**Day-18**

**Problem1**- [**152. Maximum Product Subarray**](https://leetcode.com/problems/maximum-product-subarray/)

Solution-

class Solution {

public:

int maxProduct(vector<int>& nums) {

int n= nums.size();

int pre=1;

int suff=1;

int m=0;

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

{

if(pre==0)pre=1;

if(suff==0)suff=1;

pre =pre\*nums[i];

suff=suff\*nums[n-i-1];

m=max(m,max(pre,suff));

}

if(n==1 && nums[0]<0)

return nums[0];

return m;

}

};**Problem2**- [**322. Coin Change**](https://leetcode.com/problems/coin-change/)

Solution-

#include <vector>

#include <climits> // for INT\_MAX

using namespace std;

class Solution {

public:

int solve(vector<int>& coins, int amount, vector<int>& dp) {

if (amount == 0) {

return 0;

}

if (amount < 0) {

return INT\_MAX;

}

if (dp[amount] != -1) {

return dp[amount];

}

int minCoins = INT\_MAX;

for (int i = 0; i < coins.size(); i++) {

int ans = solve(coins, amount - coins[i], dp);

// Only update if subProblem is not INT\_MAX (meaning valid solution exists)

if (ans != INT\_MAX) {

minCoins = min(minCoins, ans + 1); // +1 because we are considering this coin as a part of solution

}

}

// Memoize the result

dp[amount] = (minCoins == INT\_MAX) ? INT\_MAX : minCoins;

return dp[amount];

}

int coinChange(vector<int>& coins, int amount) {

vector<int> dp(amount + 1, -1); // Initialize dp with -1

int result = solve(coins, amount, dp);

// Check if result is INT\_MAX (no valid solution)

return (result == INT\_MAX) ? -1 : result;

}

};

**Problem 3**- [**188. Best Time to Buy and Sell Stock IV**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/)

class Solution {

public:

int solve(int ind,int opr, vector<int>&prices,int k,vector<vector<int>>&dp)

{

int n=prices.size();

if(ind ==n)return 0;

if(opr>=2\*k)return 0;

if(dp[ind][opr]!=-1)

return dp[ind][opr];

int profit=0;

if(opr%2==0)

{

int buyy= -prices[ind] + solve(ind+1,opr+1,prices,k,dp);

int skip= 0 + solve(ind+1,opr,prices,k,dp);

profit= max(buyy,skip);

}

else

{

int sale= prices[ind] + solve(ind+1,opr+1,prices,k,dp);

int skip= 0 + solve(ind+1,opr,prices,k,dp);

profit=max(sale,skip);

}

return dp[ind][opr]=profit;

}

int maxProfit(int k, vector<int>& prices) {

int n=prices.size();

vector<vector<int>>dp(n,vector<int>(2\*k,-1));

return solve(0,0,prices,k,dp);

}

};