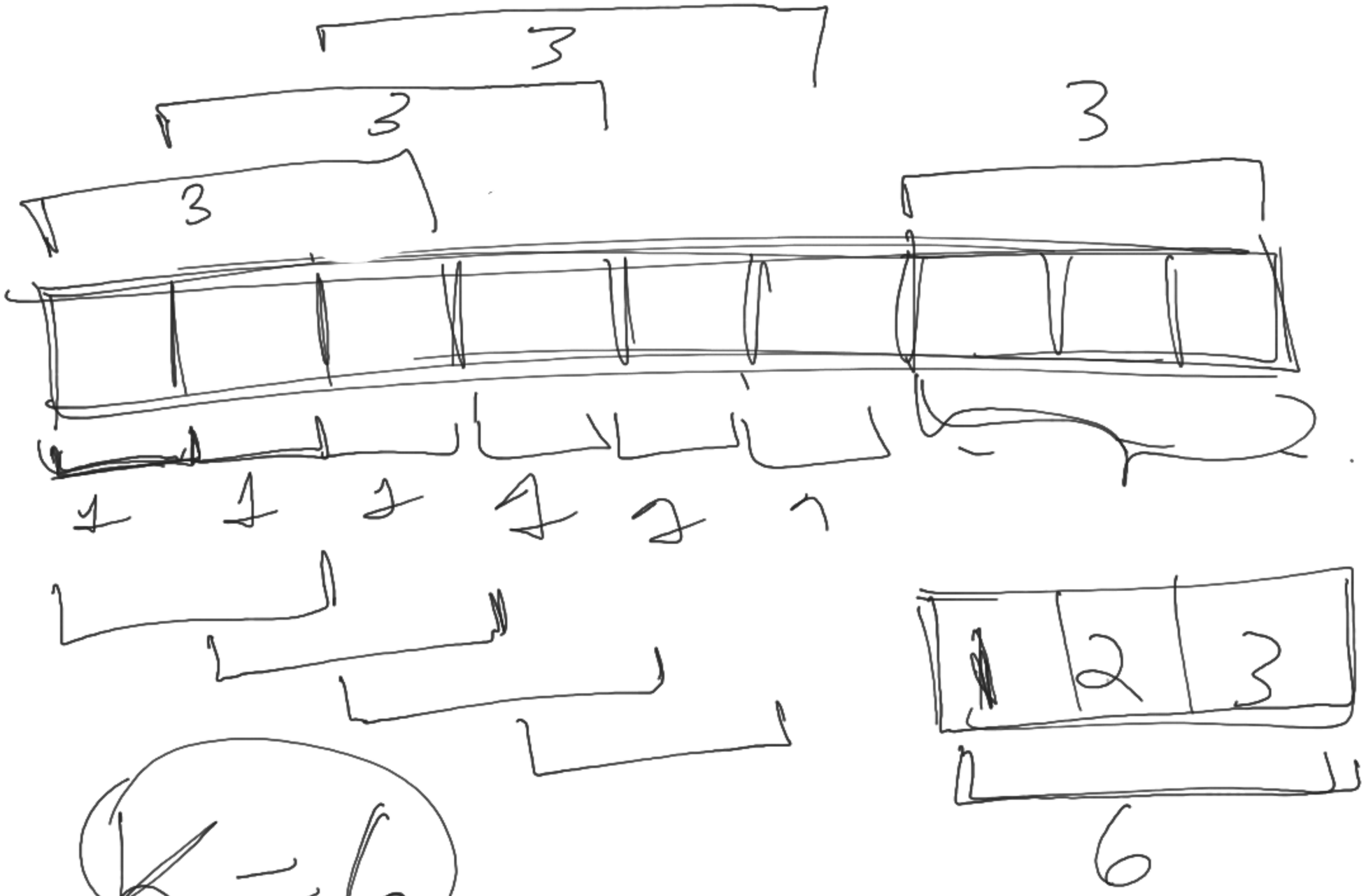
$O(n)$

The cost


3
3
3
3
1 1 2 1 2 1
1 2 3
6
K=6

K

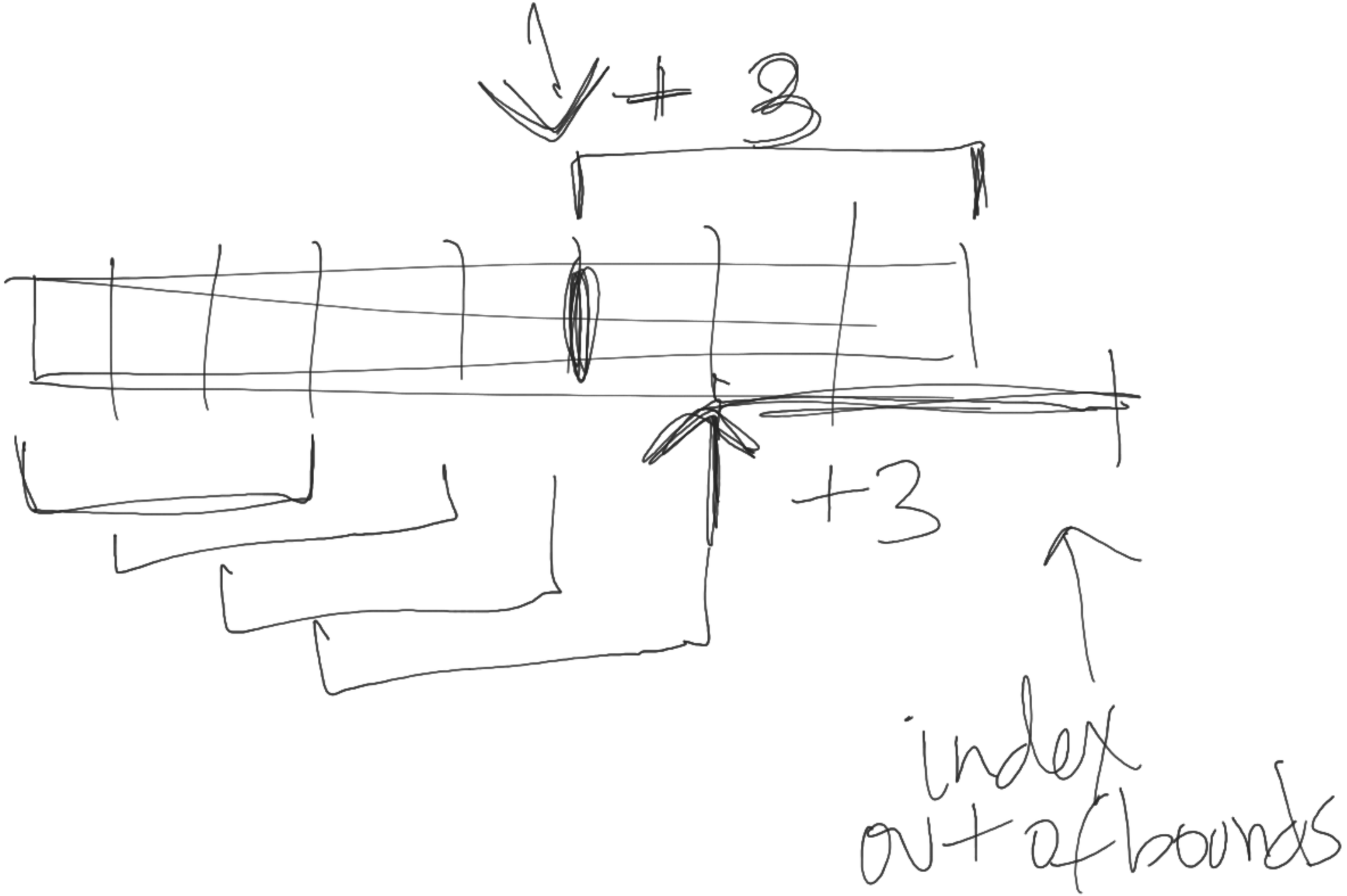
for i: len(array):

for ⇒ check each subarray with size i:

for: check the sum of subarray.

if sum == K:

increment counter


+ 3
+3
index
out of bounds

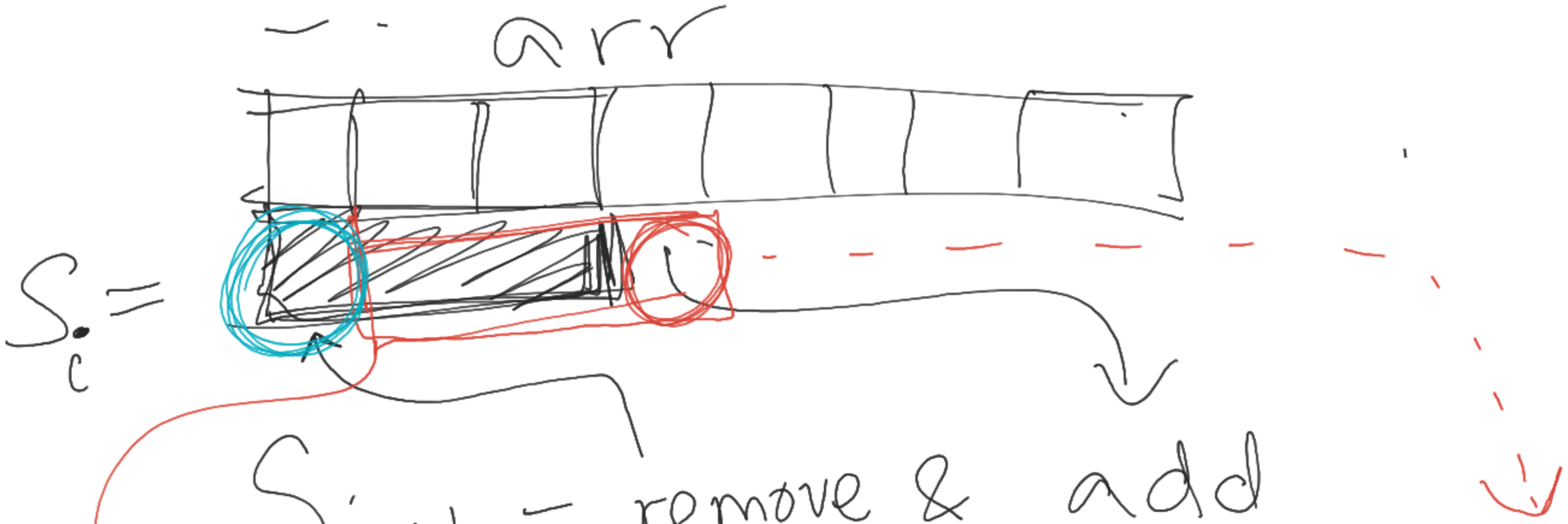
The least efficient: don't implement this way

$O(n^3)$

```
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        counter = 0
        for subSize in range(1, len(nums)+1):
            # the reason we have len(nums)+1 is to handle the case when subSize == len(nums)
            #print("subarray size: ", subSize, end = ': ')
            for i in range(0, len(nums)-subSize+1):
                S = sum(nums[i:i+subSize])
                #print(nums[i:i+subSize], end = '----')
                if k == S:
                    counter += 1
            #print()


        return counter
```

Using a window

arr



S_{i+1} = remove & add

so, $S_{i+1} = S_i - arr[i] + arr[i+k]$

subsize.

Still bad performance: $O(n^2)$

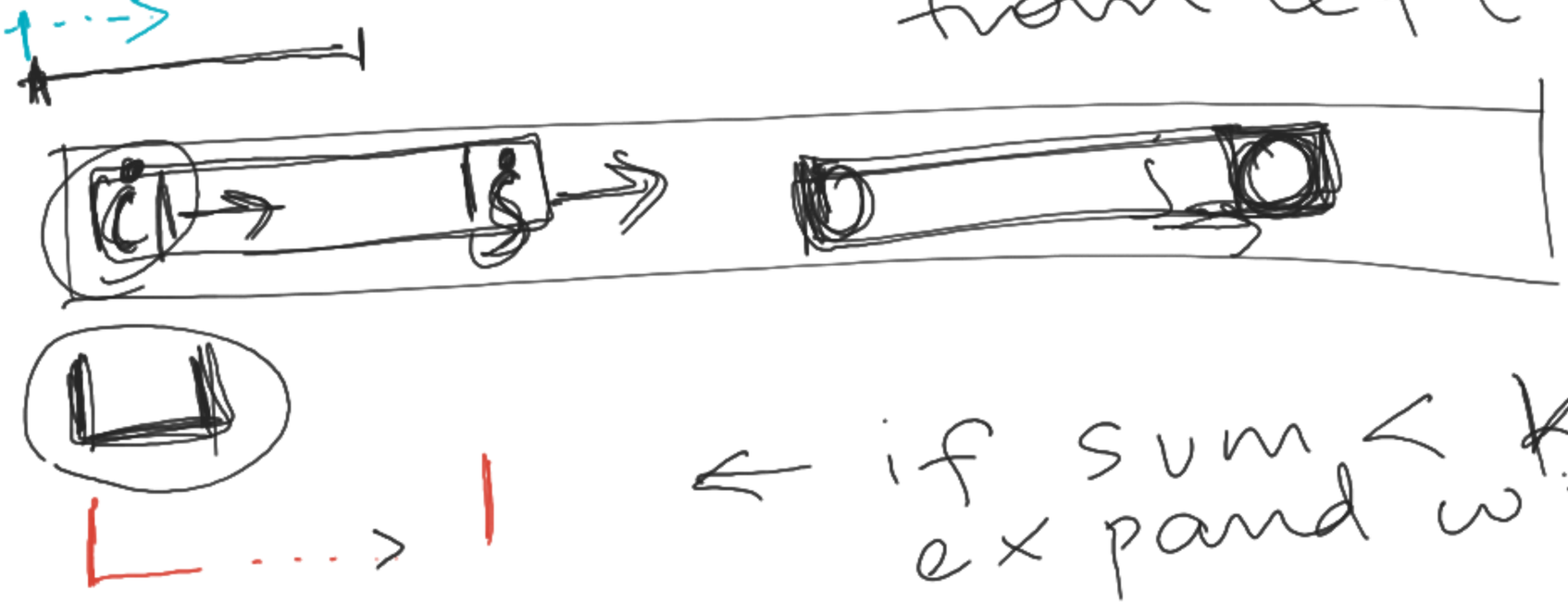
```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        counter = 0
        for subSize in range(1, len(nums)+1):
            # the reason we have len(nums)+1 is to handle the case when subSize == len(nums)
            #print("subarray size: ", subSize, end = ': ')

            # S is the window
            S = sum(nums[0:subSize])
            if S == k:
                counter += 1
            for i in range(0, len(nums)-subSize):
                S = S - nums[i] + nums[i+subSize]
                #print(nums[i:i+subSize], end = '----')
                if k == S:
                    counter += 1
            #print()


        return counter
```

we only improved this part using window technique

K ← desired sum

O(n)

if sum > K : shrink window from left



← if sum < K expand window

if sum = K : counter ++

```
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        counter = 0

        i,j = 0,0
        S = sum(nums[i:j]) # adding one because substring method is exclusive

        while i < len(nums) and j < len(nums):
            if S == k:
                counter += 1
                i += 1
                j += 1
            elif S < k:
                j += 1
                S += nums[j]
            elif S > k:
                i -= 1
                S -= nums[i]

        return counter
```