/\*

Say you have an array for which the ith 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). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

在同一天，可以先卖出，然后买入

way-1:

三种情况：

若今天是最后一天，只能卖

若今天比昨天价格低，而且不是昨天买的。那么就昨天卖，今天买入。

若今天比昨天价格低，是昨天买的，就改为今天买入。

way-2:贪心算法，只要后一项比前一项大，就买入并于后一项卖出。

\*/

class Solution {

public:

int maxProfit(vector<int>& price)

{

//way-1

/\*

int buy=0;

int yesterday=buy;

int max=0;

for(int i=1;i<price.size();i++)

{

if(price[i]>=price[yesterday] && i==price.size()-1)

{

max+=price[i]-price[buy];

}

else if(price[i]<price[yesterday] && yesterday!=buy)

{

max+=price[yesterday]-price[buy];

buy=i;

}

else if((price[i]<price[yesterday] && yesterday==buy))

{

buy=i;

}

yesterday=i;

}

return max;

\*/

//way-2

if(price.size()<2)

return 0;

int temp=price[0];

int pre;

int sum=0;

for(int i=0;i<price.size()-1;i++)

{

pre=price[i+1];

if(pre>temp)

sum+=(pre-temp);

temp=pre;

}

return sum;

}

};