**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

**Code:**

```
static int Height(BinTree node){
        if(node==null){return 0;}

        int lefth=Height(node.left);
        int righth=Height(node.right);

        if(lefth==-1 || righth==-1){return -1;}

        if(Math.abs(lefth-righth)>1){
                return -1;
        }

        return Math.max(lefth,righth)+1;
}
```

**TimeComplexity:O(n)**
**SpaceComplexity:O(n)**

## Check Equal Arrays

Difficulty: **Basic** Accuracy: **42.18%** Submissions: **366K+** 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.

**Code:**

```
  GNU nano 7.2
#include <bits/stdc++.h>
using namespace std;
bool isEquals(vector<int> a,vector<int> b){
        if(a.size()!=b.size())return false;
        unordered_map<int,int> hash1,hash2;
        for(int i=0;i<a.size();i++){
                hash1[a[i]]++;
                hash2[a[i]]+=0;
                hash2[b[i]]++;
                hash1[b[i]]+=0;
        }
        for(auto a:hash2){
                if(a.second!=hash1[a.first])return false;
        }
        return true;
}
```

**TimeComplexity: O(n)**
**SpaceComplexity:O(n)**

## Floor in a Sorted Array

Last Updated : 30 Oct, 2022

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.*

**Code:**

```
int findFloor(vector<int>& arr, int k) {
        int low=0,high=arr.size()-1;
        int ans=-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(arr[mid] > k){
                high=mid-1;
                ans=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        return ans;
}
```

**TimeComplexity: O(log n)**
**SpaceComplexity:O(1)**

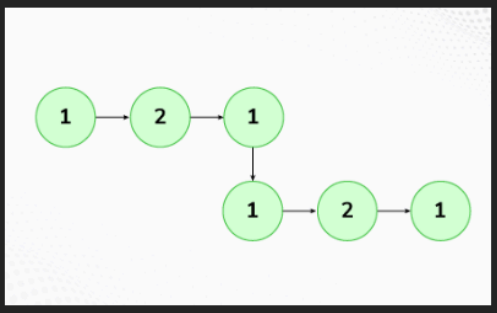
## Palindrome Linked List

Difficulty: **Medium** Accuracy: **41.48%** Submissions: **344K+** 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.




**Code:**

```
bool isPalindrome(Node *head) {
        stack<int> st;
        Node *temp=head;
        while(temp){
            st.push(temp->data);
            temp=temp->next;
        }
        temp=head;
        while(temp){
            if(st.top()!=temp->data){
                return false;
            }
            st.pop();
            temp=temp->next;
        }
        return true;
}
```

**TimeComplexity:O(n)**
**SpaceComplexity:O(n)**

## 0 - 1 Knapsack Problem

Difficulty: **Medium** Accuracy: **31.76%** Submissions: **446K+** 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).

```cpp
int KnapSack(int n,int w,vector<int> &val,vector<int> &wt){
        if(n==0){
            if(wt[0]<=w){
                return val[0];
            }
            else{
                return 0;
            }
        }
        int notTake=KnapSack(n-1,w,val,wt);
        int Take=INT_MIN;
        if(wt[n]<=w){
            Take=val[n]+KnapSack(n-1,w-wt[n],val,wt);
        }
        return max(Take,notTake);
}
```

**TimeComplexity:O(2**n)**
**SpaceComplexity: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.

```
bool find3Numbers(int arr[], int n, int x) {
        for(int i=0;i<n;i++){
            unordered_set<int> hashset;
            for(int j=i+1;j<n;j++){
                int num=(x-arr[i]-arr[j]);
                if(hashset.find(num)!=hashset.end()){
                    return true;
                }
                hashset.insert(arr[j]);
            }
        }
        return false;
}
```

**TimeComplexity:O(n**2)**
**SpaceComplexity:O(n)**