309. **Best Time to Buy and Sell Stock With Cooldown**

Problem :
Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

```
Input: [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell, cooldown, buy, sell]
```

Intuition :
We maintain max profit for each day corresponding to "Buy","Sell","Cooldown"
How?
Create an array dp[n+1][3]
//where dp[i][1] corresponds to max profit at day i if i BUY a stock or NOT
//dp[i][0] corresponds to max profit at day i if it is a COOLDOWN day
//dp[i][1] corresponds to max profit at day i if I SELL on this day or NOT
so,
dp[i][0] = dp[i-1][2] // IF ith day is cooldown then max profit at this day is going to be equal to what we got from selling on previous day
dp[i][1] = max(dp[i-1][1],dp[i][0] - prices[i]) // If ith day is buying day then max profit is maximum of (profit obtained by buying on prevoius day, profit obtained on buying on this day ---> Previous day was 'cooldown' day)
dp[i][2] = max(dp[i-1][2],dp[i-1][1] + prices[i]) // If this day is selling day the max profit is maximum of (profit obtained by selliong on previous day, profit obtained by selling on this day ---> Previous day was 'buying' day)
Finally **MAXIMUM PROFIT = MAX(dp[n][0],dp[n][1],dp[n][2])**

complexity : O(n)

CODE :

```
int maxProfit(vector<int>& prices) {
        if(prices.size() <= 0)
           return 0;
        int n = prices.size();
        int dp[n+1][3];
        dp[0][0] = dp[0][1] = dp[0][2] = 0;
        dp[1][0] = 0;                // cooldown
        dp[1][1] = -prices[0];          //buy
        dp[1][2] = 0;                //sell
        for(int i=2;i<=n;i++){
           dp[i][0] = dp[i-1][2];
           dp[i][1] = max(dp[i-1][1],dp[i-1][0] - prices[i-1]);
           dp[i][2] = max(dp[i-1][2],dp[i-1][1] + prices[i-1]);
        }
        return max(dp[n][0],max(dp[n][1],dp[n][2]));
    }
```