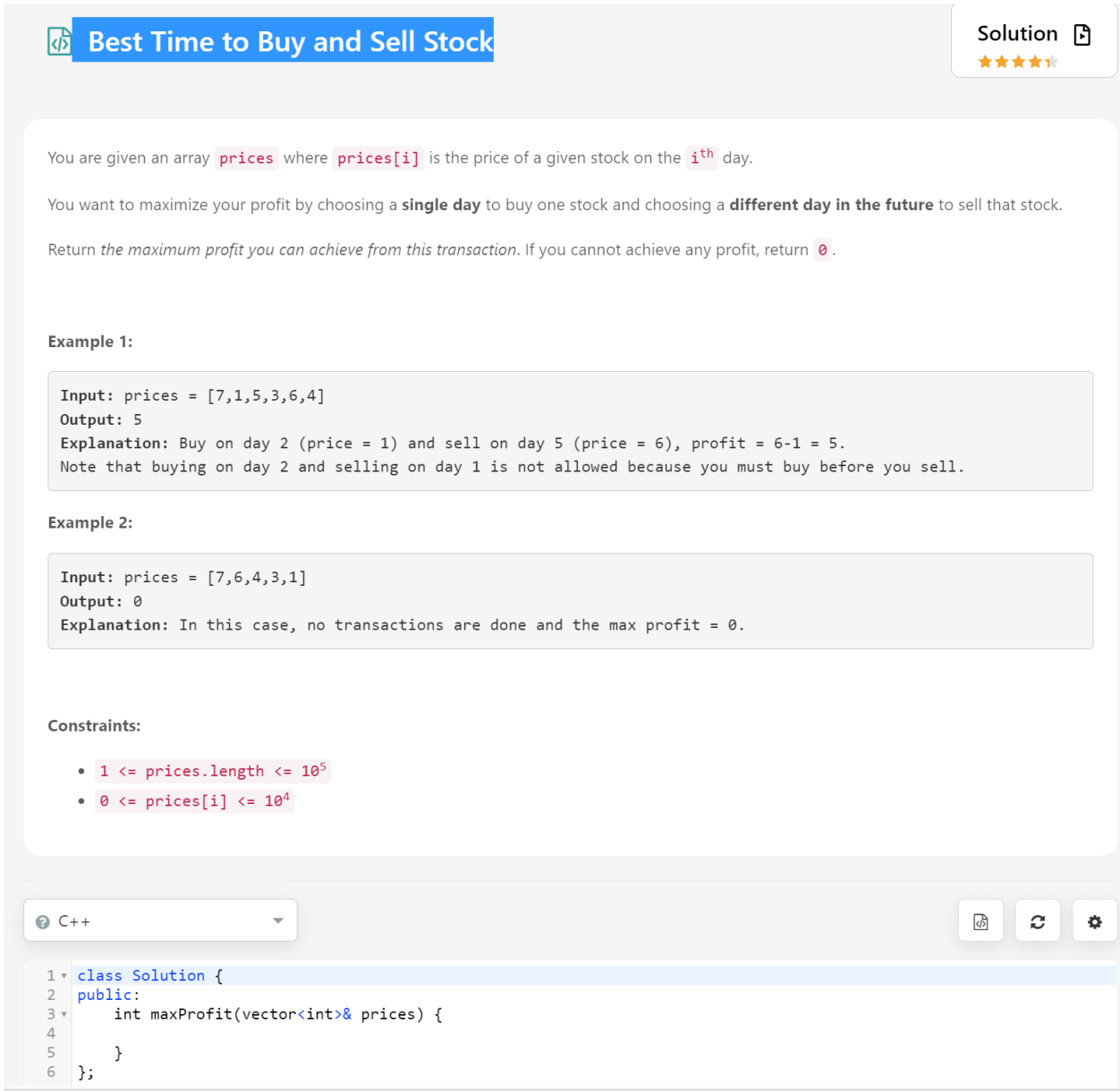


Solution Article

We need to find out the maximum difference (which will be the maximum profit) between two numbers in the given array. Also, the second number (selling price) must be larger than the first one (buying price).

In formal terms, we need to find $\max(\text{prices}[j] - \text{prices}[i])$, for every i and j such that $j > i$.

Approach 1: Brute Force

public class Solution {

```
    public int maxProfit(int prices[]) {
        int maxprofit = 0;
        for (int i = 0; i < prices.length - 1; i++) {
            for (int j = i + 1; j < prices.length; j++) {
                int profit = prices[j] - prices[i];
                if (profit > maxprofit)
                    maxprofit = profit;
            }
        }
        return maxprofit;
    }
}
```

Complexity Analysis

- Time complexity: $O(n^2)$. Loop runs $\frac{n(n-1)}{2}$ times.
- Space complexity: $O(1)$. Only two variables - $\mathrm{maxprofit}$ and profit are used.

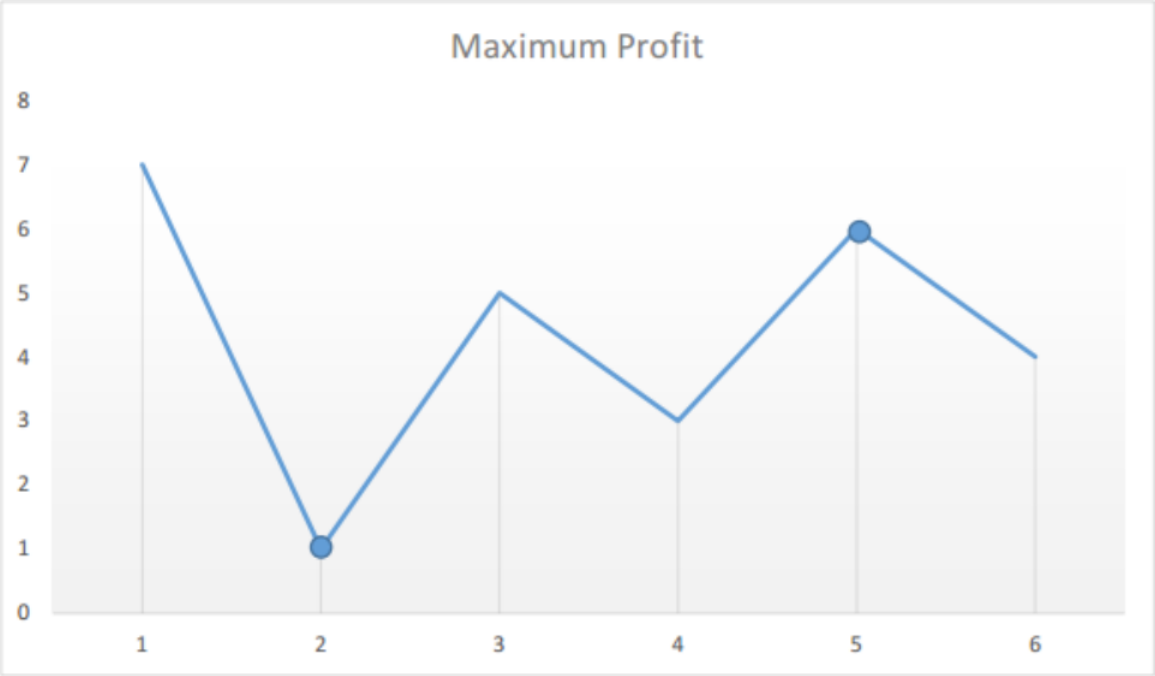
Approach 2: One Pass

Algorithm

Say the given array is:

[7, 1, 5, 3, 6, 4]

If we plot the numbers of the given array on a graph, we get:



The points of interest are the peaks and valleys in the given graph. We need to find the largest peak following the smallest valley. We can maintain two variables - minprice and maxprofit corresponding to the smallest valley and maximum profit (maximum difference between selling price and minprice) obtained so far respectively.

```
public class Solution {
    public int maxProfit(int prices[]) {
        int minprice = Integer.MAX_VALUE;
        int maxprofit = 0;
        for (int i = 0; i < prices.length; i++) {
            if (prices[i] < minprice)
                minprice = prices[i];
            else if (prices[i] - minprice > maxprofit)
                maxprofit = prices[i] - minprice;
        }
        return maxprofit;
    }
```

}

Complexity Analysis

- Time complexity: $O(n)$. Only a single pass is needed.
- Space complexity: $O(1)$. Only two variables are used.

Comment:

The sentence "We need to find the largest peak following the smallest valley" is plain wrong. What we are actually doing is this: for every element, we are calculating the difference between that element and the minimum of all the values before that element and we are updating the maximum profit if the difference thus found is greater than the current maximum profit.