PRACTICE QUESTIONS

1. 0/1 Knapsack Problem

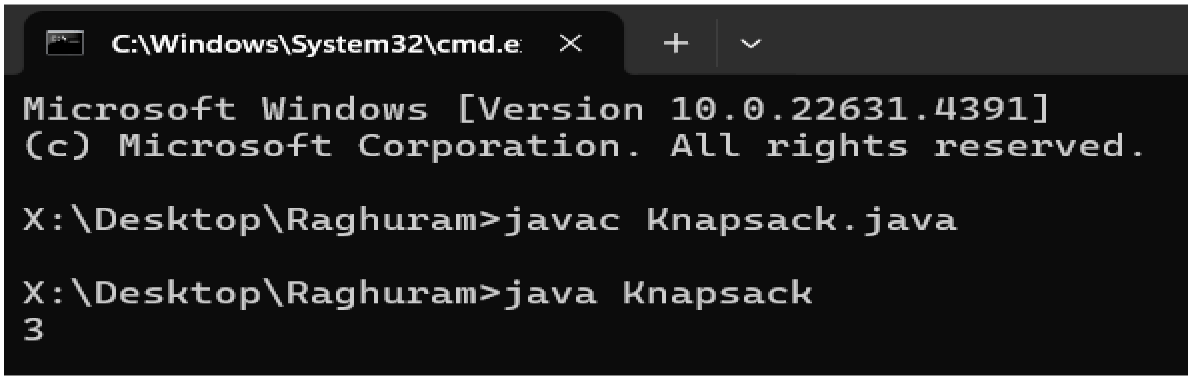
Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3
Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible 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

CODE:

```
public class Knapsack {
    public static int knapsack(int[] profit, int[] weight, int N, int W) {
        int[][] dp = new int[N + 1][W + 1];
        for (int i = 1; i <= N; i++) {
            for (int w = 1; w <= W; w++) {
                if (weight[i - 1] <= w) {
                    dp[i][w] = Math.max(profit[i - 1] + dp[i - 1][w - weight[i - 1]], dp[i -
1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[N][W];
    }
    public static void main(String[] args) {
        int N = 3;
        int W = 4;
        int[] profit = {1, 2, 3};
        int[] weight = {4, 5, 1};
        System.out.println(knapsack(profit, weight, N, W));
    }
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserved.

X:\Desktop\Raghuram>javac Knapsack.java

X:\Desktop\Raghuram>java Knapsack
3
```

TIME COMPLEXITY: O(NW)

---

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

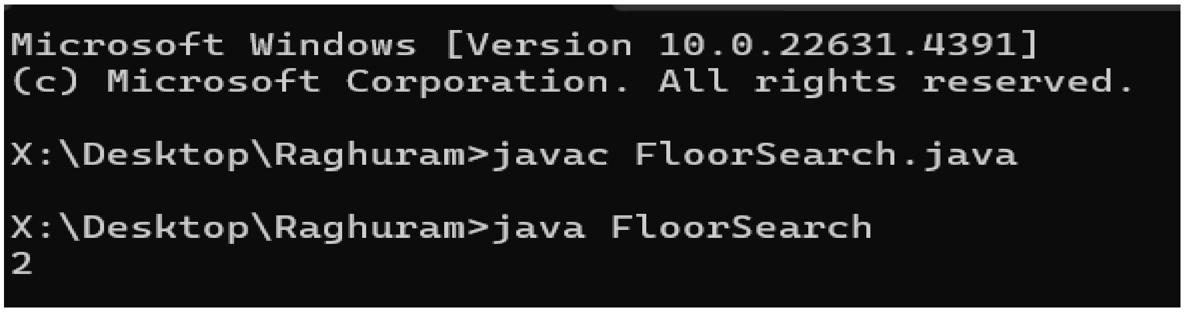
Examples:
Input: arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 5
Output: 2
Explanation: 2 is the largest element in arr[] smaller than 5

Input: arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 20
Output: 19
Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 0
Output : -1

CODE:

```
public class FloorSearch {
    public static int findFloor(int[] arr, int x) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == x) {
                return arr[mid];
            } else if (arr[mid] < x) {
                result = arr[mid];
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 8, 10, 10, 12, 19};
        System.out.println(findFloor(arr, 5));
    }
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserved.

X:\Desktop\Raghuram>javac FloorSearch.java

X:\Desktop\Raghuram>java FloorSearch
2
```

TIME COMPLEXITY: O(log N)

---

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

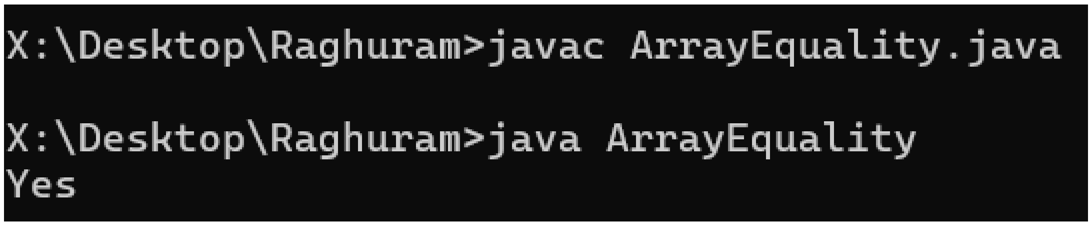
Input: arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}
Output: Yes

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

CODE:

```
import java.util.HashMap;
public class ArrayEquality {
    public static boolean areArraysEqual(int[] arr1, int[] arr2) {
        if (arr1.length != arr2.length) return false;
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int num : arr1) {
            map.put(num, map.getOrDefault(num, 0) + 1);
        }
        for (int num : arr2) {
            if (!map.containsKey(num) || map.get(num) == 0) return false;
            map.put(num, map.get(num) - 1);
        }

        return true;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 5, 4, 0};
        int[] arr2 = {2, 4, 5, 0, 1};
        System.out.println(areArraysEqual(arr1, arr2) ? "Yes" : "No");
    }
}
```

OUTPUT:



```
X:\Desktop\Raghuram>javac ArrayEquality.java

X:\Desktop\Raghuram>java ArrayEquality
Yes
```

TIME COMPLEXITY:O(n)

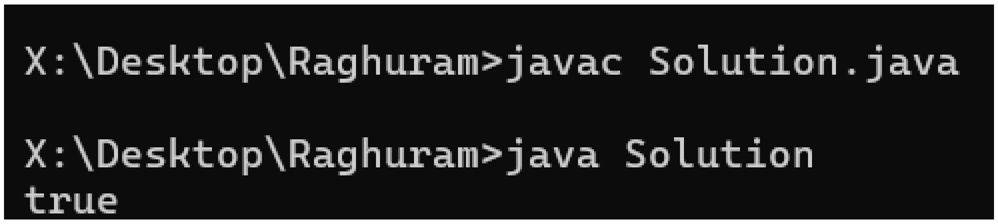
4. Palindrome Linked List

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.

CODE:
```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) { this.val = val; }
}
public class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null || head.next == null) return true;

        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        ListNode prev = null;
        while (slow != null) {
            ListNode nextNode = slow.next;
            slow.next = prev;
            prev = slow;
            slow = nextNode;
        }
        ListNode left = head, right = prev;
        while (right != null) {
            if (left.val != right.val) return false;
            left = left.next;
            right = right.next;
        }
        return true;
    }
    public static void main(String[] args) {
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(2);
        head1.next.next = new ListNode(1);
        head1.next.next.next = new ListNode(1);
        head1.next.next.next.next = new ListNode(2);
        head1.next.next.next.next.next = new ListNode(1);

        ListNode head2 = new ListNode(1);
        head2.next = new ListNode(2);
        head2.next.next = new ListNode(3);
        head2.next.next.next = new ListNode(4);

        Solution solution = new Solution();
        System.out.println(solution.isPalindrome(head1)); // Output: true
    }
}
```

OUTPUT:



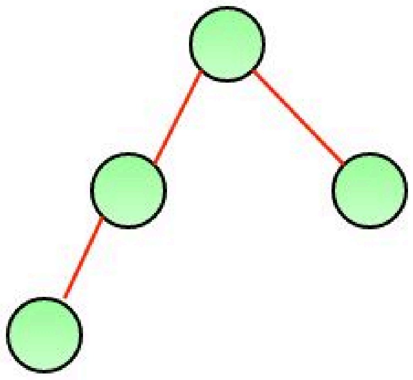
```
X:\Desktop\Raghuram>javac Solution.java

X:\Desktop\Raghuram>java Solution
true
```

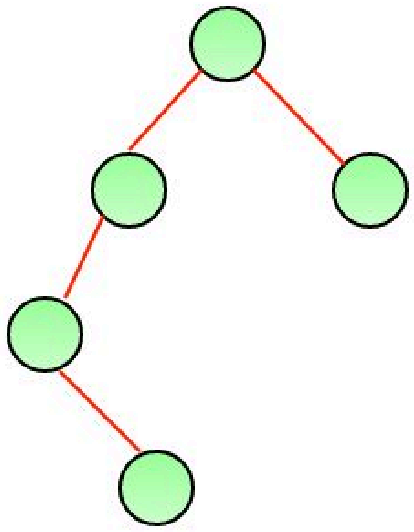
TIME COMPLEXITY: O(N)

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.

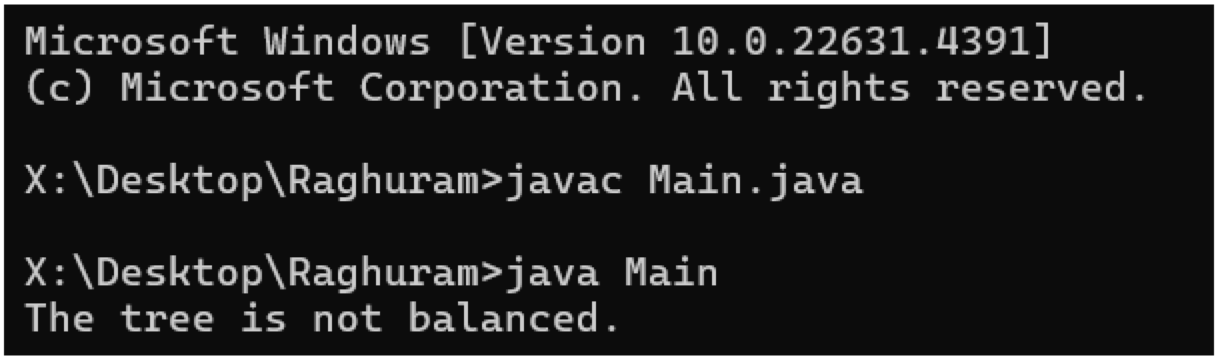

A height balanced tree


Not a height balanced tree

CODE:

```
import java.lang.Math;
class TreeNode {
    int value;
    TreeNode left;
    TreeNode right;
    TreeNode(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

class Solution {
    public boolean isBalanced(TreeNode root) {
        return calculateHeight(root) != -1;
    }
    public int calculateHeight(TreeNode root) {
        if (root == null) return 0;
        int leftHeight = calculateHeight(root.left);
        if (leftHeight == -1) return -1;

        int rightHeight = calculateHeight(root.right);
        if (rightHeight == -1) return -1;

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

        return Math.max(leftHeight, rightHeight) + 1;
    }
}
public class Main {
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.left.right.right = new TreeNode(6);
        root.left.right.right.right = new TreeNode(7);

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

OUTPUT:



```
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserved.

X:\Desktop\Raghuram>javac Main.java

X:\Desktop\Raghuram>java Main
The tree is not 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.

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.

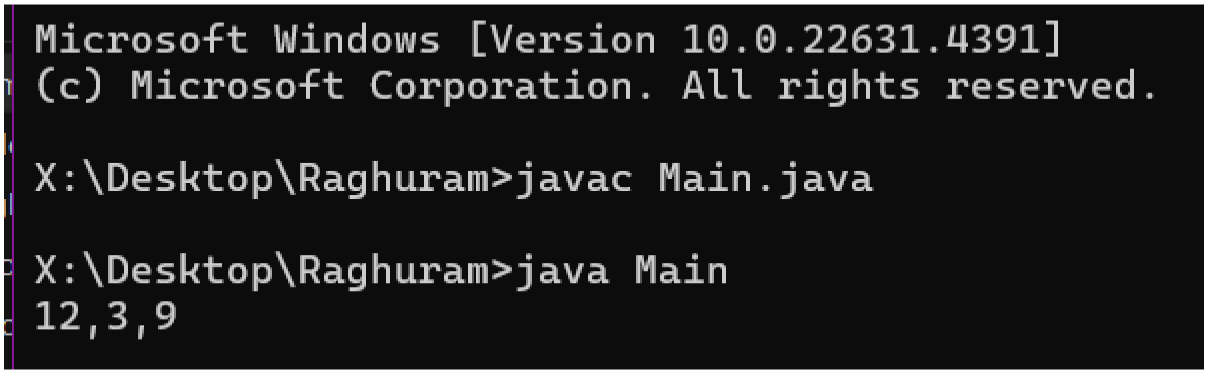
Input: arr = {1, 2, 3, 4, 5}, sum = 9
Output: 5, 3, 1 Explanation: There is a triplet (5, 3 and 1) present in the array whose sum is 9.

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

CODE:

```
import java.util.Arrays;

class Solution {

    public boolean findTriplet(int[] arr, int sum) {
        Arrays.sort(arr);
        for (int i = 0; i < arr.length - 2; i++) {
            int left = i + 1;
            int right = arr.length - 1;

            while (left < right) {
                int currentSum = arr[i] + arr[left] + arr[right];

                if (currentSum == sum) {
                    System.out.println(arr[i] + ", " + arr[left] + ", " + arr[right]);
                    return true;
                } else if (currentSum < sum) {
                    left++;
                } else {
                    right--;
                }
            }
        }

        System.out.println("No Triplet");
        return false;
    }
}

public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();

        int[] arr1 = {12, 3, 4, 1, 6, 9};
        int sum1 = 24;
        solution.findTriplet(arr1, sum1);

    }
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserved.

X:\Desktop\Raghuram>javac Main.java

X:\Desktop\Raghuram>java Main
12,3,9
```

TIME COMPLEXITY: O(N log N)