[hard,math]123.Best Time to Buy and Sell Stock III

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 *two* transactions.

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

I use the same code. In the beginning the part where we compute lowestBuyPrice2 make me so confusing. Then I try to work on math myself to see how it is so. Hope following explanation will be helpful to people who have same confusion as I.

So, in following code, the lowestBuyPrice1 and maxProfit1 are very easy to understand. The only thing that may take time to understand is the computation of lowestBuyPrice2. **Here lowestBuyPrice2 actually is not the exact price of the one we bought the stock** in the second transactoin. Actually, it contains two parts if we can open it as

```
"lowestBuyPrice2" = buyPrice2 - maxProfit1
                  = buyPrice2 - (highestSellPrice1 - lowestBuyPrice1).
```

So you will see, *"lowestBuyPrice2"* contains the buy price of 2nd transaction as well as the profit we obtained for the 1st transaction. When we compute

```
"maxProfit2" = sellPrice2 - lowestBuyPrice2
             = sellPrice2 - buyPrice2 + maxProfit1
             = profitOf2ndTrans + maxProfit1.
```

So, at the end of computation, *"maxProfit2"* will contain profits for both transactions.

To avoid ambiguity of names used for these variables, I use italic font to separate the variables (*"lowestBuyPrice2", "maxProfit2"*) whose meaning may be confusing.

这个是真的很难理解。

第二次的买价 = 当前价格 - 第一次交易赚的钱

第二次的盈利 = 当前价格 - 第二次的买价。

举几个例子

```
1 2 3 4 5
  maxProfit2 0  lowestBuyPrice2 =inf    maxProfit1 0    lowestBuyPrice1
+inf

今天价 1 maxProfit2 0  lowestBuyPrice2 =1  maxProfit1 0    lowestBuyPrice1 1
今天价 2 maxProfit2 1  lowestBuyPrice2 =1  maxProfit1 1    lowestBuyPrice1 1
今天价 3 maxProfit2 2  lowestBuyPrice2 =1  maxProfit1 2    lowestBuyPrice1 1
今天价 4 maxProfit2 3  lowestBuyPrice2 =1  maxProfit1 3    lowestBuyPrice1 1
今天价 5 maxProfit2 4  lowestBuyPrice2 =1  maxProfit1 4    lowestBuyPrice1 1
```

再来一个随意的情况

```
1 3 0 4 2
  maxProfit2 0  lowestBuyPrice2 =inf    maxProfit1 0    lowestBuyPrice1
+inf

今天价 1 maxProfit2 0  lowestBuyPrice2 =1  maxProfit1 0    lowestBuyPrice1 1
今天价 3 maxProfit2 2  lowestBuyPrice2 =1  maxProfit1 2    lowestBuyPrice1 1
今天价 0 maxProfit2 2  lowestBuyPrice2 = -2    maxProfit1 2
lowestBuyPrice1 0
今天价 4 maxProfit2 6  lowestBuyPrice2 = -2    maxProfit1 4
lowestBuyPrice1 0
今天价 2 maxProfit2 6  lowestBuyPrice2 = -2    maxProfit1 4
lowestBuyPrice1 0
```

整体来看，maxProfit1 总在买，而maxProfit2 做累加。

Java 版本如下：

```java
public class Solution {
    public int maxProfit(int[] prices) {
        int maxProfit1 = 0;
        int maxProfit2 = 0;
        int lowestBuyPrice1 = Integer.MAX_VALUE;
        int lowestBuyPrice2 = Integer.MAX_VALUE;

        for(int p:prices){
            maxProfit2 = Math.max(maxProfit2, p-lowestBuyPrice2);
            lowestBuyPrice2 = Math.min(lowestBuyPrice2, p-maxProfit1);
            maxProfit1 = Math.max(maxProfit1, p-lowestBuyPrice1);
            lowestBuyPrice1 = Math.min(lowestBuyPrice1, p);
        }
        return maxProfit2;
    }
}
```

cpp

```cpp
#include<climits>
#include<algorithm>
using namespace std;
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int maxProfit1 = 0;
        int maxProfit2 = 0;
        int lowestBuyPrice1 = INT_MAX;
        int lowestBuyPrice2 = INT_MAX;

        for( int p: prices ){
            maxProfit2 = max(maxProfit2, p-lowestBuyPrice2);
            lowestBuyPrice2 = min(lowestBuyPrice2, p-maxProfit1);
            maxProfit1 = max(maxProfit1, p-lowestBuyPrice1);
            lowestBuyPrice1 = min(lowestBuyPrice1, p);
        }

        return maxProfit2;
    }
};
```