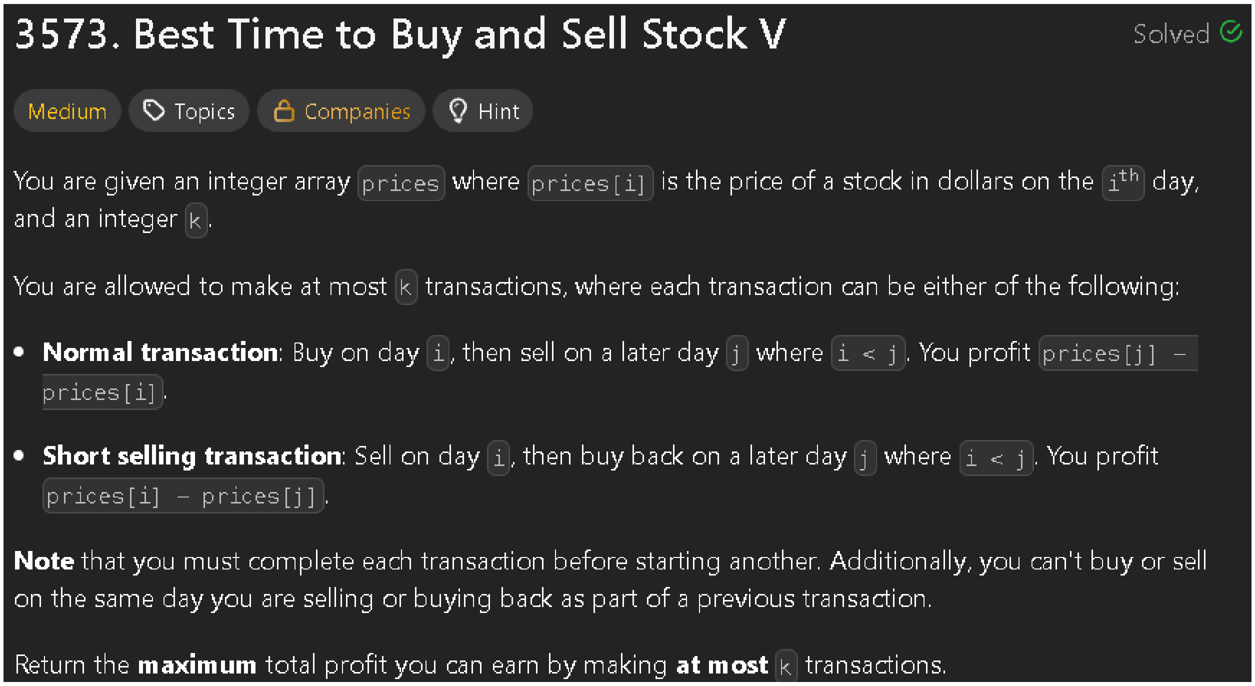


# 3573. Best Time to Buy and Sell Stock V

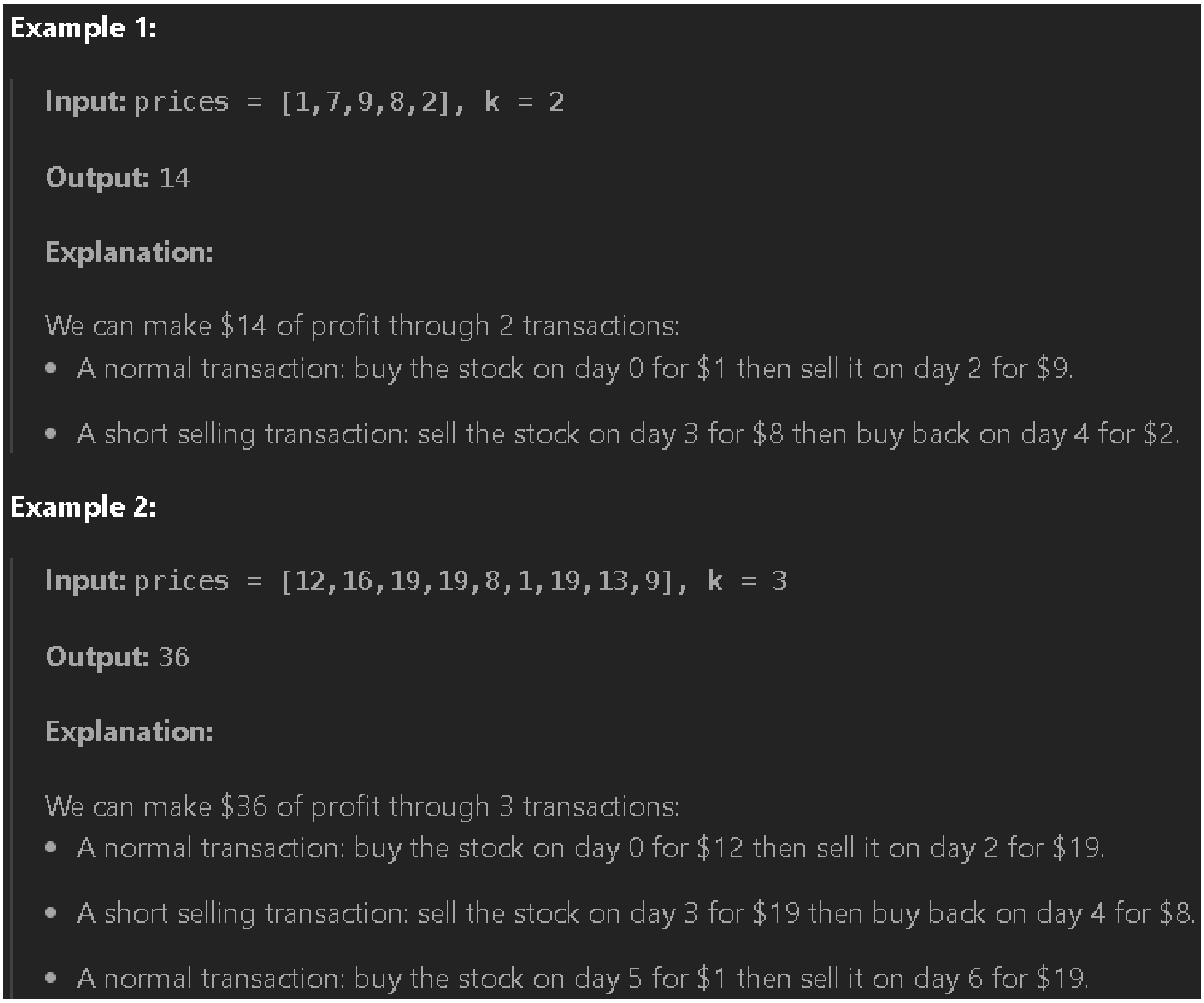
You are given an integer array `prices` where `prices[i]` is the price of a stock in dollars on the `i`th day, and an integer `k`.

You are allowed to make at most `k` transactions, where each transaction can be either of the following:

- **Normal transaction**: Buy on day `i`, then sell on a later day `j` where `i < j`. You profit `prices[j] - prices[i]`.
- **Short selling transaction**: Sell on day `i`, then buy back on a later day `j` where `i < j`. You profit `prices[i] - prices[j]`.

**Note** that you must complete each transaction before starting another. Additionally, you can't buy or sell on the same day you are selling or buying back as part of a previous transaction.

Return the **maximum** total profit you can earn by making **at most** `k` transactions.

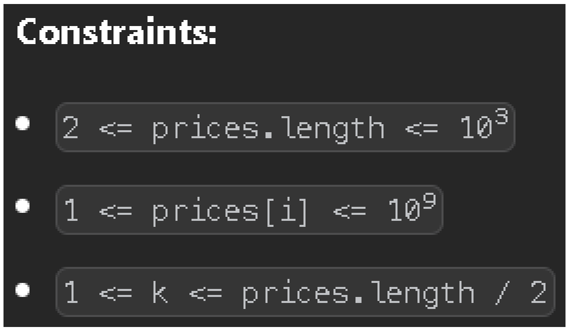


**Example 1:**

> **Input:** prices = [1,7,9,8,2], k = 2
>
> **Output:** 14
>
> **Explanation:**
>
> We can make $14 of profit through 2 transactions:
>
> - A normal transaction: buy the stock on day 0 for $1 then sell it on day 2 for $9.
> - A short selling transaction: sell the stock on day 3 for $8 then buy back on day 4 for $2.

**Example 2:**

> **Input:** prices = [12,16,19,19,8,1,19,13,9], k = 3
>
> **Output:** 36
>
> **Explanation:**
>
> We can make $36 of profit through 3 transactions:
>
> - A normal transaction: buy the stock on day 0 for $12 then sell it on day 2 for $19.
> - A short selling transaction: sell the stock on day 3 for $19 then buy back on day 4 for $8.
> - A normal transaction: buy the stock on day 5 for $1 then sell it on day 6 for $19.



**Constraints:**

- 2 <= prices.length <= $10^3$
- 1 <= prices[i] <= $10^9$
- 1 <= k <= prices.length / 2

# Python:

```
class Solution:
    def maximumProfit(self, prices: List[int], k: int) -> int:
        n = len(prices)
        mn = int(-1e14)
        dp = [[[mn] * 3 for _ in range(k + 1)] for _ in range(n + 1)]

        def f(i: int, k_left: int, state: int) -> int:
            if i == n:
                return 0 if state == 0 else mn
            if dp[i][k_left][state] != mn:
                return dp[i][k_left][state]

            p = prices[i]
            profit = mn

            # 1) do nothing
            profit = max(profit, f(i + 1, k_left, state))

            if state == 0:
                # Try buying or selling (to start a new transaction)
                profit = max(profit, f(i + 1, k_left, 1) - p)
                profit = max(profit, f(i + 1, k_left, 2) + p)
            elif k_left > 0:
                if state == 1:
                    # Complete buy-sell
                    profit = max(profit, f(i + 1, k_left - 1, 0) + p)
                else:
                    # Complete sell-buy
                    profit = max(profit, f(i + 1, k_left - 1, 0) - p)

            dp[i][k_left][state] = profit
            return profit

        return f(0, k, 0)
```

## JavaScript:

```
var maximumProfit = function(prices, k) {
    const n = prices.length;
    if (n === 0 || k === 0) return 0;

    const MIN = -1e15;
    let dp = Array.from({length: n}, () =>
        Array.from({length: k+1}, () => Array(3).fill(MIN))
    );

    for (let t = 0; t <= k; t++) {
        dp[0][t][0] = 0;
        if (t > 0) {
            dp[0][t][1] = -prices[0];
            dp[0][t][2] = prices[0];
        }
    }

    for (let i = 1; i < n; i++) {
        for (let t = 0; t <= k; t++) {
            dp[i][t][0] = dp[i-1][t][0];
            if (t <= k) {
                dp[i][t][0] = Math.max(
                    dp[i][t][0],
                    dp[i-1][t][1] + prices[i],
                    dp[i-1][t][2] - prices[i]
                );
            }

            if (t > 0) {
                dp[i][t][1] = Math.max(dp[i-1][t][1], dp[i-1][t-1][0] - prices[i]);
                dp[i][t][2] = Math.max(dp[i-1][t][2], dp[i-1][t-1][0] + prices[i]);
            }
        }
    }

    let maxProfit = 0;
    for (let t = 0; t <= k; t++) {
        maxProfit = Math.max(maxProfit, dp[n-1][t][0]);
    }

    return maxProfit;
};
```

# Java:

```
class Solution {
    long[][][] dp;
    int[] prices;
    long mn = (long)-1e14;

    public long f(int i, int k, int state) {
        if (i == prices.length) {
            return (state == 0) ? 0 : mn;
        }
        if (dp[i][k][state] != mn) return dp[i][k][state];

        long p = prices[i];
        long profit = mn;

        // 1) do nothing today
        profit = Math.max(profit, f(i + 1, k, state));

        // 2) take action
        if (state == 0) {
            profit = Math.max(profit, f(i + 1, k, 1) - p); // buy
            profit = Math.max(profit, f(i + 1, k, 2) + p); // sell
        } else if (k > 0) {
            if (state == 1) {
                profit = Math.max(profit, f(i + 1, k - 1, 0) + p); // sell to end a buy-sell
            } else {
                profit = Math.max(profit, f(i + 1, k - 1, 0) - p); // buy to end a sell-buy
            }
        }

        return dp[i][k][state] = profit;
    }

    public long maximumProfit(int[] prices, int k) {
        this.prices = prices;
        int n = prices.length;
        dp = new long[n + 1][k + 1][3];
        for (long[][] twoD : dp)
            for (long[] oneD : twoD)
                Arrays.fill(oneD, mn);

        return f(0, k, 0);
    }
}
```