Best Time to Buy and Sell Stock IV

Pick the right model first:

1. If k is large: when k >= [n/2], it's equivalent to unlimited transactions.
   Trip: just sum every positive price difference (sum(max(0, prices[i]-prices[i-1]))).
   Why: you can buy/sell on every rise without exceeding k.
2. Otherwise (k is small): use DP with k layers.

The classic O(n·k) DP:

Think in terms of t completed transactions (cell actions) up to day i.

A) "Hold / Not-hold" state per transaction count:

Maintain arrays of size k+1:

- hold[t] = max profit after buying for the t-th transaction (i.e. currently holding a stock; you've completed t-1 sells).
- cash[t] = max profit after selling for the t-th time (i.e. not holding; you've completed t sells).

Initialize:

- cash[0] = 0, hold[0] = -∞ (can't hold before any buy).
- For t ≥ 1: hold[t] = -∞, cash[t] = -∞ initially (or a very small number)

For each price p:

```
for t = 1 ... k:
    hold[t] = max(hold[t], cash[t-1] - p)
    cash[t] = max(cash[t], hold[t] + p)
```

Answer: max_{t=0..k} cash[t] (usually cash[k] is enough since cash is nondecreasing in t).

Why it works: each layer t represents you're working on the t-th transaction; buying uses profit from cash[t-1], selling closes hold[t].

B) "Best-so-far" trick per layer:

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

Transition when you sweep i:

dp[t][i] = max(dp[t][i-1], prices[i] + best)

best = max(best, dp[t-1][i] - prices[i]).

Here best tracks the best "buy point with t-1 transactions done" as you go.

Space-opt: keep only two 1D arrays for dp[t-1][*] and dp[t][*].

Complexity:

- Time: O(n·k) (early-return to greedy when k >= n/2).
- Space: O(k) with the hold/cash form, or O(n) → O(1) per layer with rolling arrays.