📘 Subarray Sum Equals K Documentation

1. Problem Statement

Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals to k.

- A subarray is a contiguous part of the array.
- Constraints:
  - 1 <= nums.length <= 2 × 10^4
  - -1000 <= nums[i] <= 1000
  - -10^7 <= k <= 10^7

2. Intuition

Using prefix sums, we can track running totals of the array and check whether a subarray sum equals k without recalculating each subarray from scratch.

3. Key Observations

- If sum(i, j) is the sum of a subarray nums[i..j], then:

  sum(i,j)=prefixSum[j]−prefixSum[i−1]sum(i, j) = prefixSum[j] - prefixSum[i-1]

- We can use a hash map to keep track of all prefix sums we've seen so far.
- If current_sum - k exists in the map, we found a valid subarray ending at the current index.

4. Approach

- Use a hashmap to store the frequency of each prefix sum.
- Initialize prefix_sums[0] = 1 to handle the case when a subarray starts at index 0.
- Iterate through the array:

    - Accumulate the current_sum.
    - Check if current_sum - k exists in the map.
    - If it does, it contributes prefix_sums[current_sum - k] to the result.
    - Update the count of current_sum in the hashmap.

5. Edge Cases

- nums contains negative numbers — handled by the prefix sum approach.
- k is zero — will check for subarrays that sum to exactly 0.
- Repeated prefix sums — the map keeps count of how many times each has occurred.

6. Complexity Analysis

Time Complexity

- O(n) — One pass through the array.

Space Complexity

- O(n) — In worst case, each prefix sum is unique and stored.

7. Alternative Approaches

Brute Force (Inefficient)

- Generate all subarrays and check if their sum equals k.
- Time Complexity: $O(n^2)$
- Space Complexity: O(1)

Prefix Sum Array (Better, but not optimal)

- Build a prefix sum array, then use nested loops to compute subarray sums.
- Time Complexity: $O(n^2)$
- Space Complexity: O(n)

8. Algorithm

- Initialize count = 0, current_sum = 0
- Create a hashmap prefix_sums with initial value {0: 1}
- For each number in nums:
    - Add number to current_sum
    - If current_sum - k in prefix_sums, increment count
    - Update prefix_sums[current_sum] += 1
- Return count

9. Test Cases

Test Case	Input	Output	Explanation
1	nums = [1,1,1], k = 2	2	Subarrays: [1,1] at (0,1) and (1,2)
2	nums = [1,2,3], k = 3	2	Subarrays: [1,2], [3]
3	nums = [1,-1,1,1], k = 2	2	Subarrays: [1,-1,1,1] and [1,1]
4	nums = [0,0,0,0], k = 0	10	All subarrays of any size add to 0

10. Final Thoughts

- This problem is a classic example of how hash maps + prefix sums can reduce time complexity from $O(n^2)$ to $O(n)$.
- It's frequently asked in interviews and tests understanding of subarray techniques.
- Avoid brute-force approaches on large input sizes.