Max Product Subarray DP

https://www.youtube.com/watch?v=lXVy6YWFcRM&t=52s

Here are the key ideas (concepts) you pick up while working through the Maximum Product Subarray:

1. **Brute-Force vs. Optimized Thinking**
   - Recognize that checking every subarray ($O(n^2)$) is too slow for large inputs, and look for a pattern to do it in one pass.
2. **Sign-Flipping with Negatives**
   - Observe that multiplying by a negative flips a large positive into a large negative and vice versa—so you must track both the "best" and the "worst" (most negative) products at each step.
3. **State Compression (Space-Optimized DP)**
   - Instead of storing a full DP array, you realize you only ever need two numbers (current max and min) to represent all earlier subarrays ending at the previous index.
4. **Handling Zeros as Reset Points**
   - Understand that a zero in the array "kills" any running product streak, so you reset your running products to the neutral multiplicative identity (1).
5. **Rolling Update with a Temporary Variable**
   - Learn why you need to stash the old `current_max×n` in `temp` before recomputing `current_max`, so you don't lose the value needed to update `current_min`.
6. **Greedy-Style Local Decisions**
   - See that at each index you make a local "best choice" (max of three candidates) and propagate it forward, yet this still leads to a globally optimal solution.
7. **Linear Time Complexity (O(n))**
   - Gain intuition for how a single pass with constant-time updates per element yields an O(n) algorithm.
8. **Constant Space Complexity (O(1))**
   - Appreciate that by only ever using a fixed handful of variables (no extra arrays), your memory usage stays constant no matter how big the input grows.

Putting these together, you end up with a neat, one-pass, constant-space dynamic-programming solution that handles positives, negatives, and zeros all in one go.