123. Best Time to Buy and Sell Stock III

Hard 2260 76 Add to List Share

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 (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
             Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.
```

Example 2:

```
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
             Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are
             engaging multiple transactions at the same time. You must sell before buying again.
```

Example 3:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

Accepted 219,138 | Submissions 584,752

```java
class Solution {
    public int maxProfit(int[] prices) {
        int oneBuy = Integer.MAX_VALUE;
        int twoBuy = Integer.MAX_VALUE;
        int oneBuyOneSell = 0;
        int twoBuyTwoSell = 0;

        for (int i = 0; i < prices.length; i++) {
            oneBuy = Math.min(oneBuy, prices[i]);
            oneBuyOneSell = Math.max(oneBuyOneSell, prices[i] - oneBuy);
            twoBuy = Math.min(twoBuy, prices[i] - oneBuyOneSell);
            twoBuyTwoSell = Math.max(twoBuyTwoSell, prices[i] - twoBuy);
        }

        return twoBuyTwoSell;
    }
}
```

Your previous code was restored from your local storage. Reset to default