**############################################################**

**1. Print first n Fibonacci Numbers**

Given a number **N,** find the first N Fibonacci numbers. The first two number of the series are 1 and 1.

**Example 1:**

**Input:**

N = 5

**Output:** 1 1 2 3 5

**Example 2:**

**Input:**

N = 7

**Output:** 1 1 2 3 5 8 13

**Your Task:**  
Your task is to complete **printFibb()**which takes single argument N and returns a list of first N Fibonacci numbers.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(N).  
**Note:**This space is used to store and return the answer for printing purpose.

**Constraints:**  
1<= N <=84

void fib(vector<long long> &ans, long long a, long long b, int n)

{ if(n == 0)

return;

ans.push\_back(a+b);

fib(ans, b, a+b, n-1);

}

vector<long long> printFibb(int n)

{ vector<long long> ans;

ans.push\_back(1);

if(n == 1)

return ans;

ans.push\_back(1);

if(n == 2)

return ans;

fib(ans, 1, 1, n-2);

return ans;

}

**############################################################**

**2. 0 - 1 Knapsack Problem**

You are given weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. Note that we have only **one quantity of each item**.  
In other words, given two integer arrays **val[0..N-1]** and **wt[0..N-1]** which represent values and weights associated with **N** items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of **val[]** such that sum of the weights of this subset is smaller than or equal to **W.** You cannot break an item, **either pick the complete item or don’t pick it (0-1 property)**.

**Example 1:**

**Input:**

N = 3

W = 4

values[] = {1,2,3}

weight[] = {4,5,1}

**Output:** 3

**Example 2:**

**Input:**

N = 3

W = 3

values[] = {1,2,3}

weight[] = {4,5,6}

**Output:** 0

**Your Task:**  
Complete the function **knapSack()** which takes maximum capacity W, weight array wt[], value array val[], and the number of items n as a parameter and returns the **maximum possible** value you can get.

**Expected Time Complexity:** O(N\*W).  
**Expected Auxiliary Space:** O(N\*W)

**Constraints:**  
1 ≤ N ≤ 1000  
1 ≤ W ≤ 1000  
1 ≤ wt[i] ≤ 1000  
1 ≤ v[i] ≤ 1000

int knapSack(int W, int wt[], int val[], int n)

{

int dp[n+1][W+1];

for(int i=0; i<=n; i++)

for(int j=0; j<=W; j++)

dp[i][j] = 0;

for(int i=1; i<=n; i++)

{ for(int j=1; j<=W; j++)

{ if(wt[i-1] <= j)

dp[i][j] = max(val[i-1] + dp[i-1][j-wt[i-1]], dp[i-1][j]);

else

dp[i][j] = dp[i-1][j];

}

}

return dp[n][W];

}

**############################################################**

**3. Stickler Thief**

Stickler the thief wants to loot money from a societyhaving**n** houses in a single line. He is a weird person and follows a certain **rule**when looting the houses. According to the rule, he will **never loot two consecutive houses**. At the same time, he wants to **maximize**the amount he **loots**. The thief knows which house has what amount of money but is unable to come up with an optimal looting strategy. He asks for your help to **find the maximum money he can get** if he strictly **follows**the **rule**. Each house has **a[i] amount of money** present in it.

**Example 1:**

**Input:**

n = 6

a[] = {5,5,10,100,10,5}

**Output:** 110

**Explanation:** 5+100+5=110

**Example 2:**

**Input:**

n = 3

a[] = {1,2,3}

**Output:** 4

**Explanation:** 1+3=4

**Your Task:**  
Complete the function **FindMaxSum()**which takes an array **arr[]** and **n** as input which returns the maximum money he can get following the rules

**Expected Time Complexity:** O(N).  
**Expected Space Complexity:** O(N).

**Constraints:**  
1 ≤ n ≤ 104  
1 ≤ a[i] ≤ 104

int FindMaxSum(int arr[], int n)

{

int sum1 = arr[0], sum2 = 0, cur = 0;

for(int i=1; i<n; i++)

{ if(sum1 > sum2)

cur = sum1;

else

cur = sum2;

sum1 = sum2 + arr[i];

sum2 = cur;

}

return max(sum1, sum2);

}

**############################################################**

**4. Coin Change - Number of ways**

You have an infinite supply of coins, each having some value. Find out the number of ways to use the coins to sum-up to a certain required value.

**Example 1:**

**Input:**

value = 4

numberOfCoins = 3

coins[] = {1,2,3}

**Output:** 4

**Explanation:** We need to make the change

for value = 4. The denominations are

{1,2,3} Change for 4 can be made:

1+1+1+1

1+1+2

1+3

2+2

So, as it is evident, we can do this in

4 ways.

**Example 2:**

**Input:**

value = 10

numberOfCoins = 4

coins[] = {2,5,3,6}

**Output:** 5

**Your Task:**  
This is a function problem where you've to complete the funcion **numberOfWays ()**. You are given an amount denoted by **value**. You are also given an array of coins. The arraycontains the **denominations**of the given coins. You need to return the **number of ways** you can make the change for **value**using the coins of given denominations. Also, keep in mind that you have an **infinite supply** of coins.  
**Note:** Try not to editing the part of the code provided to you in the function. Just complete the function as it has been described.

**Expected Time Complexity:**O(Number of Coins \* Value).  
**Expected Auxiliary Space:**O(Value).

**Constraints:**  
1 <= value <= 103  
1 <= numberOfCoins <= 103  
1 <= coinsi <= 1000

long long numberOfWays(int coins[],int n,int v)

{ long long dp[n+1][v+1];

for(int i=0; i<=n; i++)

dp[i][0] = 1;

for(int j=1; j<=v; j++)

dp[0][j] = 0;

for(int i=1; i<=n; i++)

{ for(int j=1; j<=v; j++)

{ if(coins[i-1] <= j)

dp[i][j] = dp[i][j-coins[i-1]] + dp[i-1][j];

else

dp[i][j] = dp[i-1][j];

}

}

return dp[n][v];

}

**############################################################**

**5. Coin Change - Minimum number of coins**

You are given an amount denoted by **value**. You are also given an array of coins. The **array**contains the  
**denominations**of the give coins. You need to find the **minimum number of coins**to make the change for **value**using the coins of given denominations. Also, keep in mind that you have **infinite supply** of the coins.

**Example 1:**

**Input:**

value = 5

numberOfCoins = 3

coins[] = {3,6,3}

**Output:** Not Possible

**Explanation:**We need to make the change for

value = 5 The denominations are {3,6,3}

It is certain that we cannot make 5 using

any of these coins.

**Example 2:**

**Input:**

value = 10

numberOfCoins = 4

coins[] = {2 5 3 6}

**Output:** 2

**Explanation:**We need to make the change for

value = 10 The denominations are {2,5,3,6}

We can use two 5 coins to make 10. So

minimum coins are 2.

**Your Task:**  
You only need to complete the function **minimumNumberOfCoins()**that take array of coins, size of array, and value as parameters. You need to return the minimum number of coins required. If it is not possible to make the exact value out of the given coin denominations, return -1 ("Not Possible" will be printed by the driver's code in this case).

**Expected Time Complexity:** O(number of coins \* value).  
**Expected Auxiliary Space:** O(value)

**Constraints:**  
1 <= value <= 103  
1 <= numberOfCoins <= 103  
1 <= coinsi <= 1000

long long minimumNumberOfCoins(int coins[],int n,int v)

{

long long dp[n+1][v+1];

for(int i=0; i<=n; i++)

dp[i][0] = 0;

for(int j=1; j<=v; j++)

dp[0][j] = INT\_MAX;

for(int i=1; i<=n; i++)

for(int j=1; j<=v; j++)

if(j >= coins[i-1])

dp[i][j] = min(dp[i-1][j], dp[i][j-coins[i-1]] + 1);

else

dp[i][j] = dp[i-1][j];

return (dp[n][v] == INT\_MAX)? -1: dp[n][v];

}

**############################################################**

**6. Count Palindromic Subsequences**

Given a string str of length N, you have to find number of palindromic subsequence (need not necessarily be distinct) which could be formed from the string str.  
Note: You have to return the answer module 109+7;

**Example 1:**

**Input:**

Str = "abcd"

**Output:**

4

**Explanation:**

palindromic subsequence are : "a" ,"b", "c" ,"d"

**Example 2:**

**Input:**

Str = "aab"

**Output:**

4

**Explanation:**

palindromic subsequence are :"a", "a", "b", "aa"

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **countPs()** which takes a string str as input parameter and returns the number of palindromic subsequence.

**Expected Time Complexity:** O(N\*N)  
**Expected Auxiliary Space:** O(N\*N)

**Constraints:**  
1<=length of string str <=1000

long long int countPS(string str) {

long long int mod = 1000000007;

int n = str.length();

long long int dp[n+1][n+1];

for(int i=0; i<=n; i++)

for(int j=0; j<=n; j++)

dp[i][j] = (i == j && i != 0)? 1: 0;

for(int k=2; k<=n; k++) {

for(int i=1, j=k; i<=n && j<=n; i++, j++) {

if(str[i-1] == str[j-1])

dp[i][j] = (dp[i][j-1]%mod + dp[i+1][j]%mod + 1)%mod;

else

dp[i][j] = (dp[i][j-1]%mod + dp[i+1][j]%mod - dp[i+1][j-1]%mod)%mod;

if(dp[i][j] < 0)

dp[i][j] += mod;

}

}

return dp[1][n]%mod;

}

**############################################################**

**7. Longest Increasing Subsequence**

Given an array of integers, find the **length**of the**longest (strictly) increasing subsequence** from the given array.

**Example 1:**

**Input:**

N = 16

A[]={0,8,4,12,2,10,6,14,1,9,5

  13,3,11,7,15}

**Output:** 6

**Explanation:**Longest increasing subsequence

0 2 6 9 13 15, which has length **6**

**Example 2:**

**Input:**

N = 6

A[] = {5,8,3,7,9,1}

**Output:** 3

**Explanation:**Longest increasing subsequence

5 7 9, with length **3**

**Your Task:**  
Complete the function **longestSubsequence()** which takes the input array and its size as input parameters and returns the **length**of the**longest increasing subsequence.**

**Expected Time Complexity** : O( N\*log(N) )  
**Expected Auxiliary Space**: O(N)

**Constraints:**  
1 ≤ N ≤ 105  
0 ≤ A[i] ≤ 106

int bsear(int dp[], int l, int r, int key) {

while(l < r) {

int m = l+(r-l)/2;

if(key <= dp[m])

r = m;

else

l = m + 1;

}

return r;

}

int longestSubsequence(int n, int a[]) {

int dp[n];

dp[0] = a[0];

int len = 1;

for(int i=1; i<n; i++) {

if(a[i] < dp[0])

dp[0] = a[i];

else if(a[i] > dp[len-1])

dp[len++] = a[i];

else

dp[bsear(dp, 0, len-1, a[i])] = a[i];

}

return len;

}

**############################################################**

**8. Egg Dropping Puzzle**

Suppose you have N eggs and you want to determine from which floor in a K-floor building you can drop an egg such that it doesn't break. You have to determine the minimum number of attempts you need in order find the critical floor in the worst case while using the best strategy.There are few rules given below.

* An egg that survives a fall can be used again.
* A broken egg must be discarded.
* The effect of a fall is the same for all eggs.
* If the egg doesn't break at a certain floor, it will not break at any floor below.
* If the eggs breaks at a certain floor, it will break at any floor above.

For more description on this problem see [wiki page](http://en.wikipedia.org/wiki/Dynamic_programming#Egg_dropping_puzzle)

**Example 1:**

**Input:**

N = 2, K = 10

**Output:** 4

**Example 2:**

**Input:**

N = 3, K = 5

**Output:** 3

**Your Task:**  
Complete the function **eggDrop()** which takes two positive integer N and K as input parameters and returns the minimum number of attempts you need in order to find the critical floor.

**Expected Time Complexity** : O(N\*K)  
**Expected Auxiliary Space**: O(N\*K)

**Constraints:**  
1<=N<=200  
1<=K<=200

int eggDrop(int e, int f) {

int dp[e+1][f+1];

for(int i=0; i<=e; i++)

dp[i][0] = 0;

for(int j=0; j<=f; j++)

dp[0][j] = 0;

for(int i=1; i<=e; i++)

dp[i][1] = 1;

for(int j=2; j<=f; j++)

dp[1][j] = j;

for(int i=2; i<=e; i++) {

for(int j=2; j<=f; j++) {

int temp, val = INT\_MAX;

for(int l=0, r=j-1; l<=r; l++, r--){

temp = max(dp[i-1][l], dp[i][r]) + 1;

val = min(val, temp);

}

dp[i][j] = val;

}

}

return dp[e][f];

}

**############################################################**

**9. Longest Common Subsequence**

Given two sequences, find the length of longest subsequence present in both of them. Both the strings are of uppercase.

**Example 1:**

**Input:**

A = 6, B = 6

str1 = ABCDGH

str2 = AEDFHR

**Output:** 3

**Explanation:** LCS for input Sequences

“ABCDGH” and “AEDFHR” is “ADH” of

length 3.

**Example 2:**

**Input:**

A = 3, B = 2

str1 = ABC

str2 = AC

**Output:** 2

**Explanation:** LCS of "ABC" and "AC" is

"AC" of length 2.

**Your Task:**  
Complete the function **lcs()** which takes the length of two strings respectively and two strings as input parameters and returns the length of the longest subsequence present in both of them.

**Expected Time Complexity** : O(|str1|\*|str2|)  
**Expected Auxiliary Space**: O(|str1|\*|str2|)

**Constraints:**  
1<=size(str1),size(str2)<=103

int lcs(int x, int y, string s1, string s2)

{

int dp[x+1][y+1];

for(int i=0; i<=x; i++)

dp[i][0] = 0;

for(int j=1; j<=y; j++)

dp[0][j] = 0;

for(int i=1; i<=x; i++)

for(int j=1; j<=y; j++)

if(s1[i-1] == s2[j-1])

dp[i][j] = dp[i-1][j-1] + 1;

else

dp[i][j] = max(dp[i-1][j], dp[i][j-1]);

return dp[x][y];

}

**############################################################**

**10. Subset Sum Problem**

Given a set of numbers, check whether it can be partitioned into two subsets such that the sum of elements in both subsets is same or not.

**Example 1:**

**Input:**

N = 4

arr[] = {1,5,11,5}

**Output:** YES

**Explanation:** There exists two subsets

such that {1, 5, 5} and {11}.

**Example 2:**

**Input:**

N = 3

arr[] = {1,3,5}

**Output:** NO

**Your Task:**  
Your task is to complete the **findPartition()**function which takes an array a[] and N as input parameter and return **true**if the given set can be partitionedinto two subsets such that the sum of elements in both subsets is equal, else return **false**.  
**Note:**The output will be YES or NO depending upon the value returned by your code. The printing is done by the driver's code.

**Expected Time Complexity**: O(N\*S).  
**Expected Auxiliary Space**: O(S) where S is the sum of the given Array.

**Constraints:**  
1 <= N <= 100  
0 <= arr[i] <= 1000

bool findPartition(int arr[], int n)

{

int sum = 0;

for(int i=0; i<n; i++)

sum += arr[i];

if(sum % 2 != 0)

return false;

sum /=2;

bool dp[n+1][sum+1];

for(int i=0; i<=n; i++)

dp[i][0] = true;

for(int j=1; j<=sum; j++)

dp[0][j] = false;

for(int i=1; i<=n; i++)

for(int j=1; j<=sum; j++)

if(j >= arr[i-1])

dp[i][j] = dp[i-1][j] || dp[i-1][j-arr[i-1]];

else

dp[i][j] = dp[i-1][j];

return dp[n][sum];

}

**############################################################**