/\*

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 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).

最多两次交易

way-1:用分治算法，以i为分界点，找出prices[0....i]的利润最大的交易。再找出prices[i.....length-1]的利润的最大交易。

遍历所有的i，找出利润最大的即可。

front[i]存的是i之前的一次交易最大收益

back[i]存的是i之后的一次交易最大收益

\*/

class Solution {

public:

int maxProfit(vector<int>& prices)

{

int length=prices.size();

if(length<2)

return 0;

int front[length]={0};

int back[length]={0};

int min1=prices[0];

for(int i=1;i<length;i++)

{

min1=min(min1,prices[i]);

front[i]=max(front[i-1],prices[i]-min1);

}

int max1=prices[length-1];

for(int i=length-2;i>=0;i--)

{

max1=max(max1,prices[i]);

back[i]=max(back[i+1],max1-prices[i]);

}

int maxx=0;

for(int i=0;i<length;i++)

{

maxx=max(maxx,front[i]+back[i]);

}

return maxx;

}

};