1. Best Time to Buy and Sell Stock IV

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 at most **k** transactions.

**Note:** You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

**Example 1:**

Input: [2,4,1], k = 2  
Output: 2  
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.

**Example 2:**

Input: [3,2,6,5,0,3], k = 2  
Output: 7  
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4.  
 Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.

**解** 动态规划

用一个二维矩阵trans，矩阵的列表示交易的价格，矩阵的行表示交易的次数，矩阵的值表示当前能够获得的最大利润

以允许3交易为例，考察第3天的最大利润trans[3][3]

* 第3天什么都不做，
* 以第1天价格买入，第3天价格卖出，
* 以第2天价格买入，第3天价格卖出，

取最大值即为结果。实现如下：

用两个数组buy和sell来分别表示第i次买入交易的最大收益值和第i次卖出交易的最大收益值

１．第ｉ次买操作买下当前股票之后剩下的最大利润为第(i-1)次卖掉股票之后的利润－当前的股票价格．状态转移方程为：

　　　　buy[i] = max(sell[i-1]- curPrice, buy[i]);

２．第ｉ次卖操作卖掉当前股票之后剩下的最大利润为第ｉ次买操作之后剩下的利润＋当前股票价格．状态转移方程为：

　　　　sell[i] = max(buy[i]+curPrice, sell[i]);

class Solution {  
public:  
 int maxProfit(int k, vector<int>& prices) {  
 if(prices.size() ==0) return 0;  
 int len = prices.size(), ans =0;  
 if(k >= len/2){  
 for(int i = 1; i < len; i++)   
 if(prices[i]-prices[i-1]>0)ans += prices[i]-prices[i-1];  
 return ans;   
 }  
 vector<int> buy(len+1, INT\_MIN), sell(len+1, 0);  
 for(auto val: prices)  
 {  
 for(int i =1; i <= k; i++)  
 {  
 buy[i] = max(sell[i-1]-val, buy[i]);  
 sell[i] = max(buy[i]+val, sell[i]);  
 }  
 }  
 return sell[k];  
 }  
};

[Best Time to Buy and Sell Stock II](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/)

[Best Time to Buy and Sell Stock III](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/)

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