

Binary Subarrays with Sum

Count subarrays with EXACT sum — the atMost trick ■

● MEDIUM

■ Hook

Count subarrays with sum exactly equal to **goal**. In a binary array (only 0s and 1s). Direct sliding window doesn't work easily for exact sums — but there's an elegant trick: **exact(k) = atMost(k) - atMost(k-1)**. This transforms an exact-count problem into two at-most problems.

■ The atMost Trick — Core Insight

Count subarrays with $\text{sum} \leq \text{goal}$. Then count subarrays with $\text{sum} \leq \text{goal}-1$. The difference gives exactly those with $\text{sum} == \text{goal}$. Why does this work? Because subarrays with $\text{sum} \leq k$ includes those with $\text{sum} = k, k-1, k-2, \dots, 0$. Subtracting $\leq k-1$ leaves only those with $\text{sum} = k$.

```
exact(goal) = atMost(goal) - atMost(goal - 1)

atMost(k): count subarrays where sum ≤ k
→ Standard sliding window! shrink when sum > k
→ At each right, valid windows ending at right = (right - left + 1)
```

■ Visual — [1,0,1,0,1], goal=2

```
atMost(2): sum ≤ 2
right=0: [1]          sum=1 ≤ 2 ✓ count += 1 (total=1)
right=1: [1,0]        sum=1 ≤ 2 ✓ count += 2 (total=3)
right=2: [1,0,1]      sum=2 ≤ 2 ✓ count += 3 (total=6)
right=3: [1,0,1,0]    sum=2 ≤ 2 ✓ count += 4 (total=10)
right=4: [1,0,1,0,1]  sum=3 > 2 ✗ shrink: left=1
               [0,1,0,1] sum=2 ≤ 2 ✓ count += 4 (total=14)

atMost(1): sum ≤ 1
(same logic) → total = 9

ANSWER = atMost(2) - atMost(1) = 14 - 9 = 5
Verify: [1,0,1], [0,1,0,1]... = 5 subarrays with sum=2 ✓
```

■ Code Implementation (Java)

```

public class BinarySubarraysWithSum {

    public int numSubarraysWithSum(int[] nums, int goal) {
        return atMost(nums, goal) - atMost(nums, goal - 1);
    }

    private int atMost(int[] nums, int k) {
        if (k < 0) return 0; // edge case: goal=0
        int left = 0, sum = 0, count = 0;

        for (int right = 0; right < nums.length; right++) {
            sum += nums[right];

            while (sum > k) {
                sum -= nums[left++]; // shrink from left
            }

            count += (right - left + 1); // all windows ending at right
        }

        return count;
    }
}

```

■ Time & Space Complexity

■ TIME COMPLEXITY

$O(n)$

■ SPACE COMPLEXITY

$O(1)$

Two passes of $O(n)$ each = $O(2n) = O(n)$. Constant space — no extra data structures.

■ Common Mistakes

■ Not handling goal=0 edge case

`atMost(nums, -1)` must return 0. Add 'if ($k < 0$) return 0' at the start of `atMost`. Without this, the while loop never exits and you get wrong answers or infinite loops.

■ Practice Problems

#	Problem	Difficulty	Pattern
1	Binary Subarrays With Sum (LeetCode 930)	Medium	atMost trick
2	Count Number of Nice Subarrays (LC 1248)	Medium	atMost trick

3	Subarray Product Less Than K (LC 713)	Medium	atMost directly
4	Number of Substrings Containing All 3 Chars (LC 1358)	Medium	SW + atMost
5	Subarrays with K Different Integers (LC 992)	Hard	atMost trick

■ Quick Revision

- ✓ $\text{exact}(\text{goal}) = \text{atMost}(\text{goal}) - \text{atMost}(\text{goal} - 1)$
- ✓ $\text{atMost}(k)$: count subarrays with $\text{sum} \leq k$ using sliding window
- ✓ For each valid window, $\text{count} += (\text{right} - \text{left} + 1)$
- ✓ Handle $k < 0$ edge case $\rightarrow$ return 0
- ✓ Works on binary arrays; generalizes to non-negative arrays
- ✓ Time: $O(n)$ | Space: $O(1)$

■ MENTAL MODEL — Remember This Forever

Think of it like sets: `subarrays_with_sum_≤k` is a superset of `subarrays_with_sum_≤(k-1)`. The difference between these two sets is exactly `subarrays_with_sum_==k`. This `atMost` subtraction trick shows up in dozens of counting window problems.