

LeetCode 560 - Subarray Sum Equals K

Question Summary:

Count how many contiguous subarrays have sum = k.

Dry Run Example:

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

Subarrays:

[1] -> 1

[1,2] -> 3 (valid)

[1,2,3] -> 6

[2] -> 2

[2,3] -> 5

[3] -> 3 (valid)

Total = 2 subarrays

Approach 1: Brute Force ($O(n^2)$)

Try all subarrays and check if sum = k.

Java:

```
class Solution {  
    public int subarraySum(int[] nums, int k) {  
        int count = 0;  
        for (int i = 0; i < nums.length; i++) {  
            int sum = 0;  
            for (int j = i; j < nums.length; j++) {  
                sum += nums[j];  
                if (sum == k) count++;  
            }  
        }  
        return count;  
    }  
}
```

```
    }
    return count;
}
}
```

C++:

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int count = 0;
        for(int i = 0; i < nums.size(); i++){
            int sum = 0;
            for(int j = i; j < nums.size(); j++){
                sum += nums[j];
                if(sum == k) count++;
            }
        }
        return count;
    }
};
```

Python:

```
class Solution:
    def subarraySum(self, nums, k):
        count = 0
        for i in range(len(nums)):
            s = 0
            for j in range(i, len(nums)):
                s += nums[j]
                if s == k:
                    count += 1
        return count
```

Approach 2: Prefix Sum + HashMap ($O(n)$)

Idea:

Use prefix sums and a hashmap to store previous sums.

Dry Run ($\text{nums} = [1,1,1]$, $k = 2$):

```
prefix = {0:1}
sum=1 -> need -1 -> not found
sum=2 -> need 0 -> found
sum=3 -> need 1 -> found
Result = 2
```

Java:

```
import java.util.*;
class Solution {
    public int subarraySum(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);
        int sum = 0, count = 0;
        for (int num : nums) {
            sum += num;
            if (map.containsKey(sum - k))
                count += map.get(sum - k);
            map.put(sum, map.getOrDefault(sum, 0) + 1);
        }
        return count;
    }
}
```

C++:

```
class Solution {
public:
```

```

int subarraySum(vector<int>& nums, int k) {
    unordered_map<int,int> mp;
    mp[0] = 1;
    int sum = 0, count = 0;
    for(int x : nums){
        sum += x;
        if(mp.count(sum - k))
            count += mp[sum - k];
        mp[sum]++;
    }
    return count;
}

```

Python:

```

class Solution:
    def subarraySum(self, nums, k):
        prefix = {0: 1}
        s = 0
        count = 0
        for x in nums:
            s += x
            if s-k in prefix:
                count += prefix[s-k]
            prefix[s] = prefix.get(s, 0) + 1
        return count

```

Time Complexity:

Method	Time	Space
Brute Force	$O(n^2)$	$O(1)$
Prefix + HashMap	$O(n)$	$O(n)$

Final Notes:

- Best approach = Prefix Sum + HashMap
- Handles negative numbers too
- Efficient and elegant solution