3. Variable size window

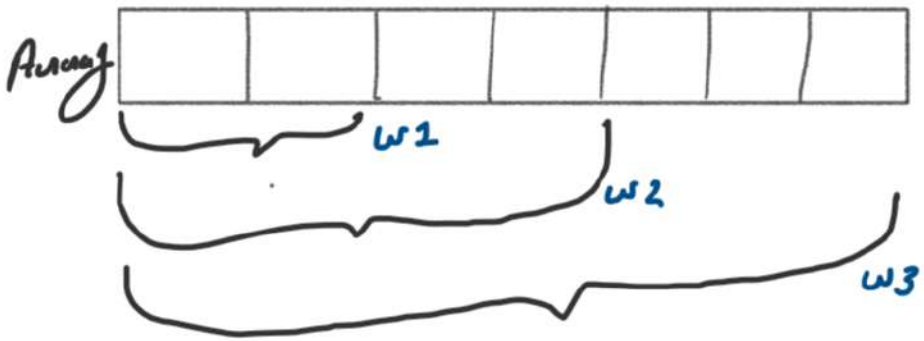
Problem 1: Minimum Size Subarray Sum (Leetcode-209)
Problem 2: Binary Subarrays With Sum (Leetcode-930)

Variable Size Sliding Window Approach:

Note: *Determine window size*
Variable Size Window

Step 1: *Start by initializing the* ***start*** *and* ***end*** *pointers to the first element of the sequence or array*

Step 2: *Find the answer*
- ***Valid Ans:*** *minimize start++*
- ***Store Ans:*** *Perform the required calculations or operations to obtain the answer*
- ***Invalid Ans:*** *explore end++*



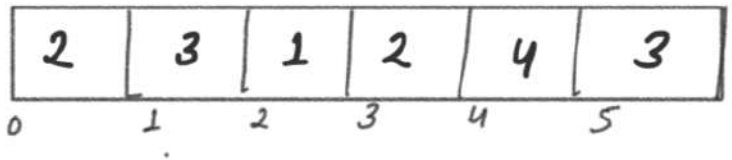
window	size
w1	2
w2	4
w3	7

} diffrent sizes of each windows

📂 1. Minimum Size Subarray Sum (Leetcode-209)

Problem Statement:
*Given an array of positive integers **nums** and a positive integer **target**, return the **minimal length** of a subarray whose sum is greater than or equal to **target**. If there is no such subarray, return **0** instead.*

Example 1:
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.

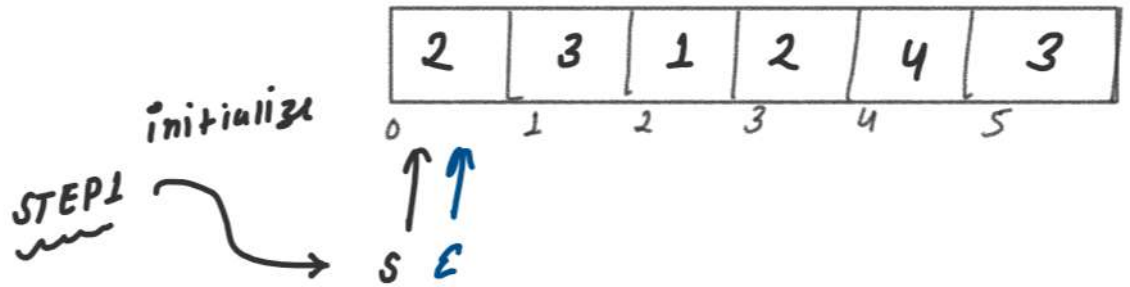


possible subarrays = 2^6

Example 2:
Input: target = 4, nums = [1,4,4]
Output: 1

Example 3:
Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0

Target = 7



Logic

STEP2 Find Ans

if (windowSum >= target)

start ++

Else

end --

why this logic ?

↳ return the smallest window length jo correct Ans deti.

DRY RUN

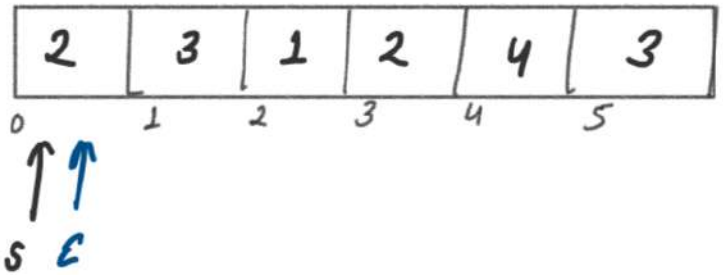
Target = 7

S	E	Sum >= Target			
0	0	2	7 = 7	X	End ++
0	1	5	7 = 7	X	End ++
0	2	6	7 = 7	X	End ++
0	3	7	7 = 7	✓	start ++
1	3	5	7 = 7	X	End ++
1	4	10	7 = 7	✓	Start ++
2	4	7	7 = 7	✓	Start ++
3	4	6	7 = 7	X	End ++
3	5	9	7 = 7	✓	Start ++
4	5	7	7 = 7	✓	Start ++
5	5	3	7 = 7	X	End ++
5	6				

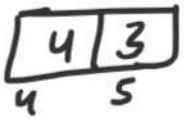
STOP

End < size

6 < 6 X



ANS



length = 2

output

```cpp
// Problem 01: Minimum Size Subarray Sum (Leetcode-209)

class Solution {
public:
    int minSubArrayLen(int target, vector<int>& nums) {
        // Step 1: Initializing the start and end
        int start = 0;
        int end = 0;
        int windowLength = INT_MAX;
        int windowSum = 0;

        // Step 2: Find the answer
        while(end < nums.size()){
            // Store the value in windowSum
            windowSum = windowSum + nums[end];

            while(windowSum >= target){
                // Minimize the window
                if(windowSum >= target){
                    int currLength = end - start + 1;
                    windowLength = min(windowLength, currLength);
                }
                // Current element, pointed by start, should be remove from the new window
                windowSum = windowSum - nums[start];
                start++;
            }

            // Loop se bahar tabhi aa skte hai jab windowSize < target hoga
            // Explore the window
            end++;
        }

        // Edge case: me hamesha ynha galti karta hu
        if(windowLength == INT_MAX){
            // Apply for example 3
            return 0;
        }
        return windowLength;
    }
};
```

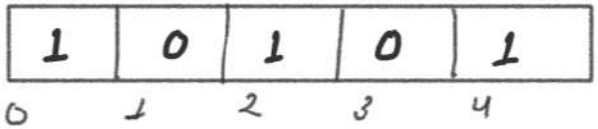
Time Complexity: O(N)
Space Complexity: O(1)

2. Binary Subarrays with Sum (Leetcode-930)

Problem Statement:
Given a binary array **nums** and an integer **goal**, return the number of non-empty subarrays with a sum **goal**.
A **subarray** is a contiguous part of the array.

Example 1:
Input: nums = [1,0,1,0,1], goal = 2
Output: 4

Example 2:
Input: nums = [0,0,0,0,0], goal = 0
Output: 15



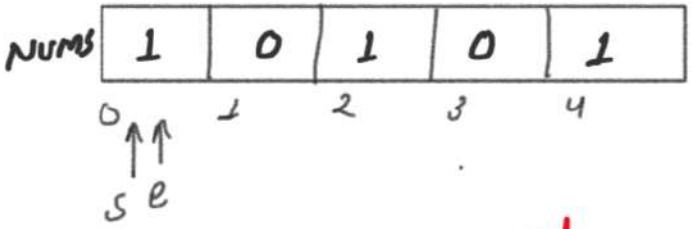
Initial State

start = 0
end = 0
Prefix Zero = 0
sum = 0
count = 0

Explanation: The 4 subarrays are
[1,0,1,_,_] = 2
[1,0,1,0,_] = 2
[_,0,1,0,1] = 2
[_,_,1,0,1] = 2

DRY RUN

goal = 2



1

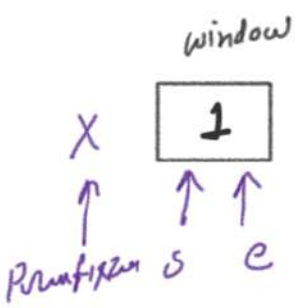
start = 0

End = 0

PrefixZero = 0

Sum = 0

Count = 0



sum += num[e]

sum = 0 + 1

= 1

Invalid

(sum < goal)

1 < 2

End++

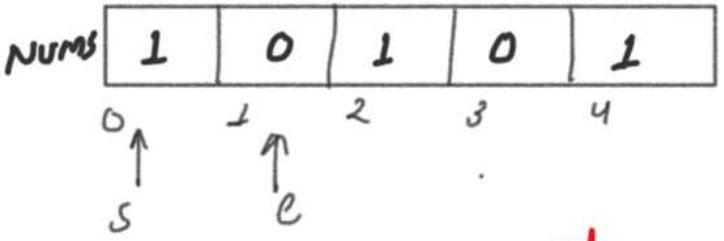
Unification

(sum == goal) X

1 2

count += PrefixZero + 1;

goal = 2



start = 0

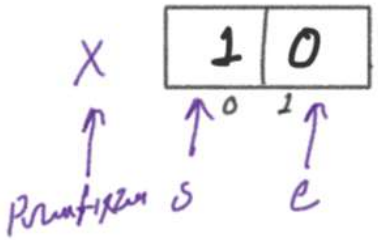
end = 1

prefixZero = 0

sum = 1

count = 0

window 2



sum += NUM[e]

sum = 1 + 0

= 1

Invalid

(sum < goal)

1 < 2

End++

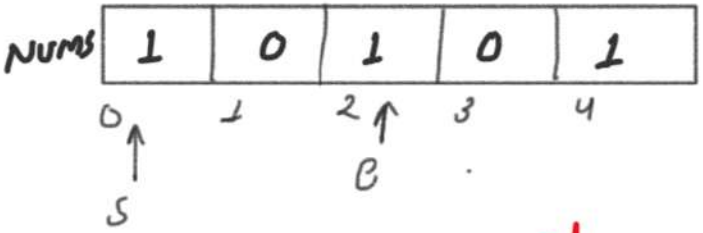
Unification

(sum == goal) X

1 2

count += prefix + 1;

goal = 2



3

start = 0

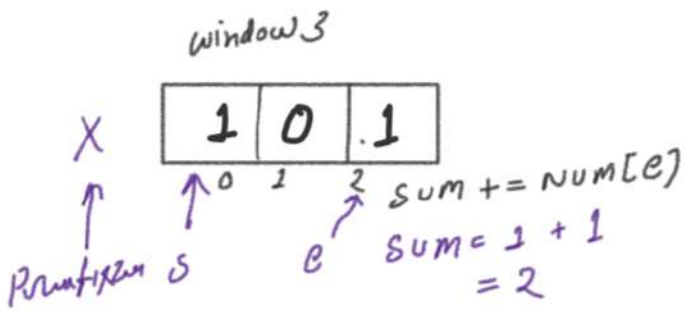
end = 2

PrefixZero = 0

sum = 1

count = 0

window 3



valid

(sum = goal)

2 < 2

End++

Unification

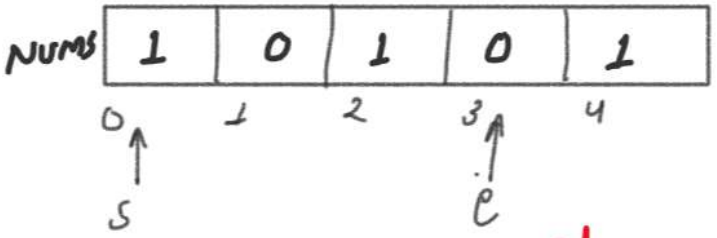
(sum == goal) X

count += PrefixZero + 1;

= 0 + 0 + 1

count = 1

goal = 2



4

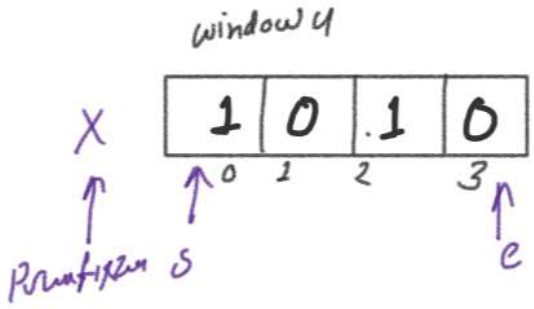
start = 0

end = 3

PrefixZero = 0

sum = 2

count = 1



sum += num[e]

sum = 2 + 0

= 2

valid

(sum == goal)

2 < 2

End++

Unification

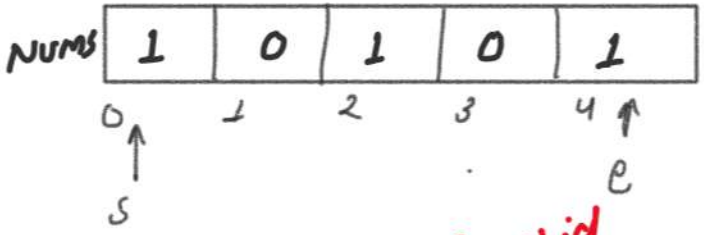
(sum == goal) X

count += Prefix + 1;

= 1 + 0 + 1

count = 2

goal = 2



S

start = 0

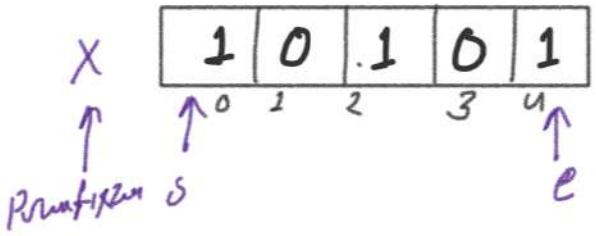
End = 5

PrefixZero = 0

sum = 2

Count = 2

window S



sum += num[e]

sum = 2 + 1

= 3

Invalid

(sum > goal)

3 > 2

if(num[s] == 1) {

1 1

PrefixZero -= 0

3

else {

PrefixZero += 1

}

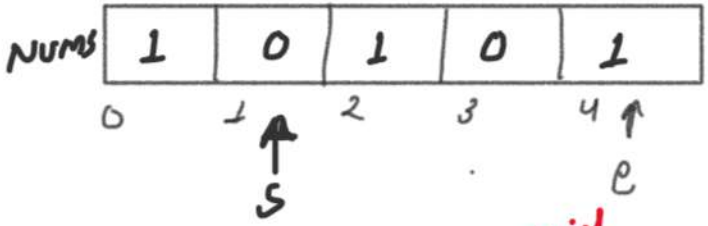
sum = sum - num[s]; ↳ s-1

= 2

s++; // minimize

↳ s = 1

goal = 2



S

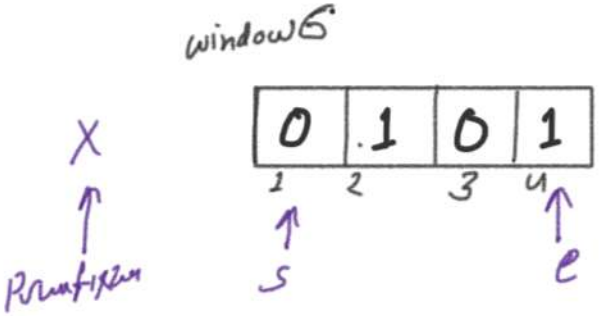
start = 1

end = 5

PrefixZero = 0

sum = 2

count = 2



valid

(NUMS[S] == 0)

0 0

if(NUM[S] == 1) {

0 == 1 ✗

PrefixZero = 0

}

else {

PrefixZero += 1 → Pre = 1

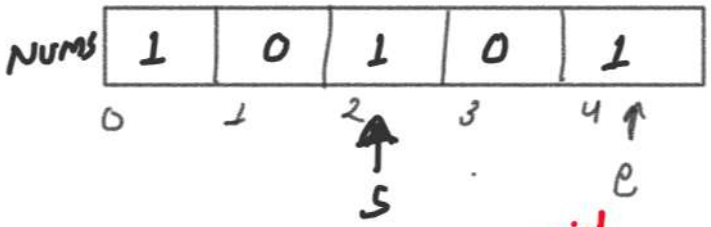
}

sum = sum - NUM[S]; → 2 - 0 = 2

S++; // minimize

→ S = 2

goal = 2



S

start = 2

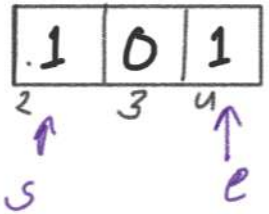
end = 5

prefixZero = 1

sum = 2

count = 2

window



valid

(sum == target)

2 2

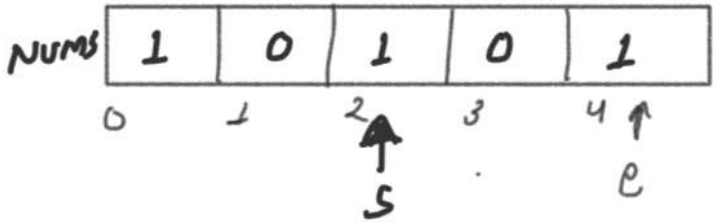
↳ count = count + prefix + 1

= 2 + 1 + 1

= 4

End ++

goal = 2



S

start = 2

end = 6 stop

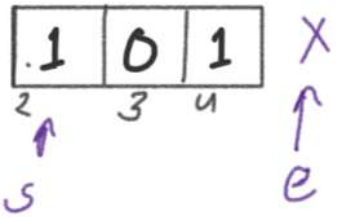
PrefixZero = 1

sum = 2

Count = 4

output

windowB

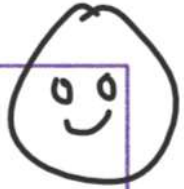


e > size

size = 5

if(NUM[s] == 0 ||
sum > goal)
↳ s++

if(sum <= goal)
↳ e++



```cpp
// Problem 02: Binary Subarrays With Sum (Leetcode-930)

class Solution {
public:
    int numSubarraysWithSum(vector<int>& nums, int goal) {
        // Step 1: Initializing the start and end
        int start = 0;
        int end = 0;
        int windowSum = 0;
        int prefixZero = 0;
        int count = 0;

        // Step 2: Find the answer
        while(end < nums.size()){
            // Store the value in windowSum
            windowSum = windowSum + nums[end];

            // Minimise the window
            while(start < end && (windowSum > goal || nums[start] == 0)){
                if(nums[start] == 1){
                    prefixZero = 0;
                }
                else{
                    prefixZero += 1;
                }
                // Current element, pointed by start, should be remove from the new window
                windowSum = windowSum - nums[start];
                start++;
            }
            // Verification
            if(windowSum == goal){
                count += prefixZero + 1;
            }
            // Explore the window
            end++;
        }
        return count;
    }
};
```

Time Complexity: O(N)
Space Complexity: O(1)