Coin Change

Core DP idea:

Let dp[a] be the fewest coins to make sum a (or ∞ if impossible).

- Base: dp[0] = 0.
- Transition: for each amount a from 1... amount dp[a] = min over coins c (dp[a-c] + 1), but only if a - c >= 0 and dp[a-c] is not ∞.

Answer is dp[amount] (return -1 if it's still ∞).

Why it works: you choose the last coin used; optimal substructure gives the recurrence.

Implementation tips:

- Use a big sentinel (e.g., amount + 1) to represent ∞.
- Initialise all dp[a] = ∞ except dp[0] = 0.
- Iterate amounts ascending so dp[a-c] is already computed.
- Optional: sort coins ascending - lets you break early when c > a (tiny speed win).

Complexity:

- Time: O(amount * #coins) - with amount <= 10^4 and #coins <= 12, this is fine.
- Space: O(amount).

Alternatives (when useful):

- BFS on amounts (each edge adds a coin): shortest path from 0 to amount. Good if you need "first time reached is optimal" reasoning.
- Top-down memo (DFS + memorization) with the same recurrence; be mindful of recursion depth and overlapping subproblems.