if(lst.get(i)<min || lst.get(i)>max){**

**count--;**

**}**

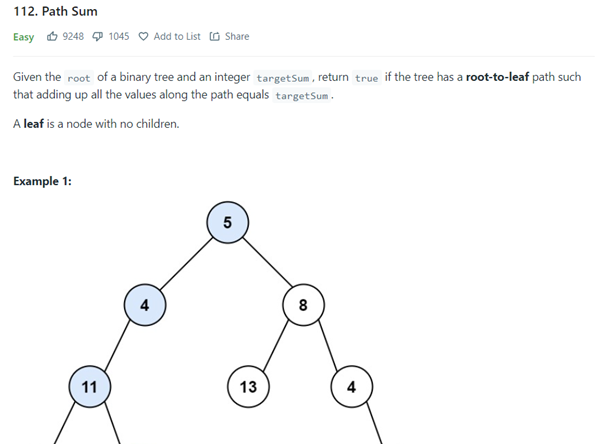
**}**

**return count;**

**}**

**}**

* Task 22: Screenshot and Code



class Solution {

public int firstUniqChar(String s) {

// Stores lowest index / first index

int ans = Integer.MAX\_VALUE;

// Iterate from a to z which is 26 which makes it constant

for(char c='a'; c<='z';c++){

// indexOf will return first index of alphabet and lastIndexOf will return last index

// if both are equal then it has occured only once.

// through this we will get all index's which are occured once

// but our answer is lowest index

int index = s.indexOf(c);

if(index!=-1&&index==s.lastIndexOf(c)){

ans = Math.min(ans,index);

}

}

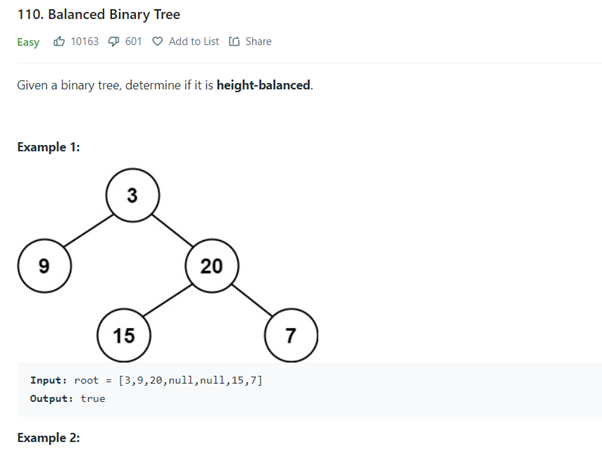
// If ans remain's Integer.MAX\_VALUE then their is no unique character

return ans==Integer.MAX\_VALUE?-1:ans;

}

}

* Task 23: Screenshot and Code



class MyQueue {

ArrayList st1 = new ArrayList();

public MyQueue() {

}

public void push(int x) {

st1.add(x);

}

public int pop() {

return (int)st1.remove(0);

}

public int peek() {

return (int)st1.get(0);

}

public boolean empty() {

if(st1.isEmpty())

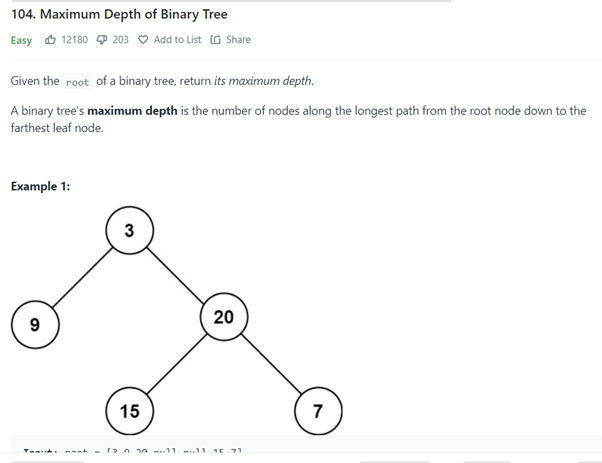
return true;

return false;

}

}

* Task 24: Screenshot and Code



class Solution {

public int countStudents(int[] students, int[] sandwiches) {

int ones = 0; //count of students who prefer type1

int zeros = 0; //count of students who prefer type0

for(int stud : students){

if(stud == 0) zeros++;

else ones++;

}

// for each sandwich in sandwiches

for(int sandwich : sandwiches){

if(sandwich == 0){ // if sandwich is of type0

if(zeros == 0){ // if no student want a type0 sandwich

return ones;

}

zeros--;

}

else{ // if sandwich is of type1

if(ones == 0){ // if no student want a type1 sandwich

return zeros;

}

ones--;

}

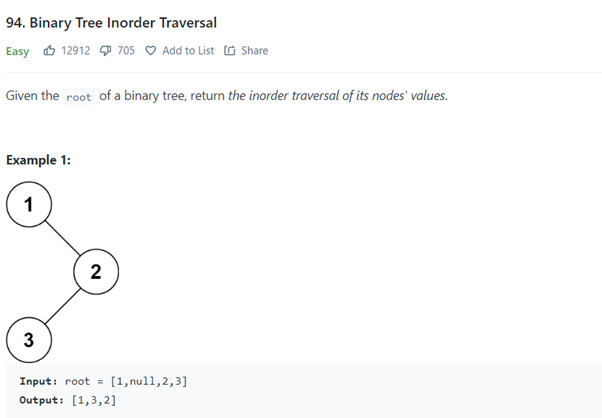
}

return 0;

}

}

* Task 25: Screenshot and Code



class Solution {

public int timeRequiredToBuy(int[] tickets, int k) {

int time = 0;

while(tickets[k] != 0){

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

if(tickets[i] != 0 && tickets[k] != 0){

tickets[i] -= 1;

time += 1;

}

}

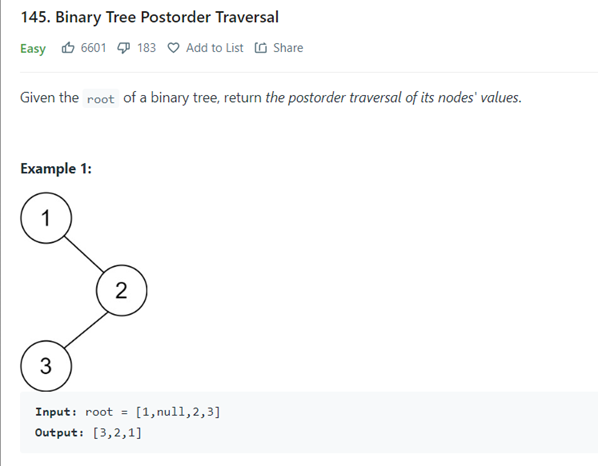
}

return time;

}

}

**Topic 6: Trees and Graphs**

* Task 26: Screenshot and Code
* 

**class Solution {**

**public List<Integer> postorderTraversal(TreeNode root) {**

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

**helper(root, ans);**

**return ans;**

**}**

**private void helper(TreeNode node, List<Integer> ans) {**

**if(node == null) return;**

**helper(node.left, ans);**

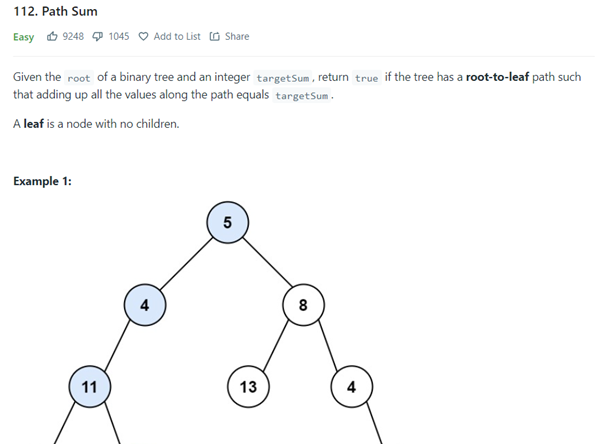
**helper(node.right, ans);**

**ans.add(node.val);**

**}**

**}**

* Task 27: Screenshot and Code



**class Solution {**

**public boolean haspath(TreeNode root,int k,int sum ){**

**if(root == null){**

**return false;**

**}**

**if(root.left == null && root.right == null){**

**sum = sum + root.val;**

**if(sum == k){**

**return true;**

**}**

**}**

**boolean leftl = haspath(root.left,k,sum+root.val);**

**boolean rightl = haspath(root.right,k,sum+root.val);**

**return (leftl || rightl);**

**}**

**public boolean hasPathSum(TreeNode root, int targetSum) {**

**if(root == null){**

**return false;**

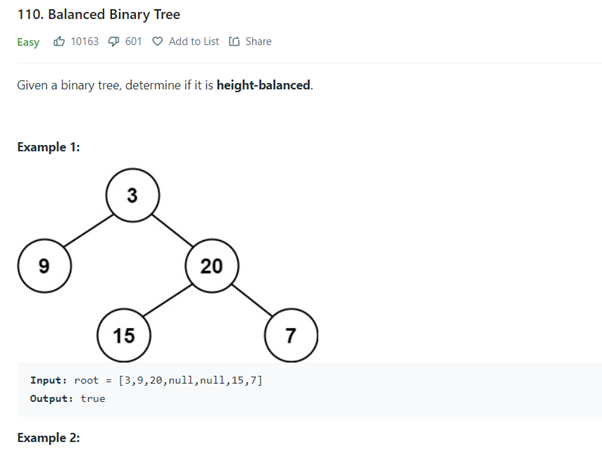
**}**

**return haspath(root,targetSum,0);**

**}**

**}**

* Task 28: Screenshot and Code



**class Solution {**

**public boolean isBalanced(TreeNode root) {**

**return isBalancedAndHeight(root) >= 0;**

**}**

**public int isBalancedAndHeight(TreeNode root) {**

**if (root == null) {**

**return 0;**

**}**

**int l = isBalancedAndHeight(root.left);**

**if (l < 0) {**

**return l;**

**}**

**int r = isBalancedAndHeight(root.right);**

**if (r < 0) {**

**return r;**

**}**

**if (Math.abs(l-r) > 1) {**

**return -Math.abs(l-r);**

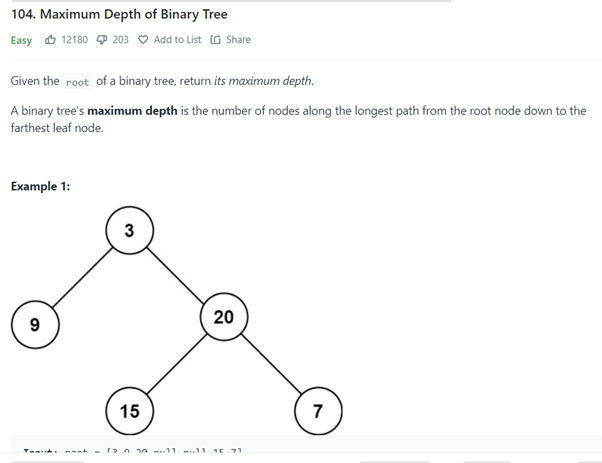
**}**

**return Math.max(l, r) + 1;**

**}**

**}**

* Task 29: Screenshot and Code



**class Solution {**

**public int maxDepth(TreeNode root) {**

**// Base Condition**

**if(root == null) return 0;**

**// Hypothesis**

**int left = maxDepth(root.left);**

**int right = maxDepth(root.right);**

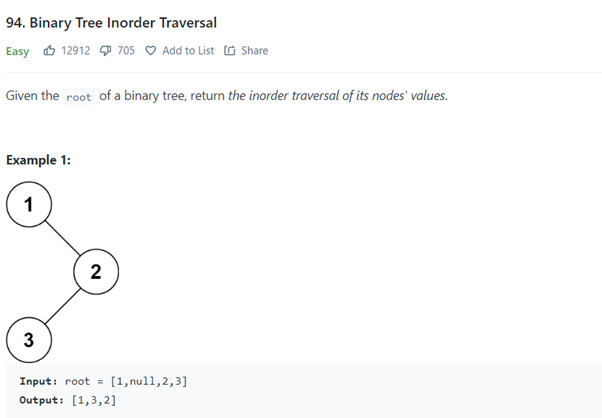
**// Induction**

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

**}**

**}**

* Task 30: Screenshot and Code



**class Solution {**

**public List<Integer> inorderTraversal(TreeNode root) {**

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

**helper(root, res);**

**return res;**

**}**

**public void helper(TreeNode root, List<Integer> res) {**

**if (root != null) {**

**helper(root.left, res);**

**res.add(root.val);**

**helper(root.right, res);**

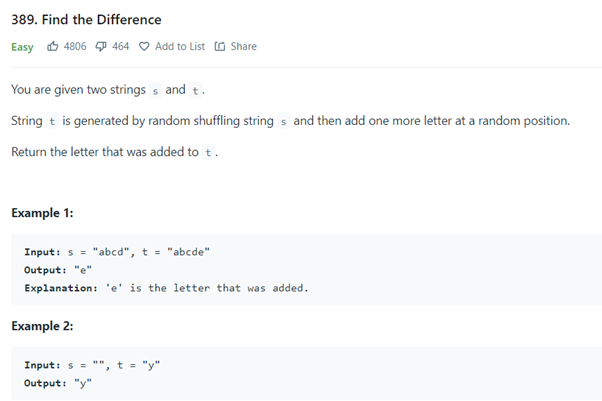
**}**

**}**

**}**

**Topic 7: Hashing and Sor-ng**

* Task 31: Screenshot and Code



**class Solution {**

**public char findTheDifference(String s, String t) {**

**char c = 0;**

**for(char cs : s.toCharArray()) c ^= cs;**

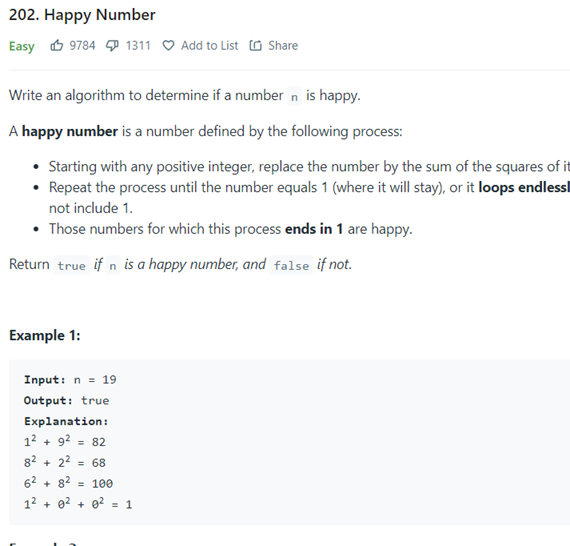
**for(char ct : t.toCharArray()) c ^= ct;**

**return c;**

**}**

**}**

* Task 32: Screenshot and Code



**class Solution {**

**public boolean isHappy(int n) {**

**int sum = n;//GIVE VALUE TO SUM**

**while (sum != 1 && sum!=4 ) {**

**int temp = sum;//THEN TEMP**

**sum = 0;**

**while (temp != 0 ) {**

**int digit = temp % 10;**

**sum += digit \* digit;//ADDING VALUES**

**temp =temp/ 10;**

**}**

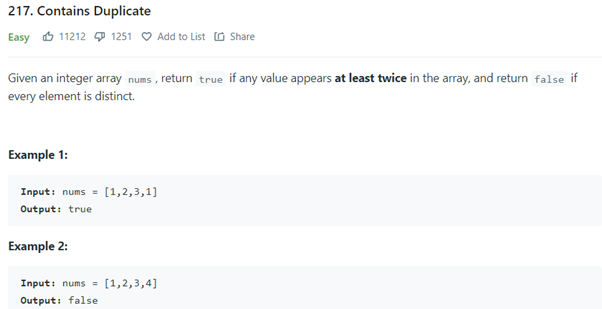
**}**

**return sum == 1;**

**}**

**}**

* Task 33: Screenshot and Code



**class Solution {**

**public boolean containsDuplicate(int[] nums) {**

**Arrays.sort(nums);**

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

**if (nums[i] == nums[i+1]) {**

**return true;**

**}**

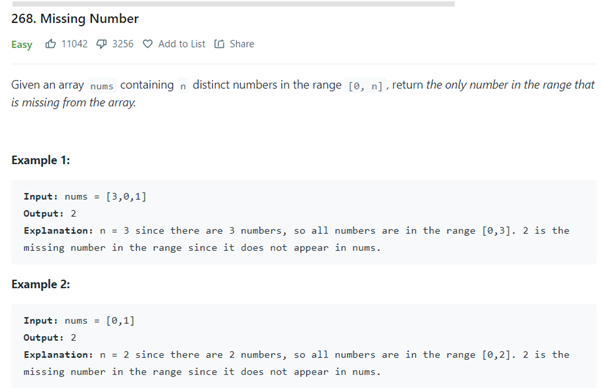
**}**

**return false;**

**}**

**}**

* Task 34: Screenshot and Code



**class Solution {**

**public int missingNumber(int[] nums) {**

**int n = nums.length;**

**int totalSum = n\*(n+1)/2;**

**int sum = 0;**

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

**sum = sum+nums[i];**

**}**

**return totalSum - sum;**

**}**

**}**

* Task 35: Screenshot and Code



**import java.util.Hashtable;**

**class Solution {**

**public boolean wordPattern(String pattern, String s) {**

**String[] arr = s.split(" ");**

**if (arr.length != pattern.length()) return false;**

**Hashtable<Character, String> hashtable = new Hashtable<>();**

**Submission Guidelines:**

* Submit a single word and its PDF version file with the specified format.
* The assignment can be done in a group of a maximum of two students.
* Ensure the correctness of code (text format) and clarity of screenshots.
* Use appropriate headings and labels for each task and topic.
* Follow the order of topics and tasks as men5oned in the assignment.
* Plagiarism is strictly prohibited. If caught, zero marks will be assigned for the en5re assignment for all involved students.
* AYer submission, be prepared for a viva session where you may be called to explain your solu5ons and demonstrate understanding of the tasks.

**Note:**

* Make sure to replace placeholders (like "Your Name," "Student ID," etc.) with your actual informa5on.
* Provide complete and working code for each task.
* Include any addi5onal explana5ons or comments if needed for be\er understanding.