3652. Best Time to Buy and Sell Stock using Strategy

Solved

Medium | Topics | Companies | Hint

You are given two integer arrays `prices` and `strategy`, where:

- `prices[i]` is the price of a given stock on the i^{th} day.
- `strategy[i]` represents a trading action on the i^{th} day, where:
  - `-1` indicates buying one unit of the stock.
  - `0` indicates holding the stock.
  - `1` indicates selling one unit of the stock.

You are also given an **even** integer `k`, and may perform **at most one** modification to `strategy`. A modification consists of:

- Selecting exactly `k` **consecutive** elements in `strategy`.
- Set the **first** `k / 2` elements to `0` (hold).
- Set the **last** `k / 2` elements to `1` (sell).

The **profit** is defined as the **sum** of `strategy[i] * prices[i]` across all days.

Return the **maximum** possible profit you can achieve.

Note: There are no constraints on budget or stock ownership, so all buy and sell operations are feasible regardless of past actions.

Example 1:

> **Input:** `prices = [4,2,8], strategy = [-1,0,1], k = 2`
>
> **Output:** `10`
>
> **Explanation:**

Modification	Strategy	Profit Calculation	Profit
Original	[-1, 0, 1]	(-1 × 4) + (0 × 2) + (1 × 8) = -4 + 0 + 8	4
Modify [0, 1]	[0, 1, 1]	(0 × 4) + (1 × 2) + (1 × 8) = 0 + 2 + 8	10
Modify [1, 2]	[-1, 0, 1]	(-1 × 4) + (0 × 2) + (1 × 8) = -4 + 0 + 8	4

> Thus, the maximum possible profit is 10, which is achieved by modifying the subarray `[0, 1]`

Example 2:

Input: `prices = [5,4,3], strategy = [1,1,0], k = 2`

Output: 9

Explanation:

Modification	Strategy	Profit Calculation	Profit
Original	[1, 1, 0]	(1 × 5) + (1 × 4) + (0 × 3) = 5 + 4 + 0	9
Modify [0, 1]	[0, 1, 0]	(0 × 5) + (1 × 4) + (0 × 3) = 0 + 4 + 0	4
Modify [1, 2]	[1, 0, 1]	(1 × 5) + (0 × 4) + (1 × 3) = 5 + 0 + 3	8

Thus, the maximum possible profit is 9, which is achieved without any modification.

Constraints:

- `2 <= prices.length == strategy.length <= 10^5`
- `1 <= prices[i] <= 10^5`
- `-1 <= strategy[i] <= 1`
- `2 <= k <= prices.length`
- `k` is even

Python:

```
class Solution:
    def maxProfit(self, prices: list[int], strat: list[int], k: int) -> int:
        n = len(prices)
        h = k // 2

        sp = [s * p for s, p in zip(strat, prices)]
        base = sum(sp)

        if n == k:
```

```
            change = sum(prices[h:]) - base
            return base + max(0, change)

        win_orig = sum(sp[:k])
        win_mod = sum(prices[h:k])
        max_ch = win_mod - win_orig

        for i in range(1, n - k + 1):
            win_orig += sp[i+k-1] - sp[i-1]
            win_mod += prices[i+k-1] - prices[i-1+h]
            max_ch = max(max_ch, win_mod - win_orig)

        return base + max(0, max_ch)
```

JavaScript:

```
/**
 * @param {number[]} prices
 * @param {number[]} strategy
 * @param {number} k
 * @return {number}
 */
var maxProfit = function(prices, strategy, k) {
    const n = prices.length;

    // Initialize prefix sum arrays
    const pref_price = [0];
    const res_pref = [0];

    // Calculate prefix sums for prices and strategy-based results
    for (let i = 0; i < n; i++) {
        pref_price.push(
            pref_price[i] + prices[i]
        );
        res_pref.push(
            res_pref[i] + prices[i] * strategy[i]
        );
    }

    // Initialize sliding window pointers
    let start = 0;
    let end = k - 1;
    // Initial maximum profit is the total from the original strategy
    let maxi = res_pref[n];
```

```
    // Slide a window of size k to find the maximum possible profit
    while (end < n) {
        // Profit from the strategy before the window
        const left = res_pref[start];
        // Sum of prices in the second half of the window
        const mid = pref_price[end + 1] - pref_price[end + 1 - Math.floor(k / 2)];
        // Profit from the strategy after the window
        const right = res_pref[n] - res_pref[end + 1];

        // Check if this window configuration yields a better profit
        maxi = Math.max(maxi, left + mid + right);

        // Move the window one position to the right
        start++;
        end++;
    }

    return maxi;
};
```

Java:

```
class Solution {
    public long maxProfit(int[] prices, int[] strat, int k) {
        int n = prices.length;
        int h = k / 2;
        long[] sp = new long[n];
        long base = 0;
        for (int i = 0; i < n; i++) {
            sp[i] = (long)strat[i] * prices[i];
            base += sp[i];
        }

        if (n == k) {
            long winOrig = base;
            long winMod = 0;
            for (int i = h; i < n; i++) winMod += prices[i];
            long change = winMod - winOrig;
            return base + Math.max(0, change);
        }

        long winOrig = 0;
        for (int i = 0; i < k; i++) winOrig += sp[i];

        long winMod = 0;
```

```
        for (int i = h; i < k; i++) winMod += prices[i];

        long maxCh = winMod - winOrig;

        for (int i = 1; i <= n - k; i++) {
            winOrig += sp[i + k - 1] - sp[i - 1];
            winMod += prices[i + k - 1] - prices[i - 1 + h];
            maxCh = Math.max(maxCh, winMod - winOrig);
        }

        return base + Math.max(0, maxCh);
    }
}
```