122. Best Time to Buy and Sell Stock II

Medium 8151 2404 Add to List Share

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the **maximum** profit you can achieve*.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.
```

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int hold[n];
        int not_hold[n];
        memset(hold, 0, sizeof(hold));
        memset(not_hold, 0, sizeof(not_hold));
        hold[0] = -prices[0];

        for (int i = 1; i < n; i++) {
            hold[i] = max(not_hold[i - 1] - prices[i], hold[i - 1]);
            not_hold[i] = max(not_hold[i - 1], hold[i - 1] + prices[i]);
        }
        return not_hold[n - 1];
    }
};
```

Propostion: `hold[i]` and `not_hold[i]` is always the maximum profit at `ith` day. `hold[i]` means the maximum profit you can earn at `ith` day if you hold a stock at `ith` day. `not_hold[i]` means the maximum profit you can earn at `ith` day if you do not hold a stock at `ith` day.

If the above proposition is proof correct, then we could just output `not_hold[n - 1]` cause it is the maximum profit we can earn at last day.

Proof by Induction:

Basecase:
when `i == 0` , if you buy a stock, then you need to pay `prices[0]`, `hold[0] = -prices[0]` , it is obvious the max profit you could earn. And if you don't buy, then of course the profit is 0, `not_hold[0] = 0`.

Induction Step:
Suppose the proposition is correct for `hold[k] and not_hold[k]`, then at `k + 1` day, there will be two cases:

1. case 1: you hold a stock at `k + 1` day. It can be the result by 1. you buy a stock at `k + 1` day. 2. you hold the stock at `k` day, and you don't sell it at `k + 1` day. Because `hold[k]` and `not_hold[k]` is the maximum profit at `k` day, then the maximum profit is the maximum profit for case 1 is `not_hold[k] - prices[k + 1]`, case 2 is `hold[k]`. The maximum profit at `k + 1` day is the larger one among the 2 cases. I.e. `hold[k + 1] = max(not_hold[k] - prices[k + 1], hold[k])`. The algorithm is correct.
2. case 2: you do not hold a stock at `k + 1` day. It can be the result by 1. you sell a stock at `k + 1` day. 2. you do not hold the stock at `k` day, and you don't buy one at `k + 1` day. Because `hold[k]` and `not_hold[k]` is the maximum profit at `k` day, then the maximum profit is the maximum profit for case 1 is `hold[k] + prices[k + 1]`, case 2 is `not_hold[k]`. The maximum profit at `k + 1` day is the larger one among the 2 cases. I.e. `not_hold[k + 1] = max(not_hold[k] + prices[k + 1], not_hold[k])`. The algorithm is correct.
   Therefore `hold[k + 1]` and `not_hold[k + 1]` is the maximum profit at `k + 1` day. The proposition is correct.

Time complexity:
We traverse the prices vector for one time, so it is O(n) where n is the length of the prices.