# **Leetcode 325 – Maximum Size Subarray Sum Equals k**

## Problem Understanding

Given an integer array nums and an integer k, return the **maximum length** of a subarray that sums to k.

* The array **can contain negative numbers**.
* Return 0 if no such subarray exists.

🧠 Key Insight:

We're not counting all subarrays that sum to k, but finding the **longest one**.

## Optimized Java Solution (Prefix Sum + HashMap)

We use:

* A HashMap to store the **first index** at which each prefix sum occurs.
* At each step, we check if the prefix sum up to index i minus k occurred earlier.
* If so, we can form a valid subarray ending at i.

class Solution {

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

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

int sum = 0, maxLen = 0;

prefixIndex.put(0, -1); // base case for exact match

for (int i = 0; i < nums.length; i++) {

sum += nums[i];

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

int prevIndex = prefixIndex.get(sum - k);

maxLen = Math.max(maxLen, i - prevIndex);

}

// Store first occurrence of prefix sum only

prefixIndex.putIfAbsent(sum, i);

}

return maxLen;

}

}

## Dry Run Using Table

Let’s dry run this example:

### Input:

nums = [1, -1, 5, -2, 3], k = 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | num | sum | sum-k | map before | Action | maxLen |
| - | - | 0 | - | {0: -1} | Initialize base case | 0 |
| 0 | 1 | 1 | -2 | {0: -1} | -2 not in map | 0 |
| 1 | -1 | 0 | -3 | {0: -1, 1: 0} | -3 not in map | 0 |
| 2 | 5 | 5 | 2 | {0: -1, 1: 0, 0: 1} | 2 not in map | 0 |
| 3 | -2 | 3 | 0 | {0: -1, 1: 0, 5: 2} | 0 in map → i - (-1) = 4 → update maxLen | 4 |
| 4 | 3 | 6 | 3 | {…} | 3 in map → i - 3 = 1 | 4 |

✅ Final Answer: maxLen = 4  
(Subarray is [1, -1, 5, -2])

## Time and Space Complexity

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

* Time: Single pass through nums, constant-time HashMap operations.
* Space: Storing prefix sums in HashMap.

## Alternate Approaches

### 1. ****Brute Force (O(n²))****

* Try all subarrays and compute their sums.
* Track the max length where sum equals k.
* Inefficient for large inputs.

### 2. ****Sliding Window?****

❌ Not applicable here because:

* Works only for **positive numbers**.
* Negative numbers break the "expand/right and shrink/left" logic.