11-11-2024

DSA Practice (Set - 2)

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:

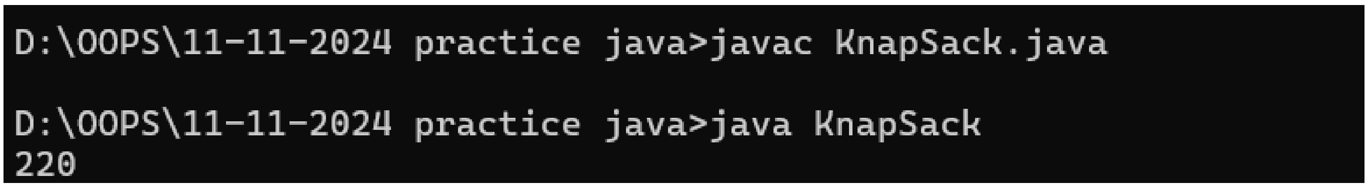
Input: *N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}*
Output: *3*
Explanation: *There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.*

Input: *N = 3, W = 3, profit[] = {1, 2, 3}, weight[] = {4, 5, 6}*
Output: *0*

SOLUTION:

```
import java.util.*;

class KnapSack {

    static int knapSack(int W, int wt[], int val[], int n) {

        int[] dp = new int[W + 1];

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

            for (int w = W; w >= 0; w--) {

                if (wt[i - 1] <= w) {

                    dp[w] = Math.max(dp[w], dp[w - wt[i - 1]] + val[i - 1]);

                }

            }

        }

        return dp[W];

    }

    public static void main(String[] args) {

        int profit[] = { 60, 100, 120 };

        int weight[] = { 10, 20, 30 };

        int W = 50;

        int n = profit.length;

        System.out.print(knapSack(W, weight, profit, n));
    }
}
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac KnapSack.java

D:\OOPS\11-11-2024 practice java>java KnapSack
220
```

Time Complexity : O(N*M)

Space Complexity : O(W)

2. **Floor in a Sorted Array**

Given a sorted array and a value **x**, the floor of x is the largest element in the array smaller than or equal to x. Write efficient functions to find the floor of x

Examples:

Input: *arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 5*
Output: *2*
Explanation: *2 is the largest element in arr[] smaller than 5*

Input: *arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 20*
Output: *19*
Explanation: *19 is the largest element in arr[] smaller than 20*

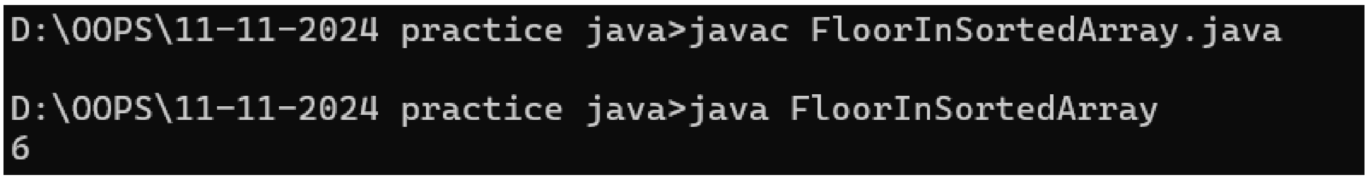
Input : *arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 0*
Output : *-1*
Explanation: *Since floor doesn't exist, output is -1.*

SOLUTION:

```
import java.io.*;

class FloorInSortedArray {

    static int floorSearch(int arr[], int low, int high, int x) {

        if (low > high)

            return -1;

        if (x >= arr[high])

            return high;

        int mid = (low + high) / 2;

        if (arr[mid] == x)

            return mid;

        if (mid > 0 && arr[mid - 1] <= x && x < arr[mid])
```

```
            return mid - 1;
        if (x < arr[mid])
            return floorSearch(arr, low, mid - 1, x);
        return floorSearch(arr, mid + 1, high, x);
    }

    public static void main(String[] args) {
        int arr[] = { 1, 2, 4, 6, 10, 12, 14 };
        int n = arr.length;
        int x = 7;
        int index = floorSearch(arr
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac FloorInSortedArray.java

D:\OOPS\11-11-2024 practice java>java FloorInSortedArray
6
```

Time Complexity : O(log N)
Space Complexity : O(1)

3. **Check if two arrays are equal or not**

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*

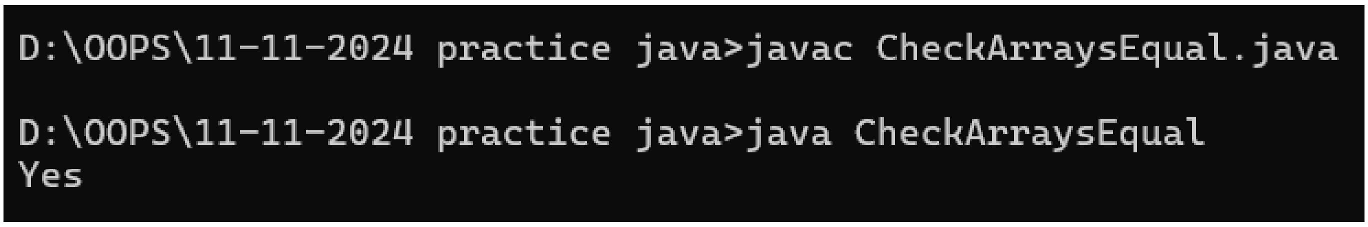
Input: *arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}*
Output: *Yes*

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

SOLUTION:

```
import java.util.*;
class CheckArraysEqual {
    public static boolean areEqual(int arr1[], int arr2[]) {
        if(arr1.length != arr2.length) return false;
        HashMap<Integer, Integer> map = new HashMap<>();
        for(int i=0;i<arr1.length;i++){
            map.put(arr1[i], map.getOrDefault(arr1[i], 0) + 1);
        }
        for(int i=0;i<arr2.length;i++){
            if(!map.containsKey(arr2[i])) return false;
            else{
                if(map.get(arr2[i]) == 0) return false;
                map.put(arr2[i],map.get(arr2[i])-1);
            }
        }
        return true;
    }
    public static void main(String[] args) {
        int arr1[] = { 3, 5, 2, 5, 2 };
        int arr2[] = { 2, 3, 5, 5, 2 };
        if (areEqual(arr1, arr2))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac CheckArraysEqual.java

D:\OOPS\11-11-2024 practice java>java CheckArraysEqual
Yes
```

Time Complexity : O(N)

Space Complexitfy : 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:

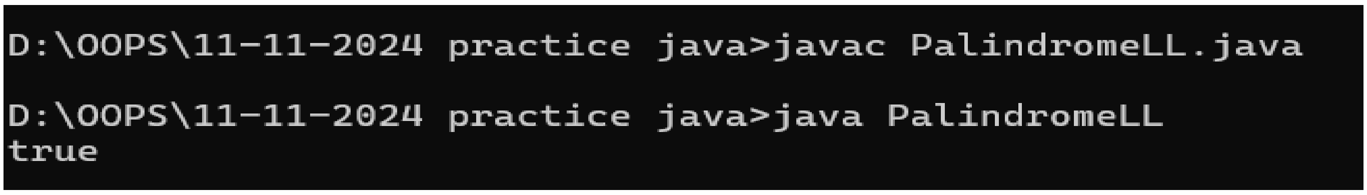
Input: *head: 1->2->1->1->2->1*
Output: *true*
Explanation: *The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.*

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.*

SOLUTION:

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}
class PalindromeLL{
    public static ListNode reverse(ListNode head) {
        ListNode curr = head;
        ListNode next = null;
        ListNode prev = null;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
    public static 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 secondHalfStart = reverse(slow);
        ListNode firstHalfStart = head;
        ListNode secondHalfIterator = secondHalfStart;
        boolean isPalindrome = true;
        while (secondHalfIterator != null) {
            if (firstHalfStart.val != secondHalfIterator.val) {
                isPalindrome = false;
                break;
            }
            firstHalfStart = firstHalfStart.next;
            secondHalfIterator = secondHalfIterator.next;
        }
        reverse(secondHalfStart);
        return isPalindrome;
    }
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(2);
        head.next.next.next.next = new ListNode(1);
        boolean result = isPalindrome(head);
        if (result)
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac PalindromeLL.java

D:\OOPS\11-11-2024 practice java>java PalindromeLL
true
```

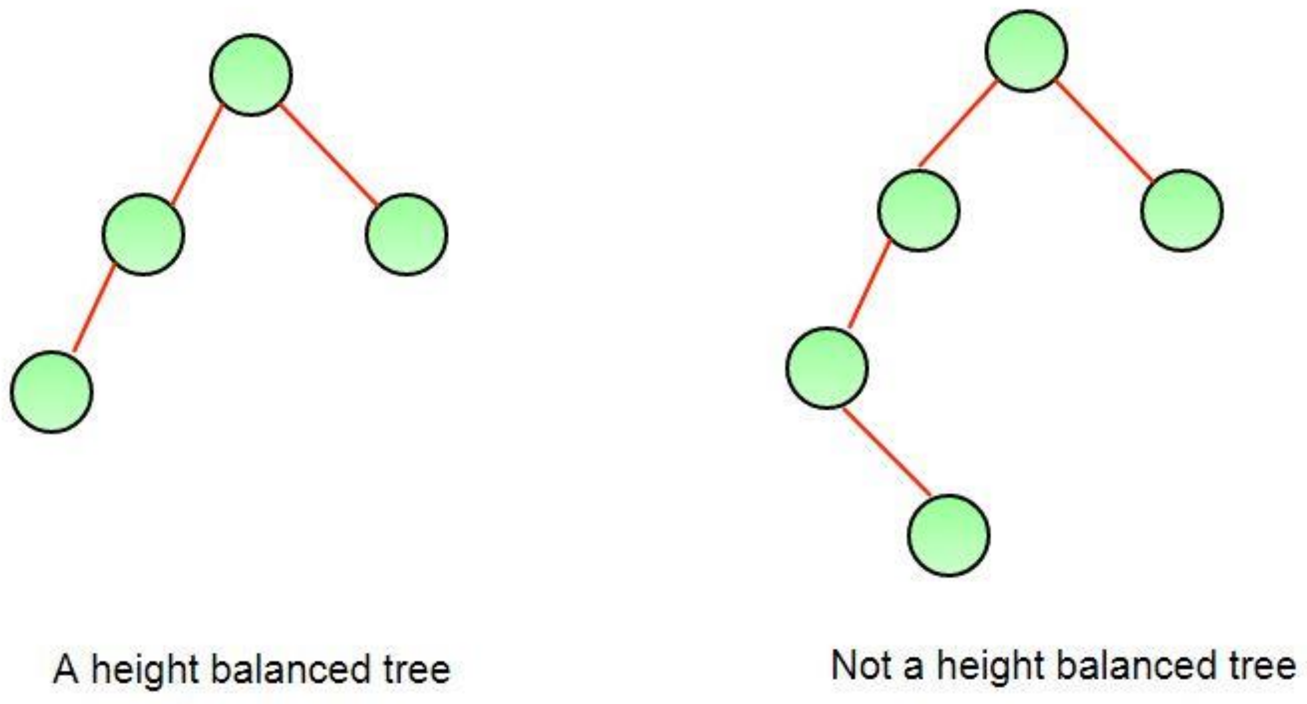
Time Complexity : O(N)

Space Complexity : O(1)

5. **Balanced Binary Tree or Not**

A **height balanced binary tree** is a binary tree in which the height of the left subtree and right subtree of any node does not differ by more than 1 and both the left and right subtree are also height balanced.

Examples: *The tree on the left is a height balanced binary tree. Whereas the tree on the right is not a height balanced tree. Because the left subtree of the root has a height which is 2 more than the height of the right subtree.*



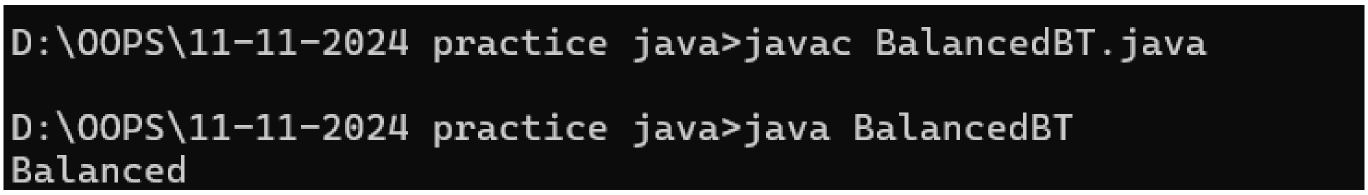
Corner Cases *: An empty binary tree (Root = NULL) and a Binary Tree with single node are considered balanced.*

SOLUTION:

```
import java.util.*;
class Node {
    int key;
    Node left;
    Node right;
    Node(int k) {
        key = k;
        left = right = null;
    }
}
```

```
class BalancedBT {
    public static int isBalanced(Node root) {
        if (root == null)
            return 0;
        int lh = isBalanced(root.left);
        if (lh == -1)
            return -1;
        int rh = isBalanced(root.right);
        if (rh == -1)
            return -1;
        if (Math.abs(lh - rh) > 1)
            return -1;
        else
            return Math.max(lh, rh) + 1;
    }
    public static void main(String args[]) {
        Node root = new Node(10);
        root.left = new Node(5);
        root.right = new Node(30);
        root.right.left = new Node(15);
        root.right.right = new Node(20);
        if (isBalanced(root) > 0)
            System.out.print("Balanced");
        else
            System.out.print("Not Balanced");
    }
}
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac BalancedBT.java

D:\OOPS\11-11-2024 practice java>java BalancedBT
Balanced
```

Time Complexity : O(N)

Space Complexity : O(H)

6. **3 Sum – 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.*

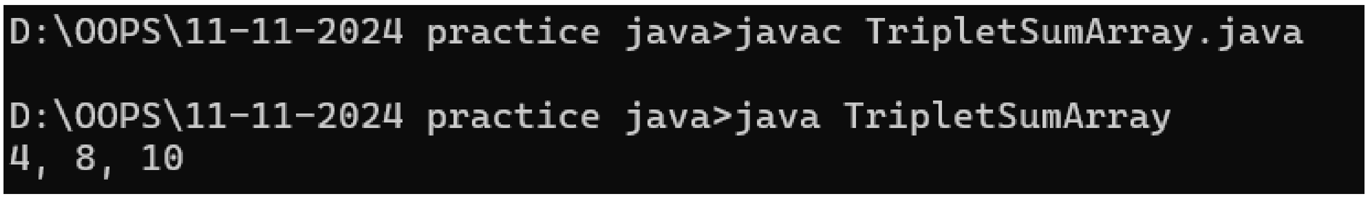
Input: *arr = {1, 2, 3, 4, 5}, sum = 9*
Output: *5, 3, 1*
Explanation: *There is a triplet (5, 3 and 1) present in the array whose sum is 9.*

Input: *arr = {2, 10, 12, 4, 8}, sum = 9*
Output: *No Triplet*
Explanation: *We do not print in this case and return false.*

SOLUTION:

```
import java.util.*;

public class TripletSumArray {

    static boolean find3Numbers(int[] arr, int sum) {

        int n = arr.length;

        Arrays.sort(arr);

        for (int i = 0; i < n - 2; i++) {

            int l = i + 1;

            int r = n - 1;

            while (l < r) {

                int curr_sum = arr[i] + arr[l] + arr[r];

                if (curr_sum == sum) {

                    System.out.println(arr[i] + ", " + arr[l] + ", " + arr[r]);

                    return true;

                } else if (curr_sum < sum) {

                    l++;

                } else {

                    r--;

                }

            }

        }

        return false;
    }
    public static void main(String[] args) {
        int[] arr = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;
        find3Numbers(arr, sum);
    }
}
```

OUTPUT:



```
D:\OOPS\11-11-2024 practice java>javac TripletSumArray.java

D:\OOPS\11-11-2024 practice java>java TripletSumArray
4, 8, 10
```

Time Complexity : O(N^2)

Space Complexity : O(1)