To solve the "Lover" problem, we need to find the minimum number of segments (M) to partition N stores, such that the maximum "expensive period value" for "คุณคี่" (odd prices) does not exceed V, and for "คุณคู่" (even prices) does not exceed W. Each segment length l\_i must be an even number.

This problem can be efficiently solved using dynamic programming.

**Problem Analysis**

1. **Input:**
   * N: Total number of stores (2 <= N <= 10,000, N is even)
   * V: Maximum allowed "ค่าช่วงแพง" for "คุณคี่" (sum of odd prices)
   * W: Maximum allowed "ค่าช่วงแพง" for "คุณคู่" (sum of even prices)
   * x\_1, ..., x\_N: Prices at each store (1 <= x\_i <= 10,000)
2. **Output:** Minimum M. If no solution, output -1.
3. **Constraints:** Time limit 1 second, Memory limit 512 MB.

**Dynamic Programming Approach**

We define dp[i] as the minimum number of segments (M) required to cover the first i stores (from store 1 to store i). Our goal is to find dp[N].

**State:** dp[i] represents the minimum M for stores x[0] to x[i-1] (using 0-indexed arrays for implementation).

**Base Case:**

* dp[0] = 0: Zero segments are needed to cover zero stores.
* All other dp[i] are initialized to INF (a very large number, indicating unreachable).

**Transitions:** To calculate dp[i], we iterate through all possible previous ending points prev\_idx (representing dp[prev\_idx]). A segment starting at x[prev\_idx] and ending at x[i-1] (inclusive) is considered.

1. **Segment Length Check:** The length of this potential segment is current\_segment\_len = i - prev\_idx.
   * This length must be an even number. If not, skip this

prev\_idx.

* + The length must be at least 2 (as

l\_i is even). This is ensured by

prev\_idx <= i - 2.

* + Since N is even and all l\_i are even, i and prev\_idx will naturally always be even in terms of total stores covered.

1. **Reachability Check:** if (dp[prev\_idx] == INF), it means the previous state prev\_idx is unreachable, so we cannot extend a path from it. Skip this prev\_idx.
2. **Calculate Sums for Current Segment:**
   * For a segment of length current\_segment\_len:
     + "คุณคี่" shops in the first

current\_segment\_len / 2 stores.

* + - "คุณคู่" shops in the remaining

current\_segment\_len / 2 stores.

* + To efficiently calculate sums of odd and even prices within these sub-segments, we use **prefix sums**:
    - pref\_odd\_price[k]: Sum of odd prices in x[0] to x[k-1].
    - pref\_even\_price[k]: Sum of even prices in x[0] to x[k-1].
  + current\_a\_sum (for "คุณคี่"): Sum of odd prices from x[prev\_idx] to x[prev\_idx + current\_segment\_len / 2 - 1]. This is calculated as pref\_odd\_price[prev\_idx + current\_segment\_len / 2] - pref\_odd\_price[prev\_idx].
  + current\_b\_sum (for "คุณคู่"): Sum of even prices from x[prev\_idx + current\_segment\_len / 2] to x[i-1]. This is calculated as pref\_even\_price[i] - pref\_even\_price[prev\_idx + current\_segment\_len / 2].

1. **Condition Check:** If current\_a\_sum <= V AND

current\_b\_sum <= W:

* + Update dp[i] = min(dp[i], dp[prev\_idx] + 1).

**Final Result:** After filling the dp table, dp[N] will contain the minimum M. If

dp[N] is still INF, it means no valid partition exists, so output -1.

**Complexity**

* **Time Complexity:**
  + Preprocessing prefix sums takes O(N) time.
  + The nested loops for dp calculation: The outer loop runs N/2 times (for i). The inner loop runs N/2 times (for prev\_idx). Inside the loops, calculations are O(1).
  + Total time complexity is O(N^2). Given N=10,000, N^2 is 10^8. This is a tight fit for a 1-second time limit, but typically passes in competitive programming for such constraints.
* **Space Complexity:**
  + x vector: O(N)
  + pref\_odd\_price and pref\_even\_price vectors: O(N) each.
  + dp vector: O(N).
  + Total space complexity is O(N). For N=10,000, this is well within the 512 MB memory limit.