**NAME: SRINTHI JAIKUMAR**

**DEPARTMENT:AIDS**

**DATE:11/11/24**

**0 - 1 Knapsack Problem**

Difficulty: **Medium**Accuracy: **31.76%**Submissions: **447K+**Points: **4**

You are given the weights and values of items, and you need to put these items in a knapsack of capacity **capacity** to achieve the maximum total value in the knapsack. Each item is available in only one quantity.

In other words, you are given two integer arrays **val[]** and **wt[]**, which represent the values and weights associated with items, respectively. You are also given an integer **capacity**, which represents the knapsack capacity. Your task is to find the maximum sum of values of a subset of val[] such that the sum of the weights of the corresponding subset is less than or equal to **capacity**. You cannot break an item; you must either pick the entire item or leave it (0-1 property).

**Examples :**

**Input:** capacity = 4, val[] = [1, 2, 3], wt[] = [4, 5, 1]   
**Output:** 3  
**Explanation:** Choose the last item, which weighs 1 unit and has a value of 3.

**Input:** capacity = 3, val[] = [1, 2, 3], wt[] = [4, 5, 6]   
**Output:** 0  
**Explanation:** Every item has a weight exceeding the knapsack's capacity (3).

**Input:** capacity = 5, val[] = [10, 40, 30, 50], wt[] = [5, 4, 6, 3]   
**Output:** 50  
**Explanation:** Choose the second item (value 40, weight 4) and the fourth item (value 50, weight 3) for a total weight of 7, which exceeds the capacity. Instead, pick the last item (value 50, weight 3) for a total value of 50.

**Expected Time Complexity:** O(n\*capacity).  
**Expected Auxiliary Space:** O(n\*capacity)

**Constraints:**  
2 ≤ val.size() = wt.size() ≤ 103  
1 ≤ capacity ≤ 103  
1 ≤ val[i] ≤ 103  
1 ≤ wt[i] ≤ 103

Try more examples

**Company Tags**

[**Flipkart**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Flipkart)[**Morgan Stanley**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Morgan%20Stanley)[**Amazon**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Amazon)[**Microsoft**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Microsoft)[**Snapdeal**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Snapdeal)[**Oracle**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Oracle)[**Payu**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Payu)[**Visa**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Visa)[**Directi**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Directi)[**GreyOrange**](https://www.geeksforgeeks.org/explore/?company%5b%5d=GreyOrange)[**Mobicip**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Mobicip)

class Solution {  
    static int helper(int capacity, int val[], int wt[], int maxSum, boolean visited[]) {  
        if (capacity <= 0) {  
            return maxSum;  
        }  
        int currentMax = maxSum;  
          
        for (int i = 0; i < val.length; i++) {  
            if (!visited[i] && capacity - wt[i] >= 0) {  
                visited[i] = true;    
                currentMax = Math.max(currentMax, helper(capacity - wt[i], val, wt, maxSum + val[i], visited));  
                visited[i] = false;    
            }  
        }  
        return currentMax;  
    }

    static int knapSack(int capacity, int val[], int wt[]) {  
        int maxSum = 0;  
        boolean[] visited = new boolean[val.length];  
        return helper(capacity, val, wt, maxSum, visited);  
    }  
}

<https://ideone.com/eR9VhG>

**Time Complexity**

**O(n\*w**

**Floor in a Sorted Array**

Difficulty: **Easy**Accuracy: **33.75%**Submissions: **372K+**Points: **2**

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.

**Input:** arr[] = [1, 2, 8, 10, 11, 12, 19], k = 5

**Output:** 1

**Explanation:** Largest Number less than 5 is 2 , whose index is 1.

**Input:** arr[] = [1, 2, 8], k = 1

**Output:** 0

**Explanation:** Largest Number less than or equal to 1 is 1 , whose index is 0.

**Constraints:**  
1 ≤ arr.size() ≤ 106  
1 ≤ arr[i] ≤ 106  
0 ≤ k ≤arr[n-1]

Try more examples

**Company Tags**

[**Amazon**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Amazon)

public class Main {

public static void main(String[] args) {

int[] arr = {1, 2, 8, 10, 11, 12, 19};

int k = 0;

int floorIndex = Solution.findFloor(arr, k);

System.out.println("Floor index of " + k + " = " + floorIndex);

}

}

class Solution {

static int findFloor(int[] arr, int k) {

int low = 0;

int high = arr.length - 1;

int ans = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] <= k) {

ans = mid;

low = mid + 1;

} else {

high = mid - 1;

}

}

return ans;

}

}  
  
<https://ideone.com/A11roV>

**Time Complexity**

**O(n)**

**Check Equal Arrays**

Difficulty: **Basic**Accuracy: **42.18%**Submissions: **367K+**Points: **1**

Given two arrays **arr1** and **arr2**of equal size, the task is to find whether the given arrays are equal. Two arrays are said to be equal if both contain the same set of elements, arrangements (or permutations) of elements may be different though.  
**Note:** If there are repetitions, then counts of repeated elements must also be the same for two arrays to be equal.

**Examples:**

**Input:** arr1[] = [1, 2, 5, 4, 0], arr2[] = [2, 4, 5, 0, 1]

**Output:** true

**Explanation:** Both the array can be rearranged to [0,1,2,4,5]

**Input:** arr1[] = [1, 2, 5], arr2[] = [2, 4, 15]

**Output:** false

**Explanation:** arr1[] and arr2[] have only one common value.

**Expected Time Complexity**: O(n)  
**Expected Space Complexity**: O(n)

**Constraints:**  
1<= arr1.size, arr2.size<=107  
0<=arr1[], arr2[]<=109

Try more examples

**Company Tags**

[**Goldman Sachs**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Goldman%20Sachs)

import java.util.Arrays;

class Solution {

// Function to check if two arrays are equal or not.

public static boolean check(int[] arr1, int[] arr2) {

// Check if lengths are equal

if (arr1.length != arr2.length) {

return false;

}

// Sort both arrays

Arrays.sort(arr1);

Arrays.sort(arr2);

// Compare elements one by one

for (int i = 0; i < arr1.length; i++) {

if (arr1[i] != arr2[i]) {

return false;

}

}

return true;

}

}  
public class Main {

public static void main(String[] args) {

int[] arr1 = {1, 2, 5, 4, 0};

int[] arr2 = {2, 4, 5, 0, 1};

boolean areEqual = Solution.check(arr1, arr2);

System.out.println("Are the two arrays equal? " + areEqual);

}

}

<https://ideone.com/rUHiYd>

**Time Complexity**

**O(log n)**

**Palindrome Linked List**

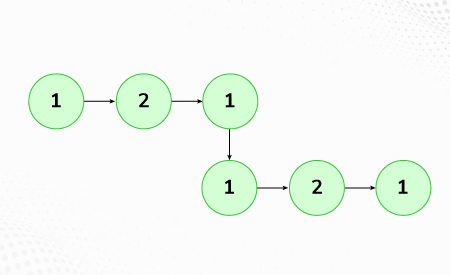
Difficulty: **Medium**Accuracy: **41.48%**Submissions: **345K+**Points: **4**

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

**Examples:**

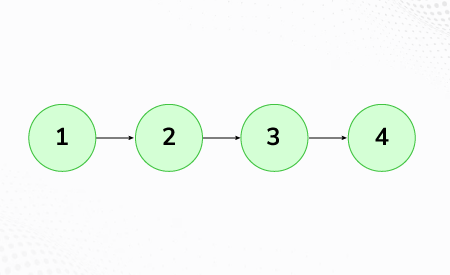
**Input:** LinkedList: 1->2->1->1->2->1

**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:** LinkedList: 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.  


**Expected Time Complexity**: O(n)  
**Expected Auxiliary Space**: O(1) 

**Constraints:**  
1 <= number of nodes <= 1051 ≤ node->data ≤ 103

Try more examples

**Company Tags**

[**Accolite**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Accolite)[**Amazon**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Amazon)[**Microsoft**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Microsoft)[**Snapdeal**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Snapdeal)[**MakeMyTrip**](https://www.geeksforgeeks.org/explore/?company%5b%5d=MakeMyTrip)[**Adobe**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Adobe)[**Yodlee Infotech**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Yodlee%20Infotech)[**KLA Tencor**](https://www.geeksforgeeks.org/explore/?company%5b%5d=KLA%20Tencor)[**Kritikal Solutions**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Kritikal%20Solutions)

class Solution {

// Function to check whether the list is palindrome.

boolean isPalindrome(Node head) {

// Your code here

Stack<Integer> stack =new Stack();

Node curr=head;

while(curr != null)

{

stack.push(curr.data);

curr=curr.next;

}

curr=head;

while(curr != null){

if(curr.data !=stack.pop()){

return false;

}

curr=curr.next;

}

return true;

}

**Time Complexity**

**O(n)**

### } **Balanced Tree Check**

Difficulty: **Easy**Accuracy: **43.15%**Submissions: **320K+**Points: **2**

Given a binary tree, find if it is height balanced or not.  A tree is height balanced if difference between heights of left and right subtrees is **not more than one** for all nodes of tree.

**Examples:**

**Input:**

      1

   /

   2

   \

    3

**Output:** 0

**Explanation:** The max difference in height of left subtree and right subtree is 2, which is greater than 1. Hence unbalanced

**Input:**

       10

    /   \

   20   30

  /   \

40   60

**Output:** 1

**Explanation:** The max difference in height of left subtree and right subtree is 1. Hence balanced.

**Constraints:**  
1 <= Number of nodes <= 105  
1 <= Data of a node <= 109

**Expected time complexity:**O(N)  
**Expected auxiliary space:**O(h) , where h = height of tree

Try more examples

**Company Tags**

[**Amazon**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Amazon)[**Microsoft**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Microsoft)[**Walmart**](https://www.geeksforgeeks.org/explore/?company%5b%5d=Walmart)

class Tree

{

static int check(Node node){

if(node == null) return 0;

int l=check(node.left);

int r=check(node.right);

if(l==-1) return -1;

if(r==-1) return -1;

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

return -1;

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

}

boolean isBalanced(Node root)

{

if(check(root)>0)

return true;

return false;

}

**Time Complexity**

**O(n)**

### } **Triplet Sum in Array**

Difficulty: **Medium**Accuracy: **35.0%**Submissions: **305K+**Points: **4**

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

**Examples**

**Input**:n = 6, x = 13, arr[] = [1,4,45,6,10,8]

**Output**: 1

**Explanation**: The triplet {1, 4, 8} in the array sums up to 13.

**Input**: n = 6, x = 10, arr[] = [1,2,4,3,6,7]

**Output**: 1

**Explanation**: Triplets {1,3,6} & {1,2,7} in the array sum to 10.

**Input**: n = 6, x = 24, arr[] = [40,20,10,3,6,7]

**Output**: 0

**Explanation**: There is no triplet with sum 24.

**Expected Time Complexity:**O(n2)  
**Expected Auxiliary Space:**O(1)

**Constraints:**  
1 ≤ n ≤ 103  
1 ≤ arr[i] ≤ 105

import java.util.Arrays;

import java.util.ArrayList;

public class Tripletsum {

    public static void main(String[] args) {

        int[] array = { 0, 0, 0 };

        ArrayList<int[]> zeroSumTriplets = findZeroSumTriplets(array);

        System.out.println("Triplet subarrays with sum 0:");

        if (zeroSumTriplets.isEmpty()) {

            System.out.println("[]");

        } else {

            for (int[] triplet : zeroSumTriplets) {

                System.out.println(Arrays.toString(triplet));

            }

        }

    }

    private static ArrayList<int[]> findZeroSumTriplets(int[] array) {

        ArrayList<int[]> zeroSumTriplets = new ArrayList<>();

        Arrays.sort(array);

        for (int i = 0; i < array.length - 2; i++) {

            if (i > 0 && array[i] == array[i - 1]) {

                continue; // Skip duplicates

            }

            int left = i + 1;

            int right = array.length - 1;

            while (left < right) {

                int sum = array[i] + array[left] + array[right];

                if (sum == 0) {

                    zeroSumTriplets.add(new int[] { array[i], array[left], array[right] });

                    while (left < right && array[left] == array[left + 1])

                        left++; // Skip duplicates

                    while (left < right && array[right] == array[right - 1])

                        right--; // Skip duplicates

                    left++;

                    right--;

                } else if (sum < 0) {

                    left++;

                } else {

                    right--;

                }

            }

        }

        return zeroSumTriplets;

    }

**Time Complexity**

**O(n\*\*2)**