📘 Target Sum – Complete Documentation

📑 Table of Contents

1. 📌 Problem Statement

You are given an array of integers nums and an integer target. Your task is to build expressions by adding either a '+' or '-' sign before each number in the array, and calculate how many different expressions evaluate to the target.

🔢 Example:

- Input: nums = [1,1,1,1,1], target = 3
- Output: 5

Constraints:

- 1 <= nums.length <= 20
- 0 <= nums[i] <= 1000
- -1000 <= target <= 1000

2. 💡 Intuition

Instead of brute-forcing all 2^n combinations of '+' and '-', we can reframe the problem using a mathematical transformation that connects it to the Subset Sum problem—a classic dynamic programming question.

3. □ Key Observations

- Let the total sum of all numbers be S.
- If we split numbers into two groups:
    - P → numbers with +
    - N → numbers with -
- Then:
    - P - N = target
    - P + N = S
- Adding these two:
    - 2P = target + S
    - P = (target + S) / 2

So, the problem reduces to finding how many subsets of nums sum to P.

4. 🛠 Approach

- Compute the total sum S of the array.
- Check if (target + S) is even and P = (target + S) / 2 is valid.
- Use 1D Dynamic Programming to count the number of subsets that sum to P.

□ DP Array:

- dp[i] will store the number of ways to get a sum of i.
- Start with dp[0] = 1 (1 way to make sum 0: use nothing).
- For each number in nums, iterate backwards to update dp.

5. ⚠ Edge Cases

- If (target + total_sum) is odd, return 0 → subset sum is not an integer.
- If abs(target) > total_sum, return 0 → target not achievable.
- If nums contains zeros, multiple combinations may contribute to same sum.

6. 📊 Complexity Analysis

⏱ Time Complexity:

- O(n * subset_sum) where n is the length of the array, and subset_sum = (target + total_sum) / 2

💾 Space Complexity:

- O(subset_sum) due to 1D DP array

7. 🔁 Alternative Approaches

- Recursive with Memoization:
  - Recursively add and subtract current number
  - Use a dictionary to memoize overlapping states
- Brute Force (Not Recommended):
  - Try all 2^n combinations of '+' and '-'
  - Inefficient for n > 15

8. 🧪 Test Cases

✅ Example 1:

Input: nums = [1,1,1,1,1], target = 3

Output: 5

✅ Example 2:

Input: nums = [1], target = 1
Output: 1

✅ Edge Case:

Input: nums = [0,0,0,0,0,0,0,0,1], target = 1
Output: 256

9. □ Final Thoughts

- Transforming the problem into a Subset Sum question makes it significantly more efficient.
- This technique is commonly useful in problems involving binary choices (like '+' or '-').
- Knowing how to reframe problems mathematically can unlock easier, optimized solutions.