Documentation: Max Profit with at Most k Transactions

Problem Description:

- You are given an integer array prices where prices[i] represents the price of a given stock on the ith day, and an integer k, which defines the maximum number of transactions allowed. Each transaction consists of buying and selling one share of the stock. The goal is to find the maximum profit you can achieve by performing at most k transactions.

You are restricted to:

- Buy at most k times.
- Sell at most k times.
- You cannot engage in multiple transactions simultaneously, i.e., you must sell the stock before buying it again.

Input:

- An integer k: The maximum number of transactions allowed.
- An integer array prices[]: Represents the stock prices over several days.

Output:

- Return the maximum profit that can be made with at most k transactions.

Example 1:

- **Input:**
  - ➢ k = 2
  - ➢ prices = [2, 4, 1]
- **Output:** 2
- **Explanation:** Buy on day 1 (price = 2) and sell on day 2 (price = 4), resulting in a profit of 4 2 = 2.

Example 2:

- **Input:**
  - ➢ k = 2
  - ➢ prices = [3, 2, 6, 5, 0, 3]
- **Output:** 7
- **Explanation:**
  - ➢ Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6 2 = 4.
  - ➢ Buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3 0 = 3.
  - ➢ Total profit = 4 + 3 = 7.

Constraints:

- 1 <= k <= 100
- 1 <= prices.length <= 1000
- 0 <= prices[i] <= 1000

Key Concepts:

1. Transaction Definition:

- A single transaction consists of one buy and one sell action. The aim is to maximize the overall profit from at most k transactions.

2. Handling Large k:

- If k is large enough (i.e., k >= n//2, where n is the number of days), this means you can make an unlimited number of transactions. In this case, the problem reduces to a simpler version where you can make as many profitable transactions as possible by buying and selling stocks whenever there is a profit opportunity between consecutive days.

3. Dynamic Programming (DP):

- The problem is best solved using a dynamic programming (DP) approach.
- ***Define a DP table dp[t][i] where:***
  - ➢ t represents the number of transactions.
  - ➢ i represents the day.
  - ➢ dp[t][i] stores the maximum profit achievable up to day i with at most t transactions.
- The goal is to compute the value of dp[k][n-1], where n is the length of the prices array (i.e., the number of days).

Approach:

1. Base Case:

- If prices is empty, the maximum profit is 0 because no transactions can be made.
- If k is greater than or equal to n//2 (where n is the number of days), it's equivalent to having unlimited transactions because you can buy and sell as much as you want.

2. DP Table Initialization:

- Create a DP table dp[t][i] of size (k+1) x n, where each entry is initialized to 0. This table will help track the maximum profit achievable by making at most t transactions by day i.

3. DP Transition:

- ***For each transaction t (from 1 to k), and for each day i (from 1 to n-1):***
  - ➢ Sell on day i: Calculate the profit as the price on day i minus the price on some earlier day j, plus the profit from making one less transaction on day j.
  - ➢ Do nothing on day i: The profit is the same as on the previous day i-1 for transaction t.
- Use an auxiliary variable max_diff to store the maximum difference between dp[t-1][j] prices[j] up to a certain day i.

4. Result:

- The answer will be stored in dp[k][n-1], which represents the maximum profit achievable by making at most k transactions on all days.

Complexity Analysis:

- **Time Complexity:** O(k * n) where k is the number of transactions and n is the number of days. This ensures the solution is efficient for large inputs.

- **Space Complexity:** O(k * n) for storing the DP table, which tracks the maximum profit for each day and transaction count.

Edge Cases:

1. **Empty prices Array:**
   - If there are no prices provided, the maximum profit is 0 since no transactions can be made.

2. **Large k:**
   - When k is large enough (i.e., k >= n//2), the problem simplifies to finding the maximum possible profit with unlimited transactions.

3. **Single Day:**
   - If there is only one day of stock prices, no transactions can be made, so the profit is 0.

Limitations:

- The solution assumes that you cannot engage in multiple transactions at the same time (i.e., you must sell the stock before buying it again).
- It focuses on achieving the maximum profit by optimizing the number of transactions allowed, but it may not work effectively for problems with other restrictions, such as transaction fees or cooldown periods.

Conclusion:

- This dynamic programming approach efficiently solves the problem of maximizing stock trading profits with a limited number of transactions. The method is optimal and handles edge cases like large k values and empty input arrays, ensuring correct results for a wide range of test cases.