Best Time to Buy and Sell Stock III

1) Two pass prefix/suffix trick (very intuitive):

Idea: split the timeline at a cut i:
- Left side (days $0 \dots i$): best profit with one transaction.
- Right side (days $i+1 \dots n-1$): best profit with one transaction.

Add them and take the best over all cuts.

How to compute each side fast:
- Prefix: scan left → right, track the minimum price so far, and at each day record `best1[i] = max(best1[i-1], price[i] - min_so_far)`.
- Suffix: scan right → left, track the maximum price so far, and at each day record `best2[i] = max(best2[i+1], max_so_far - price[i])`.

Answer = max over i of `best1[i] + best2[i+1]` (also consider only one side if needed).

Complexity: $O(n)$ time, $O(n)$ space (can be reduced but this is simplest).

2) Tiny state machine (1 pass, $O(1)$ space):

Think of four states (profits):
- buy 1: best profit after first buy (negative).
- sell 1: best profit after first sell.
- buy 2: best profit after second buy (you "rebuy" using profit from sell 1).
- sell 2: best profit after second sell.

Update every day with price p:
- `buy1 = max(buy1, -p)`
- `sell1 = max(sell1, buy1 + p)`
- `buy2 = max(buy2, sell1 - p)`
- `sell2 = max(sell2, buy2 + p)`

Answer = `max(0, sell1, sell2)` (0 handles "never trade").
Why this works: each line either keeps the previous best or executes the

relevant action today.

3) DP table with up to $k = 2$ transactions (generalizable):

Let $dp[t][i]$ be max profit up to day i with at most t transactions.

Transition for $t \in \{1, 2\}$: $dp[t][i] = \max(dp[t][i-1], price[i] + \max_j(dp[t-1][j] - price[j]))$.

Maintain $best = \max_j(dp[t-1][j] - price[j])$ as you sweep i.

Answer = $dp[2][n-1]$.

Complexity: $O(n * k)$.

[Edge Notes:]

- If prices are non-increasing, result is 0.
- "At most two" includes doing 0 or 1 transaction; the formulas naturally handle that.
- Watch out for integer overflow only if your language uses narrow ints; these constraints are safe.