# Problem Understanding

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

### Example:

Input: nums = [1,1,1], k = 2

Output: 2

Explanation: Subarrays [1,1] at indices (0,1) and (1,2)

### Key Observations:

* We're looking for **contiguous subarrays**.
* Brute force would check every subarray sum — too slow for large arrays.
* We can optimize using **prefix sum** + **HashMap**.

## Optimized Java Solution (Prefix Sum + HashMap)

class Solution {

public int subarraySum(int[] nums, int k) {

int count = 0, sum = 0;

Map<Integer, Integer> prefixSumFreq = new HashMap<>();

prefixSumFreq.put(0, 1); // base case: prefix sum 0 occurs once

for (int num : nums) {

sum += num;

// Check if (sum - k) has occurred before

if (prefixSumFreq.containsKey(sum - k)) {

count += prefixSumFreq.get(sum - k);

}

// Update frequency of current prefix sum

prefixSumFreq.put(sum, prefixSumFreq.getOrDefault(sum, 0) + 1);

}

return count;

}

}

# Dry Run Using Table

### Input:

nums = [1, 2, 3], k = 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | num | sum | sum-k | prefixSumFreq (before) | count change | Explanation |
| 0 | 1 | 1 | -2 | {0=1} | 0 | No match |
| 1 | 2 | 3 | 0 | {0=1, 1=1} | +1 | sum-k=0 seen → +1 |
| 2 | 3 | 6 | 3 | {0=1, 1=1, 3=1} | +1 | sum-k=3 seen → +1 |

✅ Final count = **2**  
Subarrays: [1,2] and [3]

## Time & Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | **O(n)** |
| Space | **O(n)** (prefixSum map) |

* We scan the array once (O(n))
* At each step, we do constant-time operations with HashMap.

# Alternate Approaches

### ****Brute Force (Nested Loops)****

for (int i = 0; i < n; i++) {

sum = 0;

for (int j = i; j < n; j++) {

sum += nums[j];

if (sum == k) count++;

}

}

* Time: **O(n²)**
* Space: **O(1)**
* ❌ Slow for large inputs

### ****Prefix Sum + Array (not hashmap)****

* Build prefix sum array
* For each i, find j where prefix[j] - prefix[i] == k
* Time: **O(n²)**
* Space: **O(n)**
* ❌ Still too slow

### ****Prefix Sum + HashMap (Best Approach)****

* Track cumulative sums
* Use map to count occurrences of sum - k
* Time: **O(n)**, Space: **O(n)**

### Quick Test Cases

nums = [1,1,1], k = 2 → 2

nums = [1,2,3], k = 3 → 2

nums = [3,4,7,2,-3,1,4,2], k = 7 → 4

nums = [1,-1,0], k = 0 → 3