GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 11-11-24 **NAME: JASHVARTHINI R – CSBS**

1. 0-1 KNAPSACK PROBLEM

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Examples:

Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3

```
import java.util.*;

public class Knapsnack{

    public static int knapsack(int[] profit, int[] weight, int W, int n){
        if(n==0 || W==0){
            return 0;
        }

        if (weight[n-1] > W){
            return knapsack(profit, weight, W, n-1);
        }

        else{
            return Math.max(knapsack(profit, weight, W, n-1), profit[n-
1]+knapsack(profit, weight, W-weight[n-1], n-1));

        }
    }

    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of items: ");
        int n= scanner.nextInt();

        int profit[]=new int[n];
        int weight[]=new int[n];

        System.out.println("Enter the profit of each item: ");
        for(int i=0; i<n; i++){
            profit[i]=scanner.nextInt();
```

```
        }

        System.out.println("Enter the weight of each item: ");
        for(int i=0; i<n; i++){
            weight[i]=scanner.nextInt();
        }

    System.out.println("Enter the capacity of knapsack: ");
    int W=scanner.nextInt();

    scanner.close();

    System.out.println("The maximum profit is: "+knapsack(profit, weight, W, n));

    }}
```

OUTPUT:

```
Enter the number of items:
3
Enter the profit of each item:
60
100
120
Enter the weight of each item:
10
20
30
Enter the capacity of knapsack:
50
The maximum profit is: 220
```

TIME COMPLEXITY: 0(2^n)

2. FLOOR IN SORTED ARRAY

Given a sorted array arr[] (with unique elements) and an integer k, find the index (0-based) of the largest element in arr[] that is less than or equal to k. This element is called the "floor" of k. If such an element does not exist, return -1.

Examples

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 0

Output: -1

Explanation: No element less than 0 is found. So output is -1.

```
import java.util.*;
```

```java
public class Floor{

    public static int floor(int[] arr, int k){
        int start=0;
        int end=arr.length-1;
        int res=-1;
        while(start<=end){
            int mid=(start+end)/2;
            if(arr[mid]==k){
                return mid;
            }
            else if(arr[mid]<k){
                res=mid;
                start=mid+1;
            }
            else{
                end=mid-1;
            }
        }
        return res;

    }
    public static void main(String[] args){
        Scanner sc= new Scanner(System.in);

        System.out.println("Enter the number of elements in the array:");
        int n= sc.nextInt();

        int[] arr= new int[n];
        System.out.println("Enter the elements:");
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }

        System.out.println("Enter the value of k:");
        int k= sc.nextInt();

        System.out.println("The floor of " +k +" is: "+ floor(arr, k));
    }
}
```

OUTPUT:

```
Enter the number of elements in the array:
7
Enter the elements:
1
2
8
10
11
12
19
Enter the value of k:
5
The floor of 5 is: 2
```

```
Enter the number of elements in the array:
7
Enter the elements:
1
2
8
10
11
12
19
Enter the value of k:
0
The floor of 0 is: -1
```

```
Enter the number of elements in the array:
3
Enter the elements:
1
2
8
Enter the value of k:
1
The floor of 1 is: 0
```

TIME COMPLEXITY: O(log n)

3. CHECK EQUAL ARRAYS

Given two arrays, arr1 and arr2 of equal length N, the task is to determine if the given arrays are equal or not. Two arrays are considered equal if:

Both arrays contain the same set of elements.

The arrangements (or permutations) of elements may be different.

If there are repeated elements, the counts of each element must be the same in both arrays.

Examples:

Input: arr1[] = {1, 2, 5, 4, 0}, arr2[] = {2, 4, 5, 0, 1}
Output: Yes

Input: arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}
Output: Yes

Input: arr1[] = {1, 7, 1}, arr2[] = {7, 7, 1}
Output: No

```java
import java.util.*;

public class EqualArrays {
    public static boolean checkEqualArrays(int[] arr1, int[] arr2) {

        if (arr1.length != arr2.length) {
            return false;
        }

        Arrays.sort(arr1);
        Arrays.sort(arr2);

        for (int i = 0; i < arr1.length; i++) {
            if (arr1[i] != arr2[i]) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in array 1:");
        int n1 = sc.nextInt();

        int[] arr1 = new int[n1];
        System.out.println("Enter the elements in array 1:");
        for (int i = 0; i < n1; i++) {
            arr1[i] = sc.nextInt();
        }

        System.out.println("Enter the number of elements in array 2:");
        int n2 = sc.nextInt();

        int[] arr2 = new int[n2];
        System.out.println("Enter the elements in array 2:");
        for (int i = 0; i < n2; i++) {
            arr2[i] = sc.nextInt();
        }
```

```
        if (checkEqualArrays(arr1, arr2)) {
            System.out.println("The two arrays are equal.");
        } else {
            System.out.println("The two arrays are not equal.");
        }

        sc.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in array 1:
5
Enter the elements in array 1:
1
2
5
4
0
Enter the number of elements in array 2:
5
Enter the elements in array 2:
2
4
5
0
1
The two arrays are equal.
```

```
Enter the number of elements in array 1:
3
Enter the elements in array 1:
1
7
1
Enter the number of elements in array 2:
3
Enter the elements in array 2:
7
7
1
The two arrays are not equal.
```

TIME COMPLEXITY: O(n)

4. PALINDROME LINKED LIST

Given a singly linked list. The task is to check if the given linked list is palindrome or not.

Examples:

Input: head: 1->2->1->1->2->1
Output: true
Explanation: The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.

Input: head: 1->2->3->4
Output: false
Explanation: The given linked list is 1->2->3->4, which is not a palindrome and Hence, the output is false.

```java
import java.util.Scanner;

class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class PalindromeLinkedList {

    public static boolean isPalindrome(Node head) {
        if (head == null || head.next == null) {
            return true;
        }

        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        Node secondHalf = reverseList(slow);
        Node firstHalf = head;

        while (secondHalf != null) {
            if (firstHalf.data != secondHalf.data) {
                return false;
            }
            firstHalf = firstHalf.next;
            secondHalf = secondHalf.next;
        }
        return true;
    }

    public static Node reverseList(Node head) {
        Node prev = null;
        Node curr = head;
```

```
        while (curr != null) {
            Node next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }

    public static Node insert(Node head, int data) {
        Node newNode = new Node(data);
        if (head == null) {
            return newNode;
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
        return head;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Node head = null;
        System.out.println("Enter number of nodes:");
        int n = sc.nextInt();

        System.out.println("Enter the values of nodes:");
        for (int i = 0; i < n; i++) {
            int data = sc.nextInt();
            head = insert(head, data);
        }

        if (isPalindrome(head)) {
            System.out.println("The linked list is a palindrome");
        } else {
            System.out.println("The linked list is not a palindrome");
        }

        sc.close();
    }
}
```

OUTPUT:

```
Enter number of nodes:
5
Enter the values of nodes:
1
2
3
2
1
The linked list is a palindrome
```

```
Enter number of nodes:
6
Enter the values of nodes:
1
2
3
3
2
1
The linked list is a palindrome
```

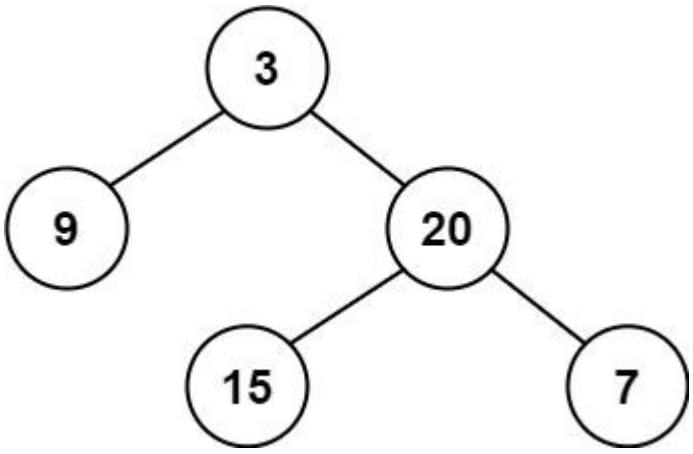
```
Enter number of nodes:
5
Enter the values of nodes:
1
2
3
4
5
The linked list is not a palindrome
```

TIME COMPLEXITY: O(n)

5. BALANCED TREE CHECK

Given a binary tree, determine if it is height-balanced

Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: true

```java
import java.util.Scanner;

class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
        left = right = null;
    }
}

public class Sol {
    public boolean isBalanced(TreeNode root) {
        return height(root) != -1;
    }

    private int height(TreeNode node) {
        if (node == null) {
            return 0;
        }

        int leftHeight = height(node.left);
        if (leftHeight == -1) {
            return -1;
        }

        int rightHeight = height(node.right);
        if (rightHeight == -1) {
            return -1;
        }

        if (Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
        }

        return 1 + Math.max(leftHeight, rightHeight);
    }

    public static TreeNode insertLevelOrder(int[] arr, TreeNode root, int i) {
            if (i < arr.length && arr[i] != -1) {
            TreeNode temp = new TreeNode(arr[i]);
            root = temp;
            root.left = insertLevelOrder(arr, root.left, 2 * i + 1);
            root.right = insertLevelOrder(arr, root.right, 2 * i + 2);
        }
        return root;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of nodes in the tree:");
        int n = sc.nextInt();
        int[] nodes = new int[n];

        System.out.println("Enter node values in level order (-1 for null
nodes):");
        for (int i = 0; i < n; i++) {
            nodes[i] = sc.nextInt();
        }

        TreeNode root = insertLevelOrder(nodes, null, 0);

        Sol solution = new Sol();
        if (solution.isBalanced(root)) {
            System.out.println("The tree is balanced.");
        } else {
            System.out.println("The tree is not balanced.");
        }

        sc.close();
    }
}
```

OUTPUT:

```
Enter number of nodes in the tree:
7
Enter node values in level order (-1 for null nodes):
3
9
20
-1
-1
15
7
The tree is balanced.
```

```
Enter number of nodes in the tree:
0
Enter node values in level order (-1 for null nodes):
The tree is balanced.
```

TIME COMPLEXITY: O(n)

6. **TRIPLET SUM IN ARRAY**

Given an array arr[] of size n and an integer sum. Find if there's a triplet in the array which sums up to the given integer sum.

Examples:

Input: arr = {12, 3, 4, 1, 6, 9}, sum = 24;
Output: 12, 3, 9
Explanation: There is a triplet (12, 3 and 9) present
in the array whose sum is 24.

```java
import java.util.Arrays;
import java.util.Scanner;

public class TripletSumInArray {

    public static boolean find3Numbers(int[] arr, int X) {
        Arrays.sort(arr);
        int n = arr.length;

        for (int i = 0; i < n - 2; i++) {
            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int currentSum = arr[i] + arr[left] + arr[right];
                if (currentSum == X) {
                    return true;
                } else if (currentSum < X) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = sc.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.println("Enter the target sum X:");
        int X = sc.nextInt();
```

```
        if (find3Numbers(arr, X)) {
            System.out.println("Yes, there exists a triplet with the sum " + X);
        } else {
            System.out.println("No, there does not exist a triplet with the sum " +
X);
        }

        sc.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array:
6
Enter the elements of the array:
1
4
45
6
10
8
Enter the target sum X:
22
Yes, there exists a triplet with the sum 22
```

TIME COMPLEXITY: O(n^2)