1. **Top Down = Recursion + Memoization.**

**Time complexity: O(N), Space Complexity: O(N)**

#include <iostream>

#include<vector>

using namespace std;

int fib(int num,vector<int>& dp){

If(num<=1) return num;

//step 3 return if already calculated

if(dp[num]!=-1) return dp[num];

//step 2 store the value

dp[num] = fib(num-1,dp)+fib(num-2,dp);

return dp[num];

}

int main()

{

int num;

cout<<"Enter Number: ";

cin>>num;

//step 1, storage creation

vector<int> dp(num+1,-1);

fib(num,dp);

cout<<dp[num]<<endl;

return 0;

}

2. **Bottom up Approach:**

1. Create dp array;
2. Initialize with known value.
3. Store the value and return

#include <iostream>

#include<vector>

using namespace std;

int main()

{

int num;

cout<<"Enter Number: ";

cin>>num;

//step 1, storage creation

vector<int> dp(num+1,-1);

//step 2, initialize with known value

dp[0]=0; dp[1]=1;

//step 3

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

dp[i] = dp[i-1]+dp[i-2];

}

cout<<dp[num]<<endl;

return 0;

}

**1D DP..How to decide if it is DP problem:**

Should ask “count the total number of ways”. Or asking the Max/min values sum.

**All possible ways we need to apply recursion.**

**How to solve:**

1. **Try to represent the problem in terms of index.**
2. **Do all possible stuff on index according to the problem statement.**
3. **Sum up all the stuff(count all ways)**

**Min(find min) same for Max.**