Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

**Example 1:**

**Input:** [7,1,5,3,6,4]

**Output:** 5

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

  Not 7-1 = 6, as selling price needs to be larger than buying price.

**Example 2:**

**Input:** [7,6,4,3,1]

**Output:** 0

**Explanation:** In this case, no transaction is done, i.e. max profit = 0.

解法：

动态规划

我们要观察一个事实

就是对于最低价格i，最高价格k

i比他前面所有价格都低，k比后面所有价格都高，不然我完全可以在更便宜的前K天买入或者后面几天卖出

注意3562918,虽然1小于2，但是2是比他前面的都低，所以对于2来说，他那里能选的最低买入点是2

创建两个array

min

maxprofit //不创造max因为如果创造max，就要遍历整个后面，没意义

class Solution {

public int maxProfit(int[] prices) {

int n=prices.length;

if (n<=1) return 0;

int[] min=new int[n];

int[] maxprofit=new int[n];

min[0]=prices[0]; //一开始最低价是price0

maxprofit[0]=0;

for(int i=1;i<n;i++){

min[i]=Math.min(min[i-1],prices[i]); //min[i-1]是保存了前面为止最低值的

maxprofit[i]=Math.max(maxprofit[i-1],prices[i]-min[i]); //maxprofit[i-1]只是一个单纯的价格，他可能取min的点与min[i]并不同，但是，我们能比出两个差价，如果我心差价更大，完全可以取而代之，而且Min[i]是已经更新过的

}

return maxprofit[n-1];//那么最后一个数必然保存着最大价格

}

}