121. 买卖股票的最佳时机

给定一个数组 prices ，它的第 i 个元素 prices[i] 表示一支给定股票第 i 天的价格。

你只能选择 某一天 买入这只股票，并选择在 未来的某一个不同的日子 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回0 。

输入样例

7,1,5,3,6,4

输出样例

5

解析

暴力法

最直观的想法就是暴力求解最优间距。当然这种算法也是性能最差的。

```
#include <bits/stdc++.h>

using namespace std;

int a[10000];
int result = 0;

int main() {
    int temp;
    int index = 0;
    while (cin >> temp) {
        a[index] = temp;
        index++;
    }
    for (int i = 0; i < index; ++i) {
        for (int j = i+1; j < index; ++j) {
            result = max(a[j] - a[i], result);
        }
    }
    cout << result;
    return 0;
}
```

贪心法

贪心法是在计算过程中找到左侧的最小值，然后计算当前的差值。

```
#include <bits/stdc++.h>

using namespace std;

int a[10000];
int result = 0;

int main() {
    int temp;
    int index = 0;
    while (cin >> temp) {
        a[index] = temp;
        index++;
    }
    int low = INT_MAX;
    for (int i = 0; i < index; ++i) {
        low = min(low, a[i]);
        result = max(a[i] - low,result);
    }
    cout << result;
    return 0;
}
```

动态规划

我们设dp[i]为第i天的最大收益，则很容易想到第i天的最大收益来源有两个：

1、前一天的最大收益，即dp[i-1]

2、第i天与前面价值最小的一天的差值，即price[i] - minPrice

二者当中取最大的一个。

```
#include<bits/stdc++.h>

using namespace std;

class Solution {
public:
    int maxProfit(vector<int> &prices) {
        int res = 0;
        int minPrice = prices[0];
        //遍历全部的价格
       for (int i = 1; i < prices.size(); ++i) {
            minPrice = min(minPrice, prices[i]);
```

```
            res = max(res, prices[i] - minPrice);
        }
        return res;
    }
};

int main() {
    Solution s;
    vector<int> g = {7, 1, 5, 3, 6, 4};
    cout << s.maxProfit(g);
}
```