Best
Time to Buy and Sell Stock

Difficulty	Easy
Category	Sliding Window Two Pointers
Question	https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
Solution	https://youtu.be/1pkOgXD63yU
Status	Not started

Question

> 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`.

Example

Example 1:

```
Input: prices = [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.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you
sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

Idea

Two Pointer: Left one for buying and Right one for selling

Solution

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # Initialize two pointers: l for buying and r for selling
        l, r = 0, 1
        maxP = 0  # Initialize the maximum profit

        while r < len(prices):
            if prices[r] > prices[l]:
                # Calculate the profit if selling on day r
                profit = prices[r] - prices[l]
                maxP = max(profit, maxP)  # Update maxP if profit is higher
            else:
                # If the price is lower, update the buying pointer (l)
                l = r
            r += 1  # Move the selling pointer to the next day

        return maxP

# Time Complexity:
# The while loop runs for each element in the prices array once, so the time complexity is O(n),
# where n is the length of the prices array.

# Space Complexity:
# The code uses a constant amount of extra space, regardless of the input size,
# so the space complexity is O(1).
```

> **Difference from the Two-Pointer Approach:**
>
> - In the sliding window approach, we maintain a sliding window over the array to keep track of the minimum buying price and calculate

the profit by selling at the current day.

- In contrast, the two-pointer approach uses two pointers to find the maximum profit by iterating through the array only once.
- Both approaches have a time complexity of O(n), but the sliding window approach may have a slightly higher constant factor due to the additional calculation of the profit at each step.
- The sliding window approach is more versatile and can be applied to a wider range of problems involving contiguous subarrays or subranges, while the two-pointer approach is specifically tailored for scenarios like this where you're interested in elements between two pointers.