

## 1.0-1 Knapsack Problem

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.

```
// } Driver Code Ends
class Solution {
public:
    int helper(int idx,int weight,vector<int>& val,vector<int>& wt,int& capacity,vector<vector<int>>& dp){
        if(weight>=capacity || idx>=wt.size()) return 0;
        if(dp[idx][weight]!=-1) return dp[idx][weight];
        int take=0;
        if(weight+wt[idx]<=capacity) take=val[idx]+helper(idx+1,weight+wt[idx],val,wt,capacity,dp);
        int not_take=helper(idx+1,weight,val,wt,capacity,dp);
        return dp[idx][weight]=max(take,not_take);
    }
    int knapSack(int capacity, vector<int> &val, vector<int> &wt) {
        int n=wt.size();
        vector<vector<int>> dp(n,vector<int>(capacity+1,-1));
        return helper(0,0,val,wt,capacity,dp);
    }
};
// } Driver Code Ends
```

## Compilation Completed

For Input:  

4

1 2 3

4 5 1

Your Output:

3

Expected Output:

3

**Time Complexity:**  $O(n \times \text{capacity})$

**Space Complexity:**  $O(n \times \text{capacity})$  for memorization plus  $O(n)$  for the recursive stack.

## 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`.

**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.

```
// Driver Code Ends
class Solution {
public:
    int findFloor(vector<int>& nums, int target) {
        int left=0;
        int right=nums.size()-1;
        while(left<=right){
            if(target>=nums[right]) return right;
            int mid=(left+right)/2;
            if(target==nums[mid]) return mid;
            if(mid>0 && target>=nums[mid-1] && target<nums[mid]) return mid-1;
            if(target<nums[mid]) right=mid-1;
            else left=mid+1;
        }
        return -1;
    }
};
```

## Compilation Completed

For Input:  

1 2 8 10 11 12 19

0

Your Output:

-1

Expected Output:

-1

**Time Complexity :**  $O(\log n)$

**Space Complexity :**  $O(1)$

### 3. Check equal arrays

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.

**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.

```
class Solution {
public:
    bool check(vector<int>& arr1, vector<int>& arr2) {
        map<int,int> map;
        for(auto i:arr1) map[i]++;
        for(auto i:arr2){
            map[i]--;
            if(map[i]<0) return false;
        }
        for(auto i:map){
            if(i.second!=0) return false;
        }
        return true;
    }
};
```

## Compilation Completed

For Input:  

1 2 5 4 0

2 4 5 0 1

Your Output:

true

Expected Output:

true

**Time Complexity:**  $O(n)$

**Space Complexity :**  $O(n)$

#### 4. Palindrome Linked List

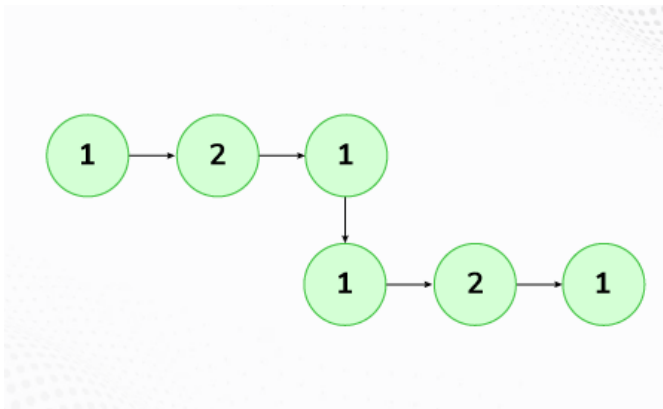
Given a singly linked list of integers. The task is to check if the given linked list is palindrome or not.

##### Examples:

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

**Output:** true

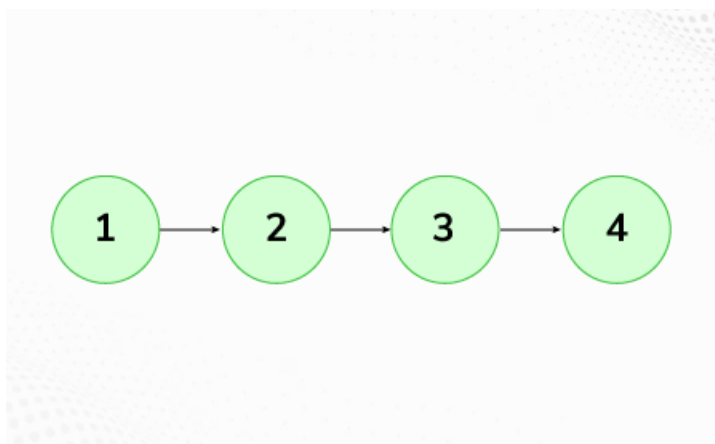
**Explanation:** The given linked list is 1->2->1->1->2->1, which is a palindrome and Hence, the output is true.



**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.




```

class Solution {
public:
    ListNode* rev(ListNode* root){
        ListNode* prev=NULL;
        ListNode* crr=root;
        while(crr!=NULL){
            ListNode* nextnode=crr->next;
            crr->next=prev;
            prev=crr;
            crr=nextnode;
        }
        return prev;
    }
    bool isPalindrome(Node *head) {
        ListNode* slow=head,*fast=head,*tem=head;
        while(fast){
            slow=slow->next;
            fast=fast->next;
            if(fast) fast=fast->next;
        }
        slow=rev(slow);
        while(slow){
            if(tem->val!=slow->val) return false;
            tem=tem->next;
            slow=slow->next;
        }
        return true;
    }
};

```

## Compilation Completed

For Input:  

1 2 1 1 2 1

Your Output:

true

Expected Output:

true

**Time Complexity:**  $O(n)$

**Space Complexity :**  $O(1)$



## 5.Balanced Tree Check

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.

```

class Solution{
public:
    int helper2(Node* root){
        if(!root) return 0;
        int left=1+helper2(root->left);
        int right=1+helper2(root->right);
        return max(left,right);
    }
    bool helper(Node* root){
        if(!root) return true;
        int left=helper2(root->left);
        int right=helper2(root->right);
        return abs(left-right)<=1 && helper(root->left) && helper(root->right);
    }
    bool isBalanced(Node *root)
    {
        return helper(root);
    }
};

```

### Compilation Completed

For Input:  

1 2 N N 3

Your Output:

0

Expected Output:

0

**Time Complexity :**  $O(n)$ ;

**Space Complexity : Best case (balanced tree):**  $O(\log N)$  space complexity.

**Worst case (unbalanced tree):**  $O(N)$  space complexity.

## 6.Triplet Sum in Array

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.

```
class Solution {
public:
    bool find3Numbers(int arr[], int n, int x) {
        std::sort(arr, arr+n);
        for(int l=0; l<n-2; l++){
            int left=l+1;
            int right=n-1;
            int target=x-arr[l];
            while(left<right){
                if(arr[left]+arr[right]==target) return 1;

                if(arr[left]+arr[right]<target) left++;
                else right--;
            }
        }
        return 0;
    }
};
// } Driver Code Ends
```

## Compilation Completed

For Input:  

6 13

1 4 45 6 10 8

Your Output:

1

Expected Output:

1

**Time Complexity :**  $O(n^2)$

**Space Complexity :**  $O(1)$