Interviewer Packet B - TIPS 4

# Interviewer:

## Behavioral:

Tell me about a time when you were under a lot of pressure. What was going on, and how did you get through it?

## Question:

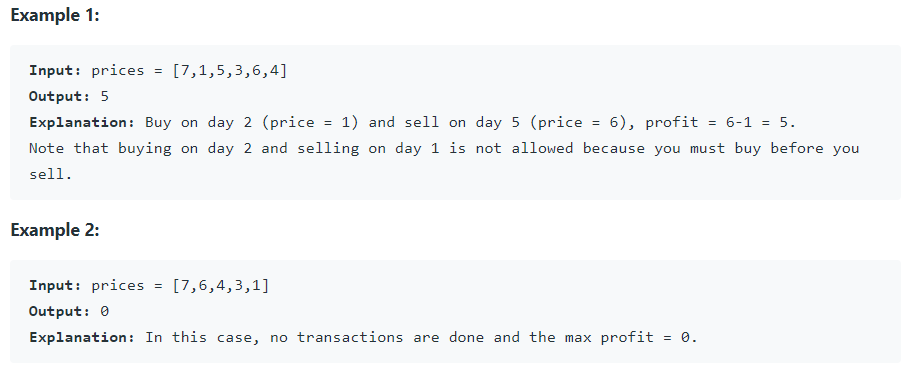
<https://leetcode.com/problems/best-time-to-buy-and-sell-stock/>

You are given an array prices where prices[i] is the price of a given stock on the ith day.

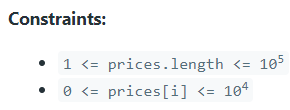
You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

## Examples:



## Follow up Q&A:



* Can there be multiple purchases and sales of stock?
  + No. You buy one share on one day and sell it on a later day.

## Hint(s):

*Ask if they would like a hint before giving a hint*

## Solution(s): (General concept and time/space complexity)

### Brute force

class Solution {

public:

int maxProfit(vector<int>& prices) {

int max\_profit = 0;

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

for (int j = i + 1; j < prices.size(); j++) {

int profit = prices[j] - prices[i];

if (profit > max\_profit)

max\_profit = profit;

}

}

return max\_profit;

}

};

Normally generates a Time Limit Exceeded error in Leetcode. For each element in the array except the last element, compute the profit generated from buying on that day and selling on a later date. Keep track of the highest profit seen. After loops terminate, return the highest profit.

Time complexity: O(n^2) due to nested O(n) for-loops.

Space complexity: O(1). Only uses four extra variables if you include the index variables.

### One pass

class Solution {

public:

int maxProfit(vector<int>& prices) {

int min\_price = INT\_MAX;

int max\_profit = 0;

for (int i = 0; i < prices.size(); i++) {

if (prices[i] < min\_price)

min\_price = prices[i];

else if (prices[i] - min\_price > max\_profit)

max\_profit = prices[i] - min\_price;

}

return max\_profit;

}

};

One pass that keeps track of the maximum profit and minimum price. If the current price is less than the current minimum price, update the current minimum price. If the difference between the current price and the minimum price is greater than the maximum profit, update the maximum profit.

Time complexity: O(n). Solves in a single pass.

Space complexity: O(1). Only uses four extra variables if you include the index variables.

### Other questions follow up

*Ask if there is more than 5 minutes remaining when they finish their code and testing.*

* What are the time and space complexities of your algorithm?
* If you were allowed to make as many transactions as you wanted, how would that change your solution? (This is a [Leetcode Medium by itself](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/), so don’t waste too much time on it. The basic idea is that you should buy at valleys and sell at peaks - to find the max profit, you can actually just get away with adding any day-to-day increases in price to the max\_profit variable and ignoring day-to-day drops.)

# Interviewee:

## Question:

<https://leetcode.com/problems/asteroid-collision/>

We are given an array asteroids of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

## Example(s):

Example 1:

Input: asteroids = [5,10,-5]

Output: [5,10]

Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:

Input: asteroids = [8,-8]

Output: []

Explanation: The 8 and -8 collide exploding each other.

Example 3:

Input: asteroids = [10,2,-5]

Output: [10]

Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Example 4:

Input: asteroids = [-2,-1,1,2]

Output: [-2,-1,1,2]

Explanation: The -2 and -1 are moving left, while the 1 and 2 are moving right. Asteroids moving the same direction never meet, so no asteroids will meet each other.

## Code below or on leetcode

class Solution {

public:

vector<int> asteroidCollision(vector<int>& asteroids) {

}

};