DSA - DAY 1

https://www.geeksforgeeks.org/problems/0-1-knapsack-problem0945/1

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;


// } Driver Code Ends
class Solution {
  public:
    // Function to return max value that can be put in knapsack
    int f(int i,int j,vector<int>& val,vector<int>& wt){
    //  cout<<i<<" "<<j<<endl;
        if(i==val.size()  ){
            return 0;
        }

        int np = f(i+1,j,val,wt);
        int p = 0;
        if(wt[i]<=j){
                 p = val[i] + f(i+1,j-wt[i],val,wt);
        }
        // cout<<p<<" "<<np<<" "<<i<<" "<<" "<<j;
        return max(np,p);
    }
    int knapSack(int c, vector<int> &val, vector<int> &wt) {
        // code here
        int n=val.size();
        vector<vector<int>> dp(n+1,vector<int>(c+1,0));
        for(int i=n-1;i>=0;i--){
            for(int j=0;j<=c;j++){
                int np = dp[i+1][j];
                int  p =0;
                if(wt[i]<=j){
                    p =val[i] + dp[i+1][j-wt[i]];
                }
                dp[i][j]=max(p,np);
            }
        }
        return dp[0][c];
    }

};
```

https://www.geeksforgeeks.org/problems/floor-in-a-sorted-array-1587115620/1

```
class Solution {
  public:

    int findFloor(vector<int>& arr, int k) {

        // Your code here
        int  l=0,h=arr.size()-1,ans = h;
        while(l<=h){

            int mid = (l+h)/2;
            if(arr[mid]>k){
                ans = mid;
                h=mid-1;
            }
            else{
                l=mid+1;
            }

        }
        if(ans==0  || arr[ans-1]>k  ) return -1;
        return ans-1;
    }
};
```

https://www.geeksforgeeks.org/problems/check-if-two-arrays-are-equal-or-not3847/1

```
// User function template for C++

class Solution {
  public:
    // Function to check if two arrays are equal or not.
    bool check(vector<int>& arr1, vector<int>& arr2) {
        // code here
        unordered_map<int,int> mp;
        if(arr1.size()!=arr2.size()) return false;
        for(int i=0;i<arr1.size();i++){
            mp[arr1[i]]++;
            mp[arr2[i]]--;
        }
        for(pair<int,int> i:mp){
            if(i.second !=0 ) return false;
        }
        return true;
    }
};
```

https://www.geeksforgeeks.org/problems/check-if-linked-list-is-pallindrome/1

```
/*
struct Node {
  int data;
  struct Node *next;
  Node(int x) {
    data = x;
    next = NULL;
  }
};
*/

class Solution{
  public:
    //Function to check whether the list is palindrome.
    bool isPalindrome(Node *head)
    {
        //Your code here
        stack<int>mystack;
        Node *start=head;
        while(start){
            mystack.push(start->data);
            start=start->next;
        }
        while(!mystack.empty()){
            if(mystack.top()==head->data){
                head=head->next;
                mystack.pop();

            }
            else{
                return false;
            }
        }
        return true;
    }
};
```

https://www.geeksforgeeks.org/problems/triplet-sum-in-array-1587115621/1

```
class Solution {
  public:

    // Should return true if there exists a triplet in the
    // array arr[] which sums to x. Otherwise false
    bool find3Numbers(int arr[], int n, int x) {
        sort(arr,arr+n);
        for(int i=0;i<n-2;i++){
            if(i > 0 && arr[i]==arr[i-1]) continue;
            int j=i+1,k=n-1;
            while(j<k){
                int s = arr[i]+arr[j]+arr[k];
                if(s==x) return true;
                else if (s>x) k--;
                else j++;
            }

        }
        return false;


    }
};
```

https://www.geeksforgeeks.org/problems/check-for-balanced-tree/1

```
/* A binary tree node structure

struct Node
{
    int data;
    struct Node* left;
    struct Node* right;

    Node(int x){
        data = x;
        left = right = NULL;
    }
};
 */

class Solution{
    public:
    //Function to check whether a binary tree is balanced or not
    bool ans = true;
    int solve(Node* root){
        if (root==NULL) return 0;
        int left = solve(root->left);
        int right =solve(root->right);
        if(abs(left-right) > 1) ans = false;
        return max(left,right) + 1;
    }
    bool isBalanced(Node *root)
    {
        //  Your Code here
        solve(root);
        return ans;
    }
};
```