Approach #1: Dynamic Programming [Accepted]

**Intuition and Algorithm**

At the end of the i-th day, we maintain cash, the maximum profit we could have if we did not have a share of stock, and hold, the maximum profit we could have if we owned a share of stock.

To transition from the i-th day to the i+1-th day, we either sell our stock cash = max(cash, hold + prices[i] - fee) or buy a stock hold = max(hold, cash - prices[i]). At the end, we want to return cash. We can transform cash first without using temporary variables because selling and buying on the same day can't be better than just continuing to hold the stock.

**Python**

class Solution(object):

def maxProfit(self, prices, fee):

cash, hold = 0, -prices[0]

for i in range(1, len(prices)):

cash = max(cash, hold + prices[i] - fee)

hold = max(hold, cash - prices[i])

return cash

**Java**

class Solution {

public int maxProfit(int[] prices, int fee) {

int cash = 0, hold = -prices[0];

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

cash = Math.max(cash, hold + prices[i] - fee);

hold = Math.max(hold, cash - prices[i]);

}

return cash;

}

}

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*), where N*N* is the number of prices.
* Space Complexity: O(1)*O*(1), the space used by cash and hold