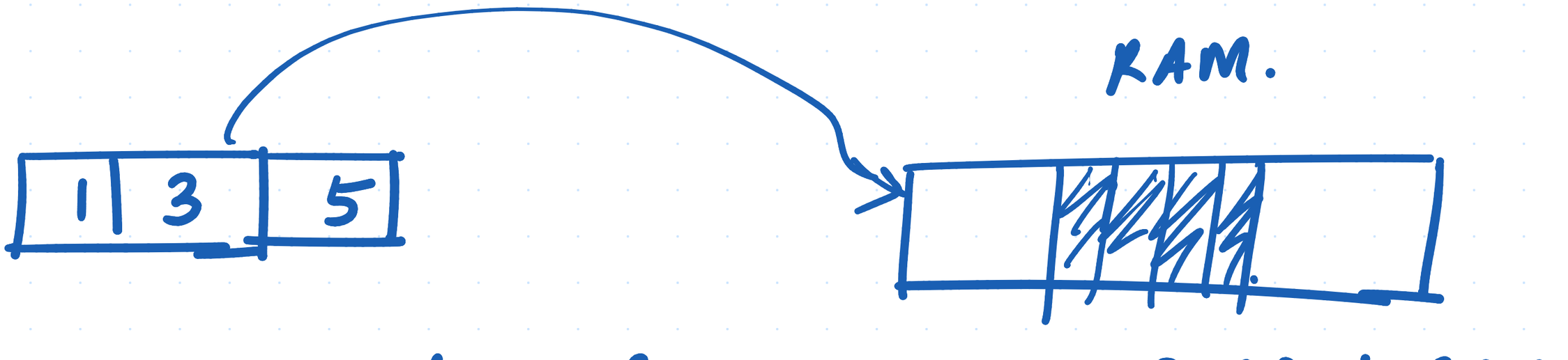
16/08/25.

Arrays / Data Structure.



Measured in Bytes , 8 GB of RAM.

Byte → 8 bits.

10^9 → Bytes

Bit → a portion that can store 0/1.

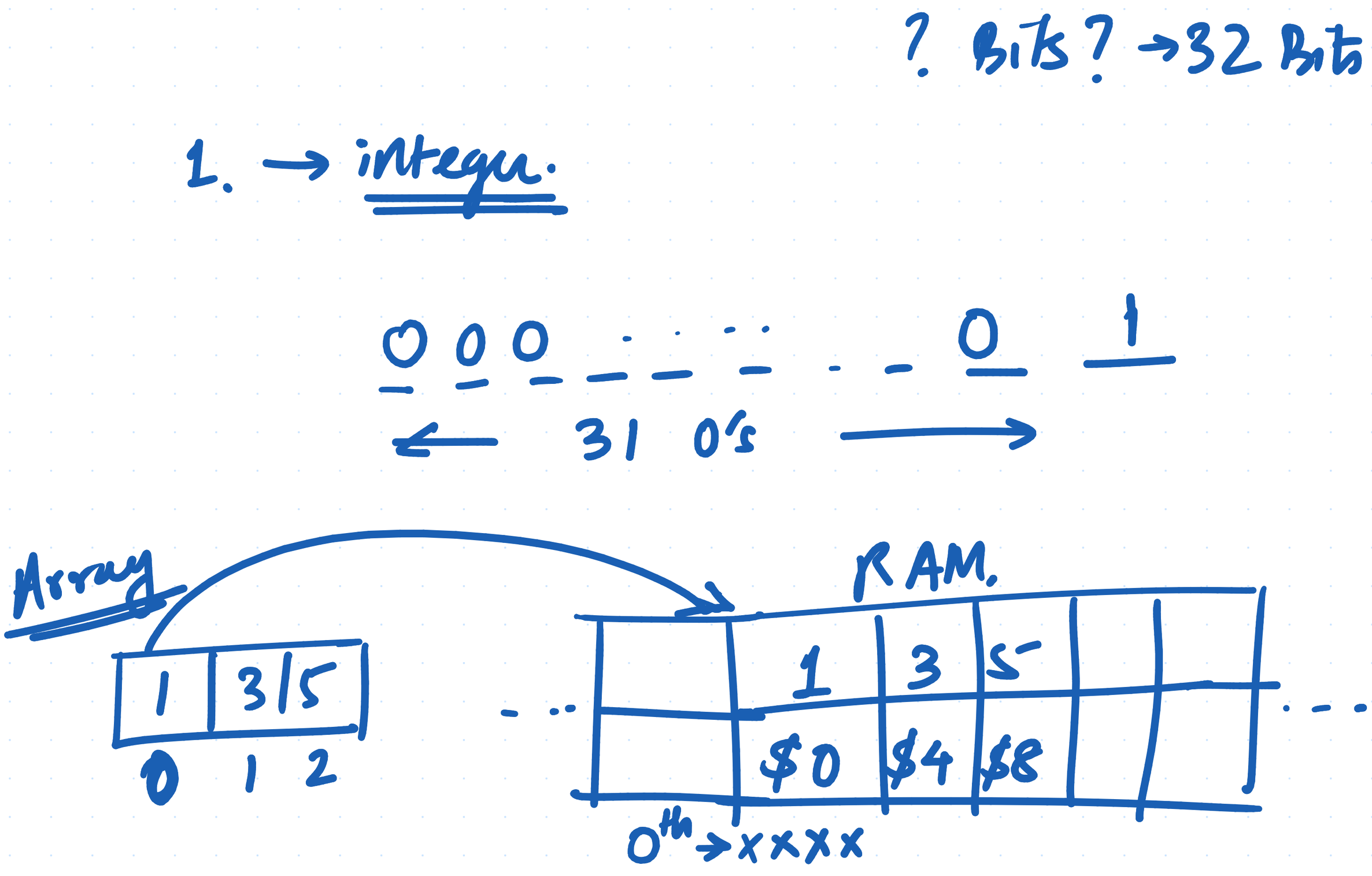
Integer → 4 byte.

? Bits ? → 32 Bits.

1. → integer.

0 0 0 . . . . . 0 1

← 31 0's →



(xxxx +4) → 3 value.

1 byte → 1 character (xxxx +8) → 5 value.

ASCII value representation.

access.

Read → O(1). constant time read

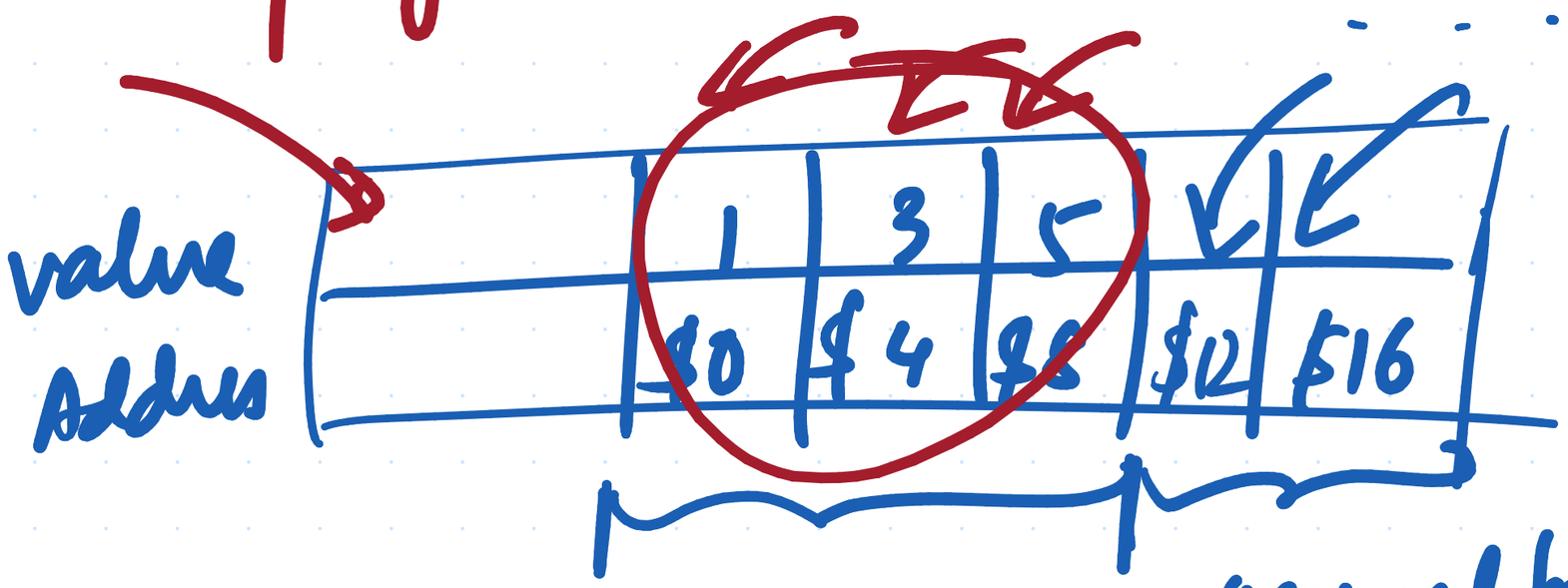
Write → O(1) constant time write.

Random Access Memory → we can access in constant time, any part of the memory given we have the correct location / address for it.

C, C++ (Garbage collector).

java.

program.

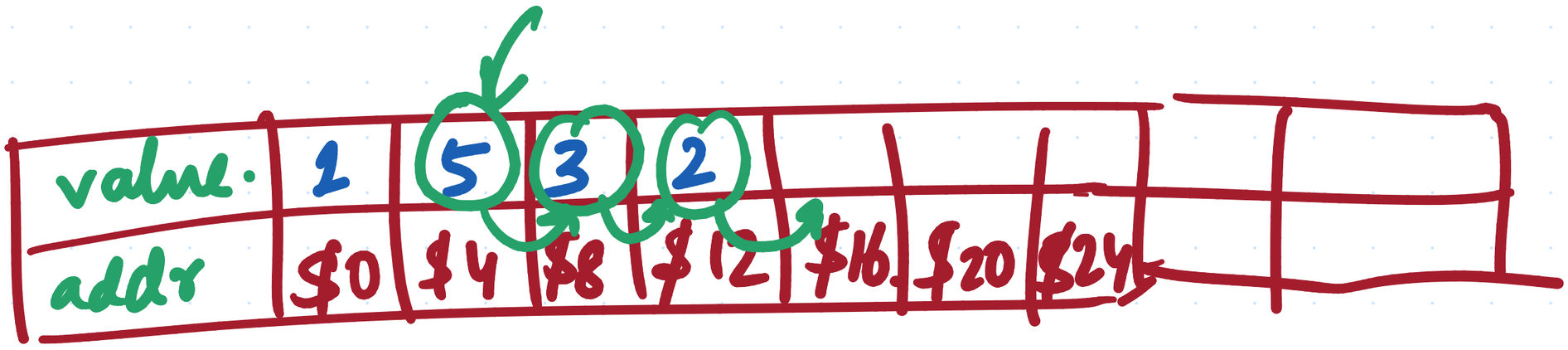


accessed by other / used by others.

C, C++ → vectors → dynamic.

int arr[10];

Python → dynamic array → List.



← 8 →

inserting in the middle of the array, we need to shift element, so it will take O(n) time.

Deletion of first & last → O(1)

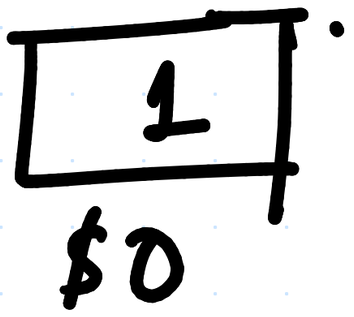
O(n) → remove & shift elements of the array

Dynamic Array, how is it maintained in Python or any kind of Prog Lang.

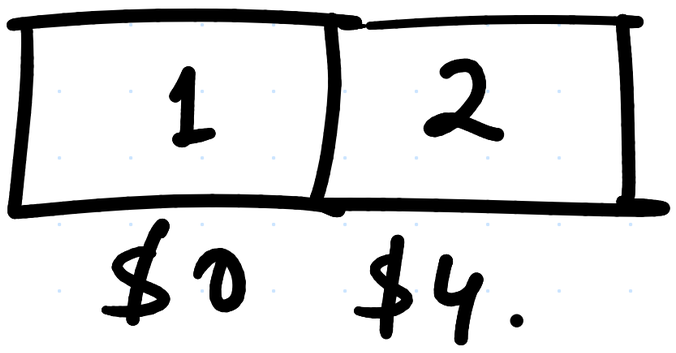
→ No size is mentioned.

a = [ ] ; list.

#1



#2

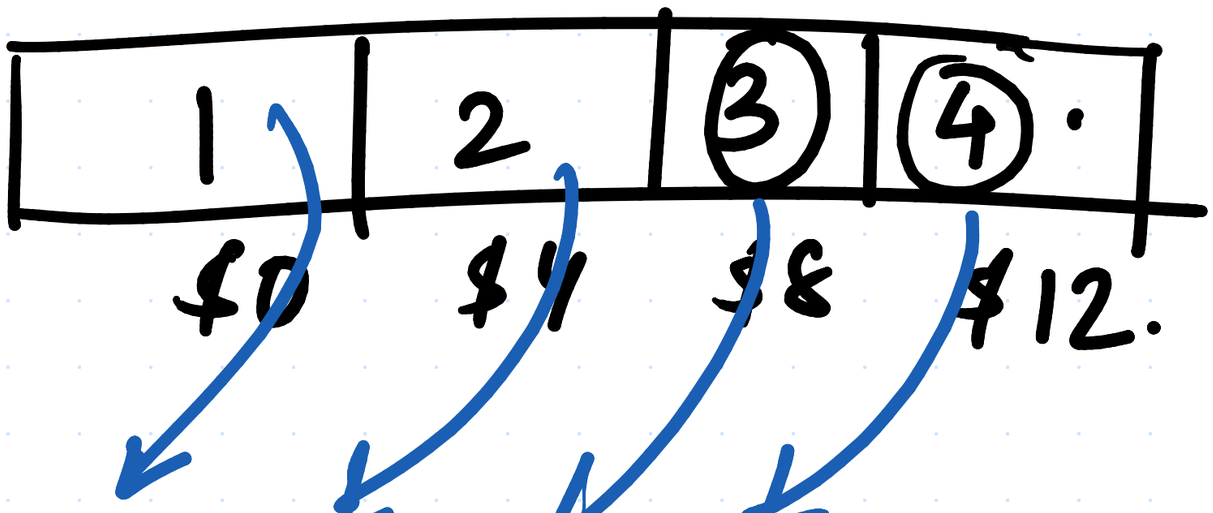


2 > 1

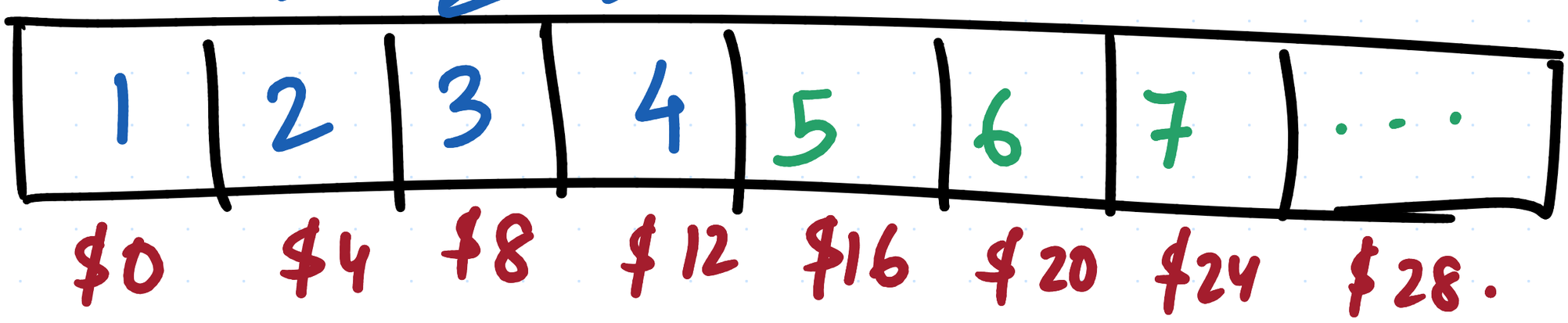
4 > 2+1

8 > 4+2+1

#4



#8



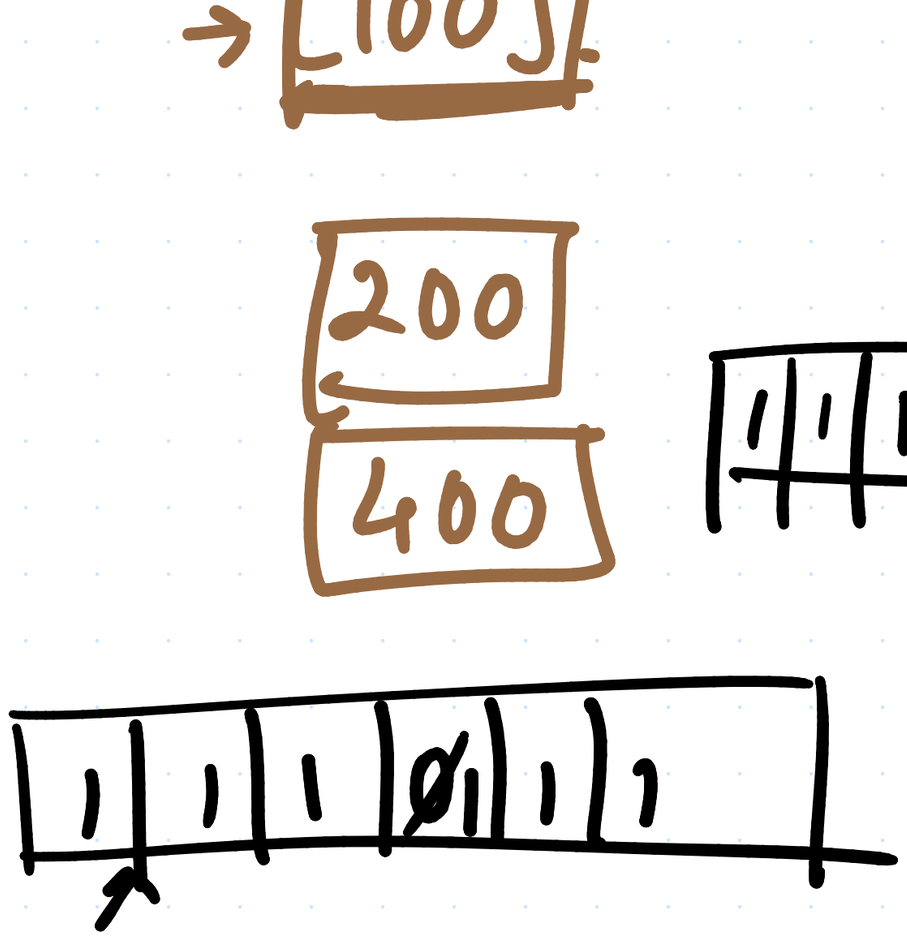
Tradeoff b/w time & space.

→ [100]

Optimal → 2x.

200

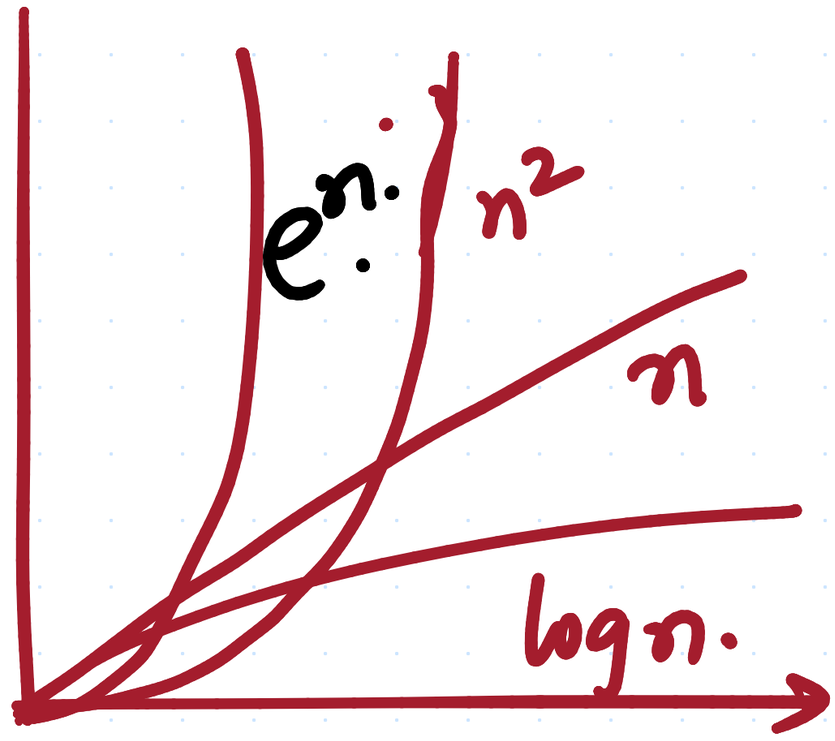
400



Mostly O(n) & O(log n)

↓

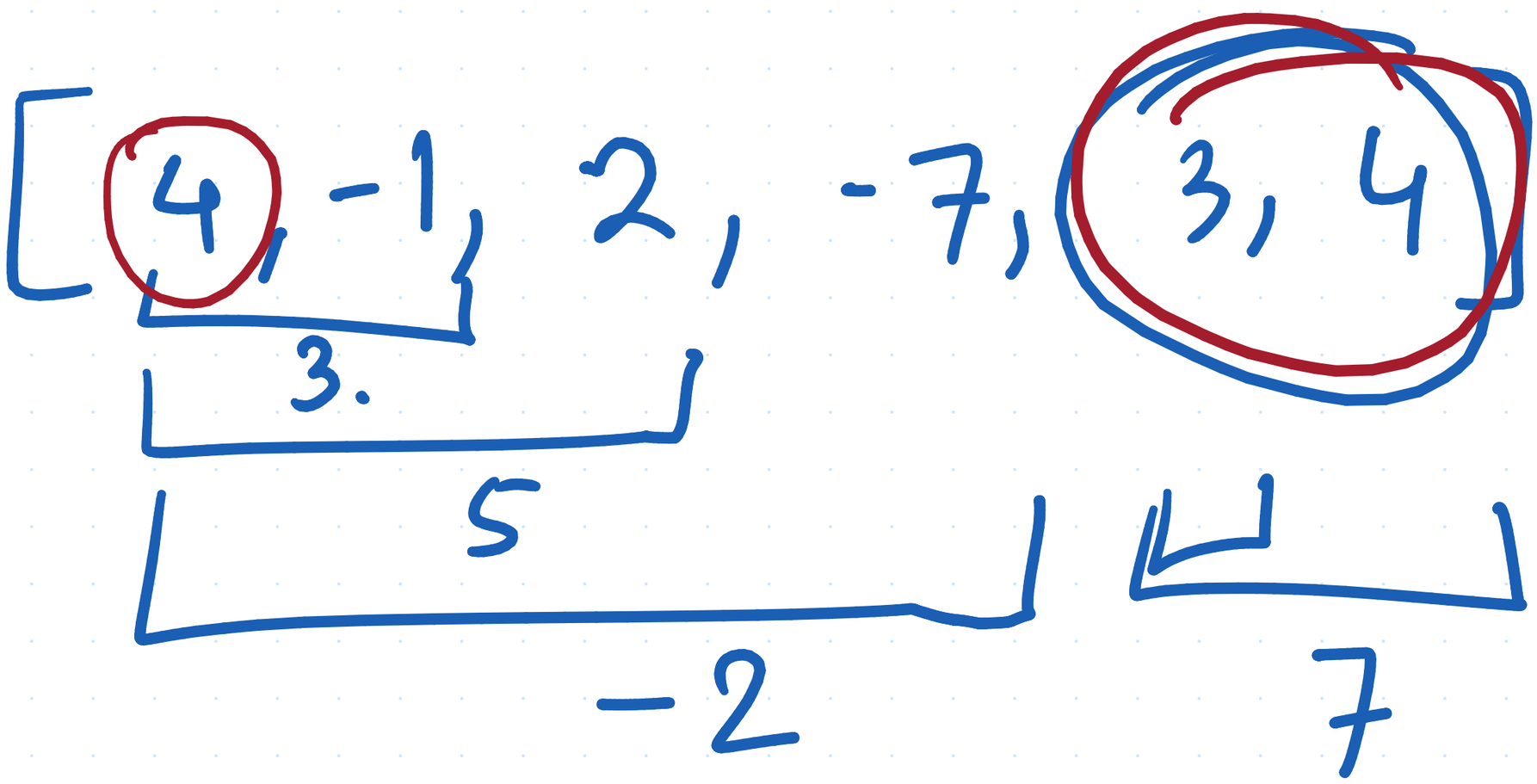
type of Algo.



Brute force → $O(n^2)$.

Kadane's Algorithm (2 ptr method)

Find a non-empty subarray with the largest sum.



o/p → 7.

```
def bruteForce (nums)
    maxSum = nums[0]
    for i in range (len(nums)):
        curSum = 0
        for j in range (i, len(nums)):
            curSum += nums[j]
            maxSum = max(maxSum, curSum).
    return maxSum.
```

nums =

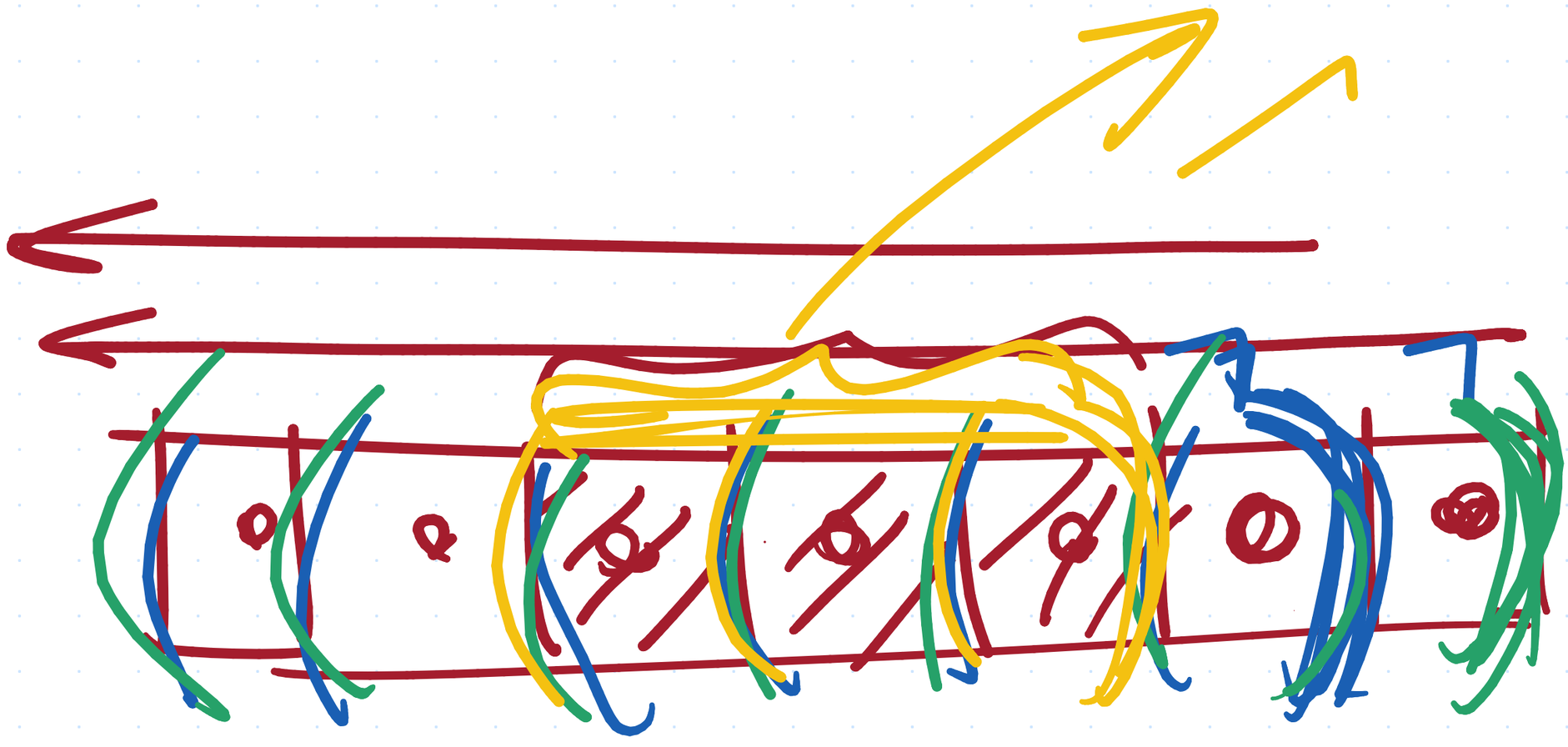
4	-1	2	-7	3	4

i = 0 1 2 3 4 5

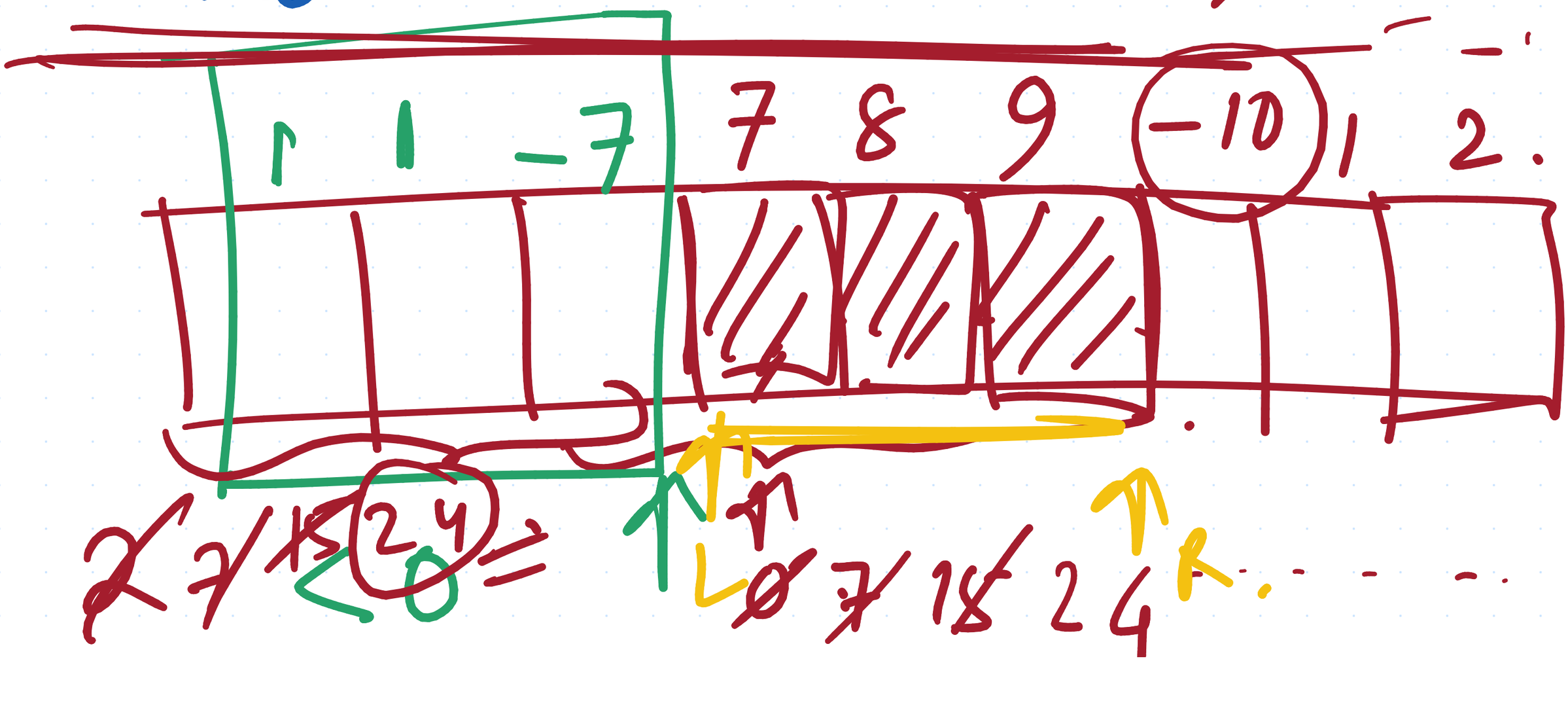
maxSum = 4.

curSum = 0 4 3 -4 5.

maxSum = 4, 4, 4, 5, 5

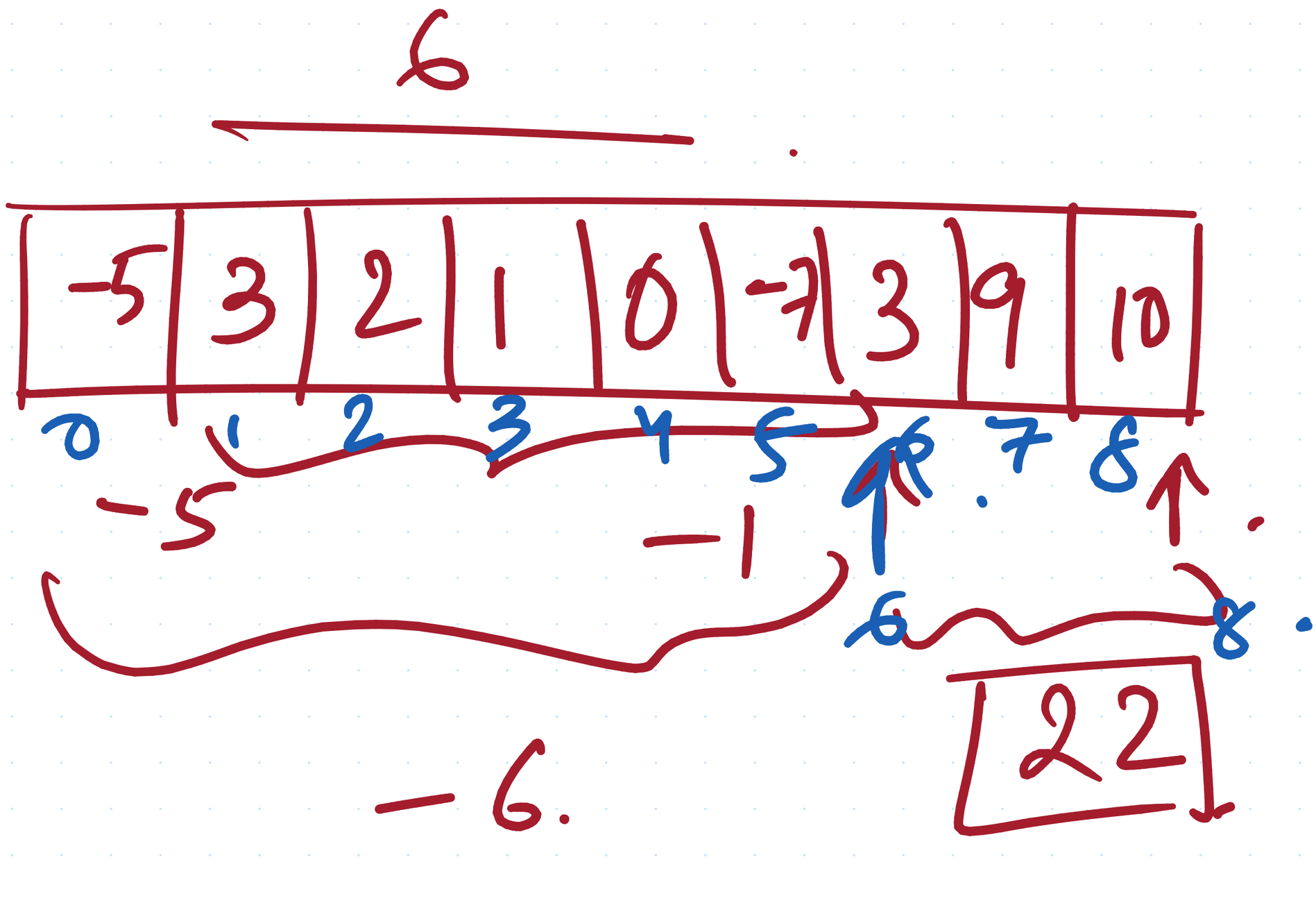


```
def Kadanes (nums):
    maxsum = nums[0]
    cursum = 0
    for n in nums:                    O(n)
        cursum = max(cursum, 0)
        cursum += n
        maxsum = max(maxsum, cursum)
    return maxsum.
```



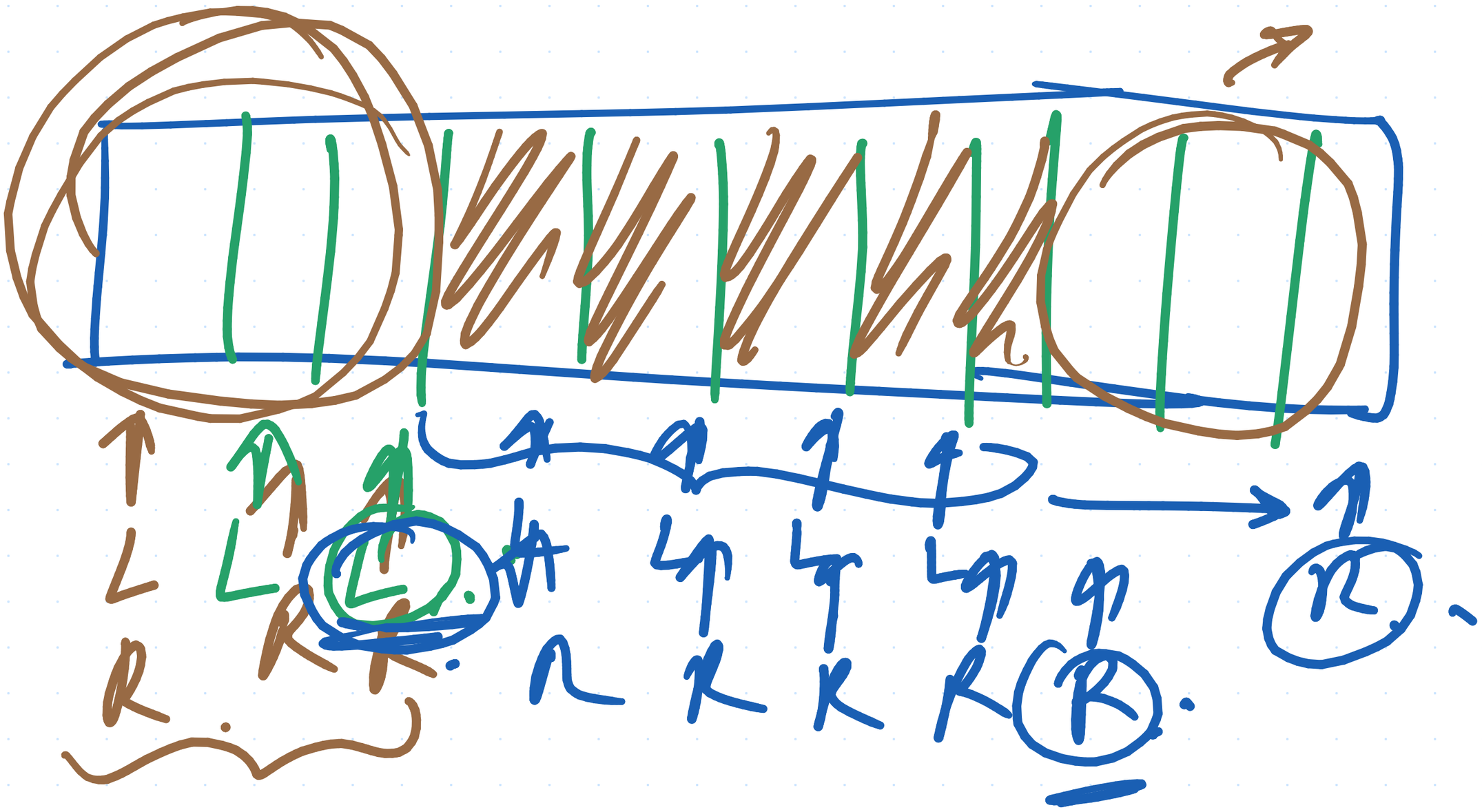
Sliding window Version of Kadane's Algorithm.

Q: Return the left & right index of the max-subarray sum, assuming there is exactly one result.



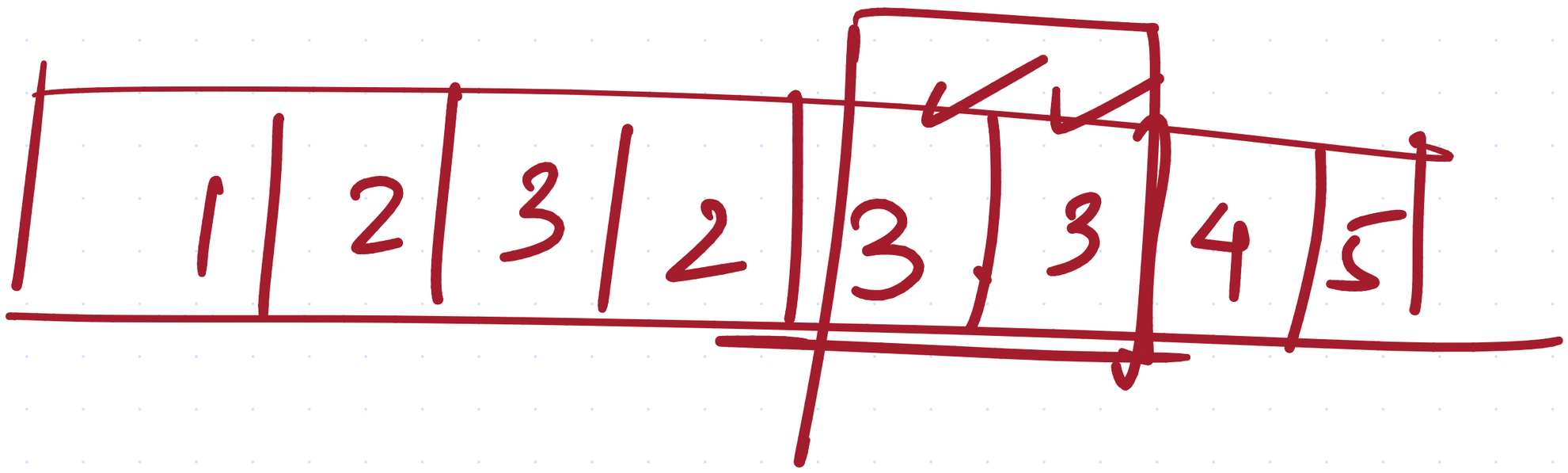
-6.

```
def slidingWindow(nums):
    maxSum = nums[0]
    curSum = 0
    maxL, maxR = 0, 0
    L = 0
    for R in range(len(nums)):
        if curSum < 0:
            curSum = 0; L = R
        curSum += nums[R]
        if curSum > maxSum:
            maxSum = curSum
            maxL, maxR = L, R
    return [maxL, maxR]
```

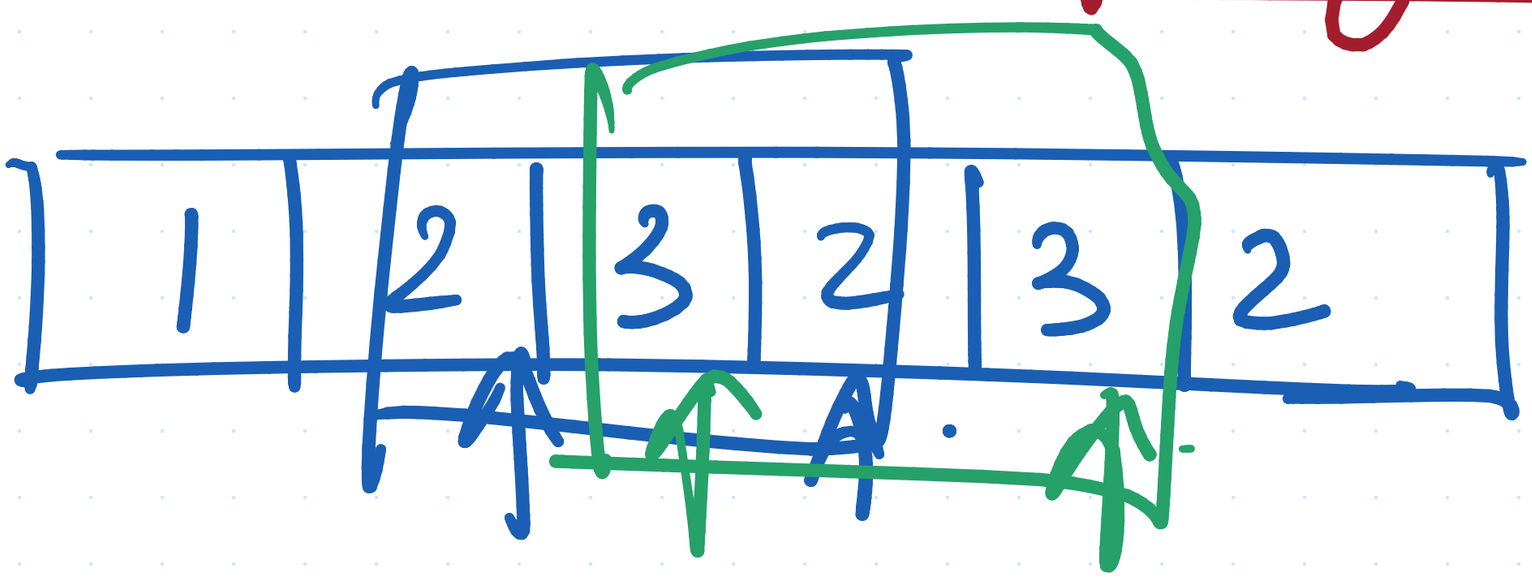


Sliding Window

Given an array, return true if there are two elements within a window of size k. that are equal.



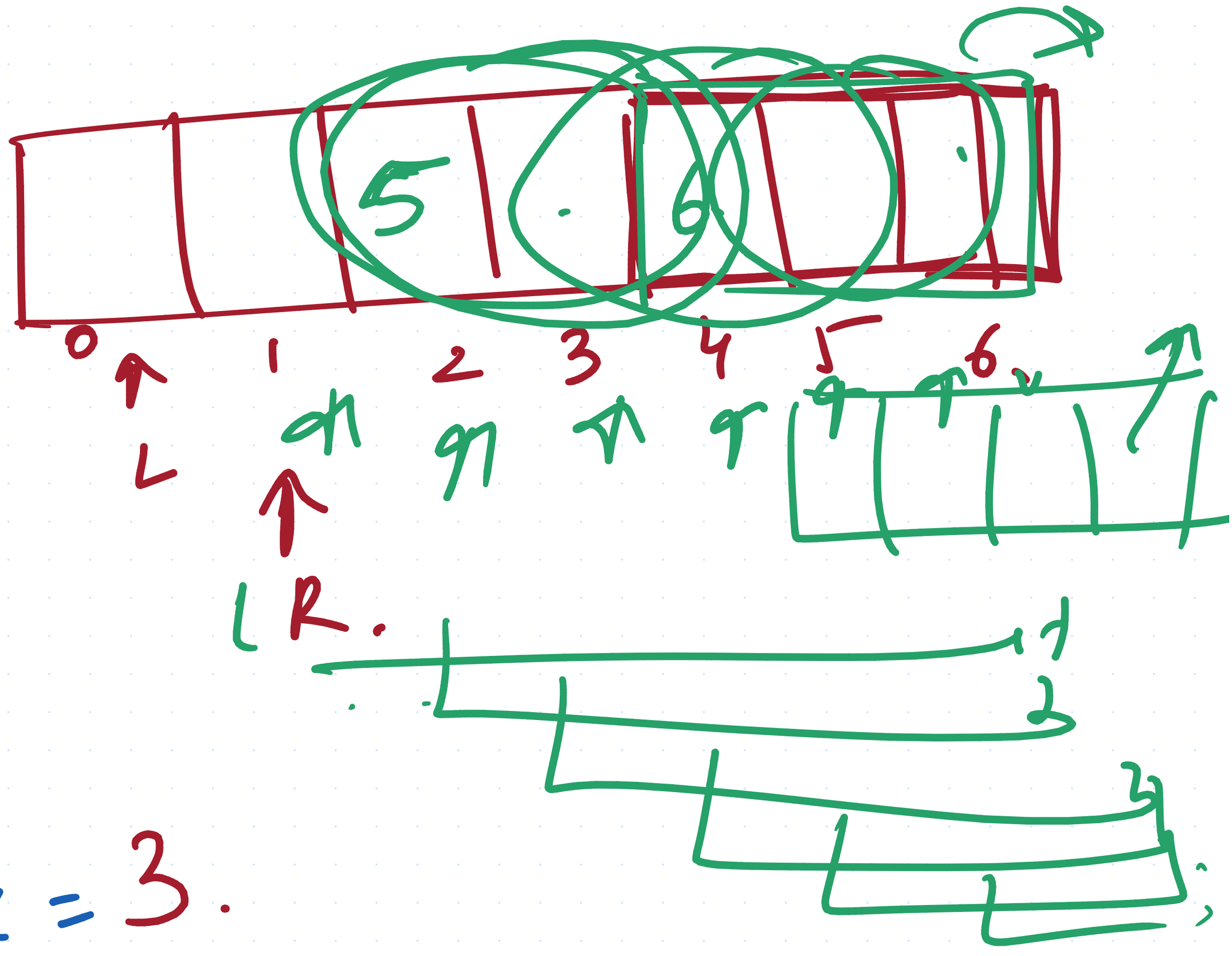
window of size. $k = 2$.

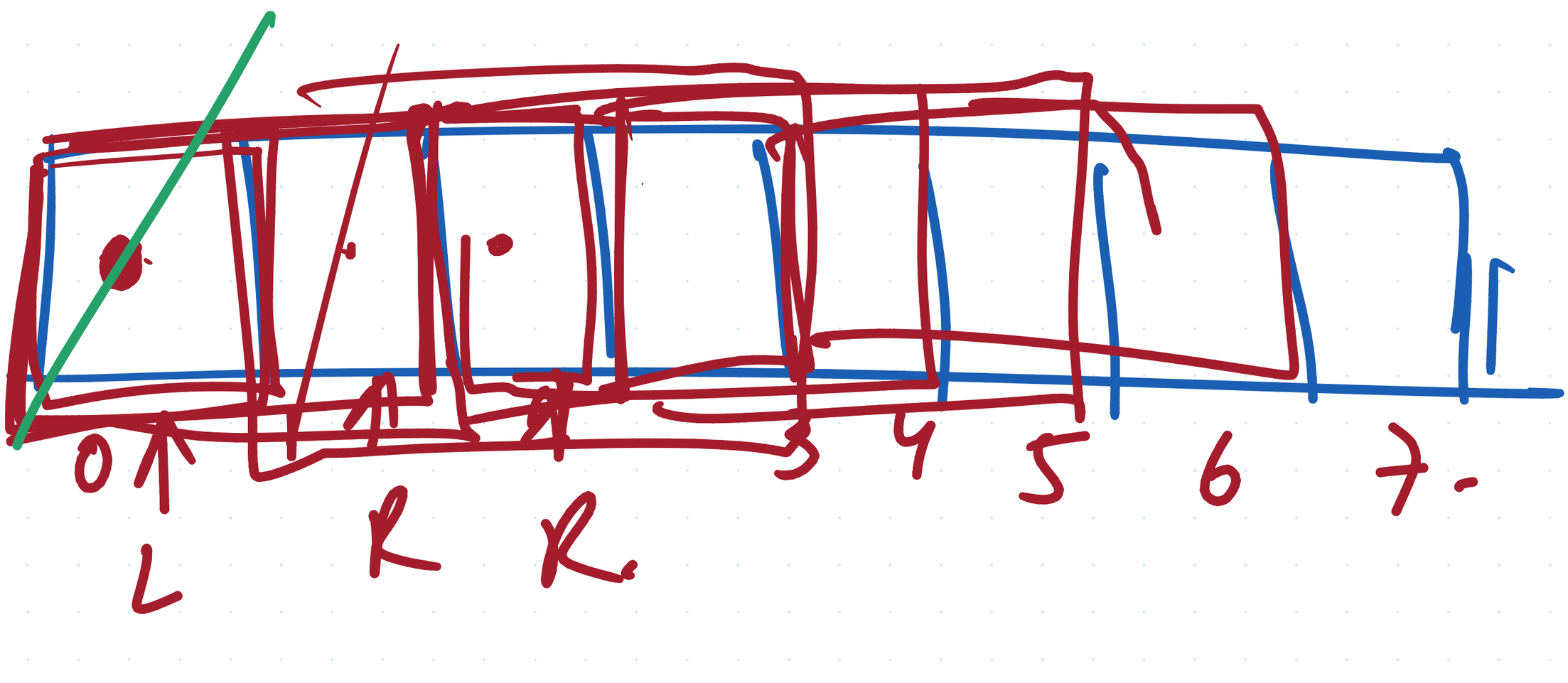


$k = 3$.

2 time.

```
def closeDupBF(nums, k):
    for L in range(len(nums)):
        for R in range(L+1,
                       min(len(nums), L+k)):
            if nums[L] == nums[R]:
                return True.
    return False.
```



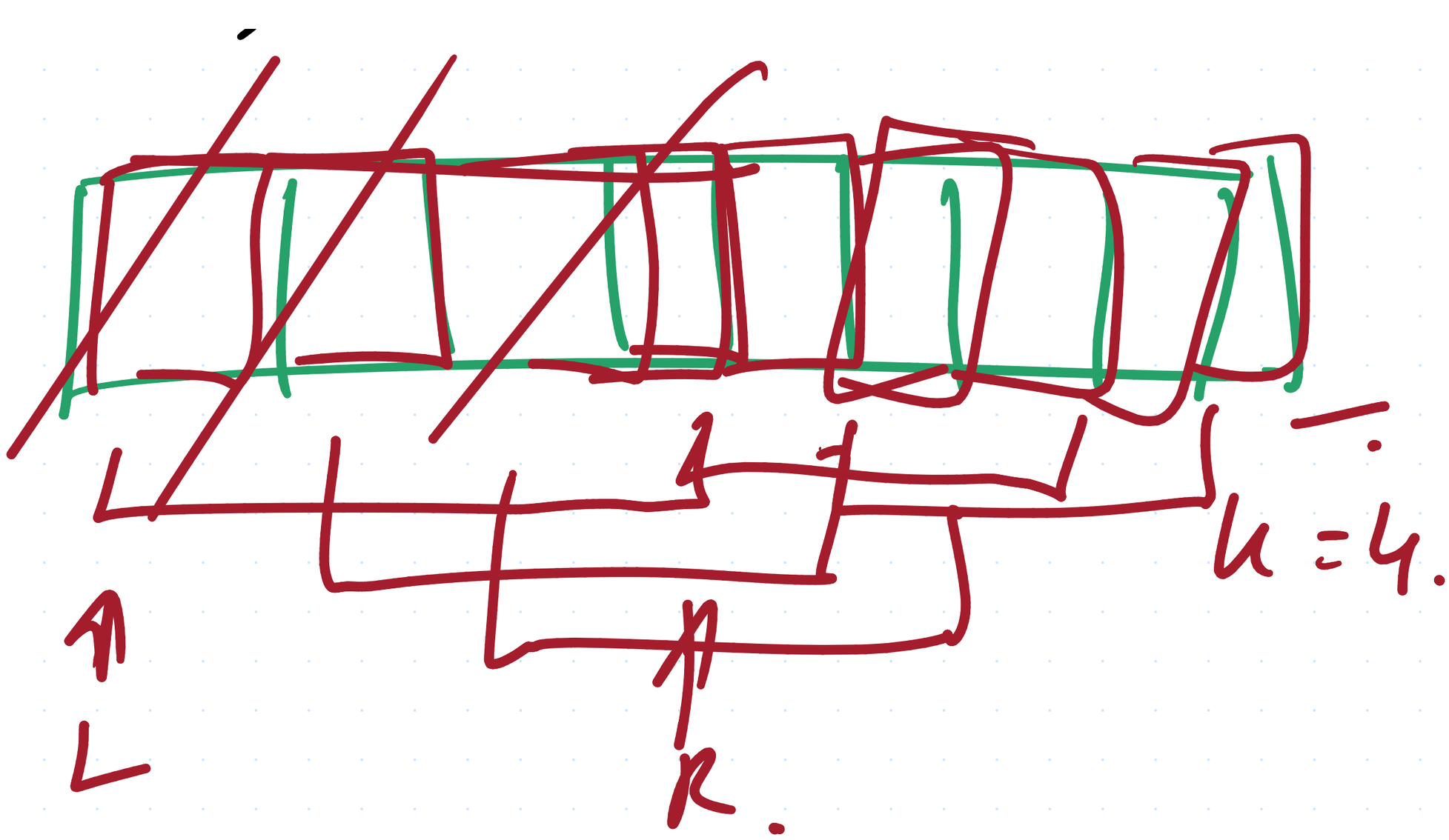


Sliding window.

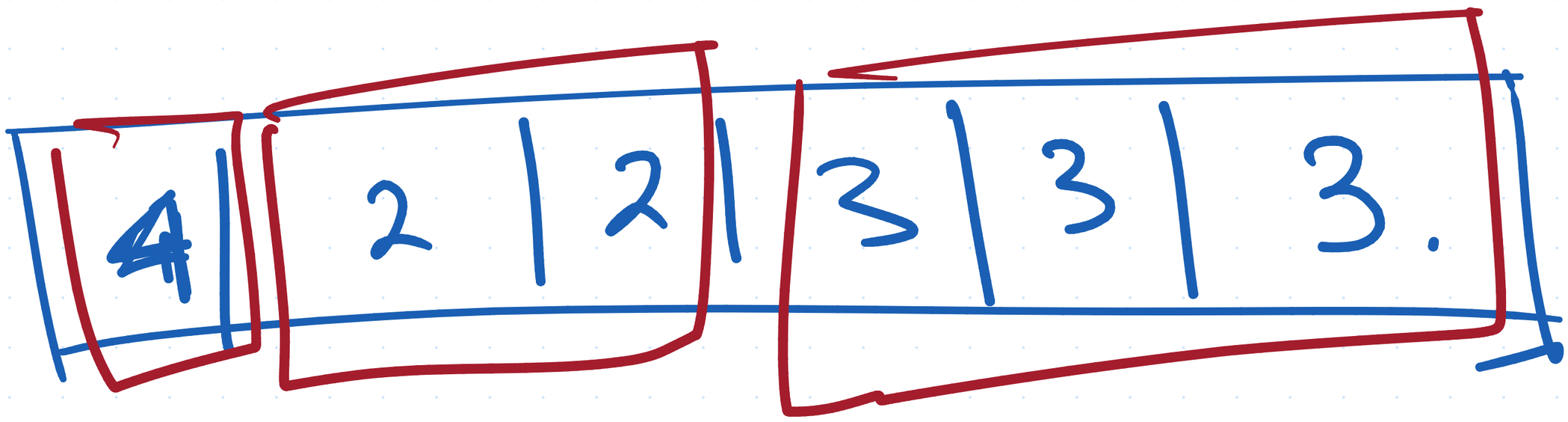
O(1) → hash set.

O(n) complexity:

```
def closeDuplicates(nums, k):
    window = set()
    L = 0
    for R in range(len(nums)):
        if R - L + 1 > k:
            window.remove(nums[i])
            L += 1
        if nums[k] in window:
            return True
        window.add(nums[R])
    return False.
```



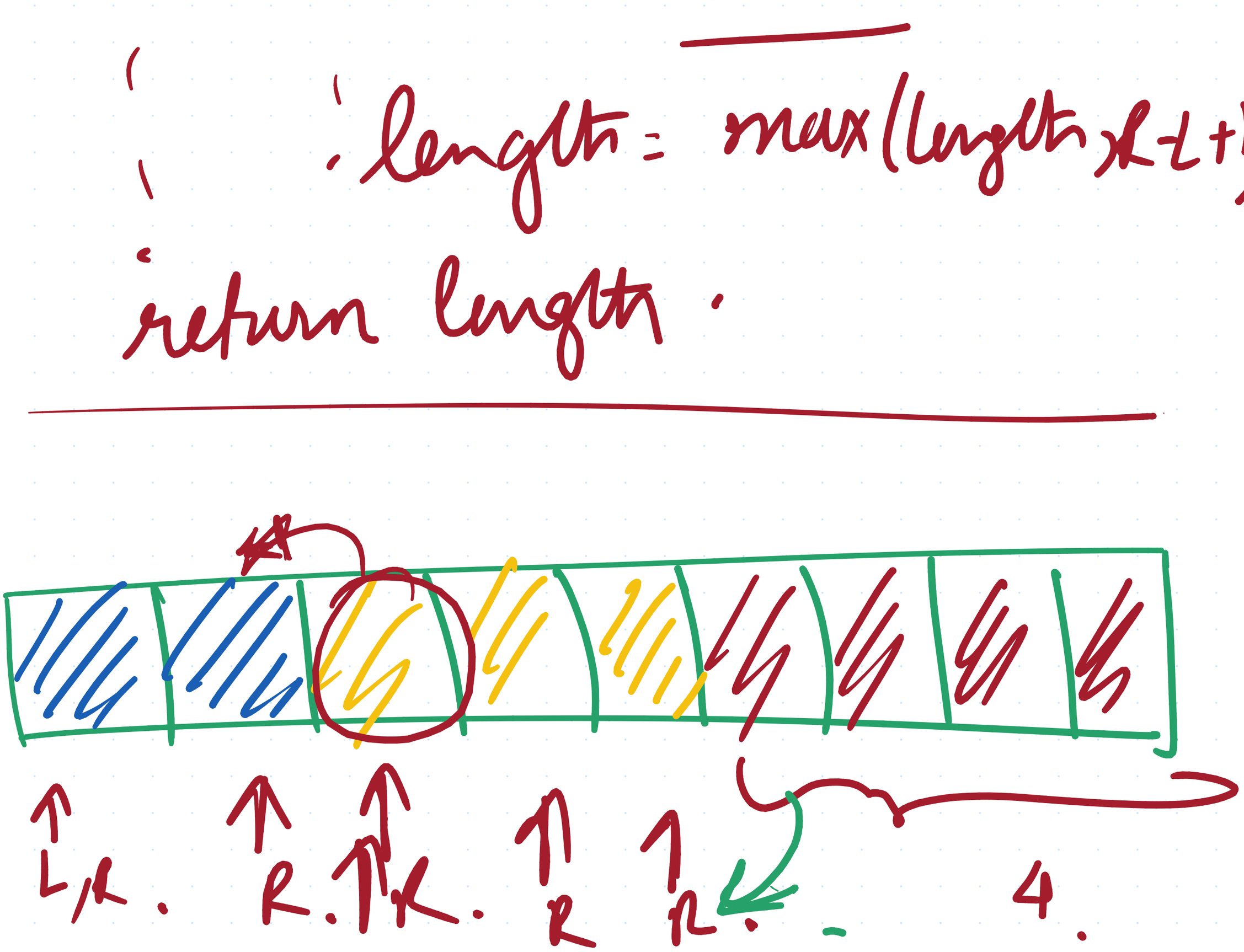
Sliding window with variable length.



- Find the length of the longest subarray with the same value in each position.

O(n)

```
def longestSubarray(nums):
    length = 0; L = 0
    for R in range(len(nums)):
        if nums[L] != nums[R]:
            L = R
        length = max(length, R-L+1)
    return length
```



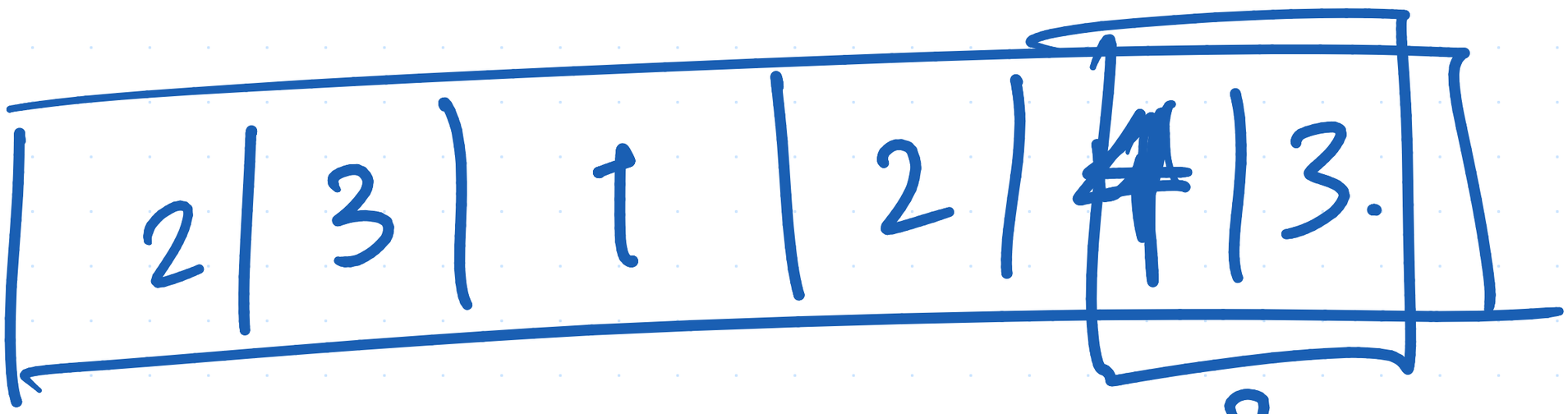
L

L, R - - - - - - R.

= (R - L + 1).

find the min length subarray where the sum is greater than or equal to the target.

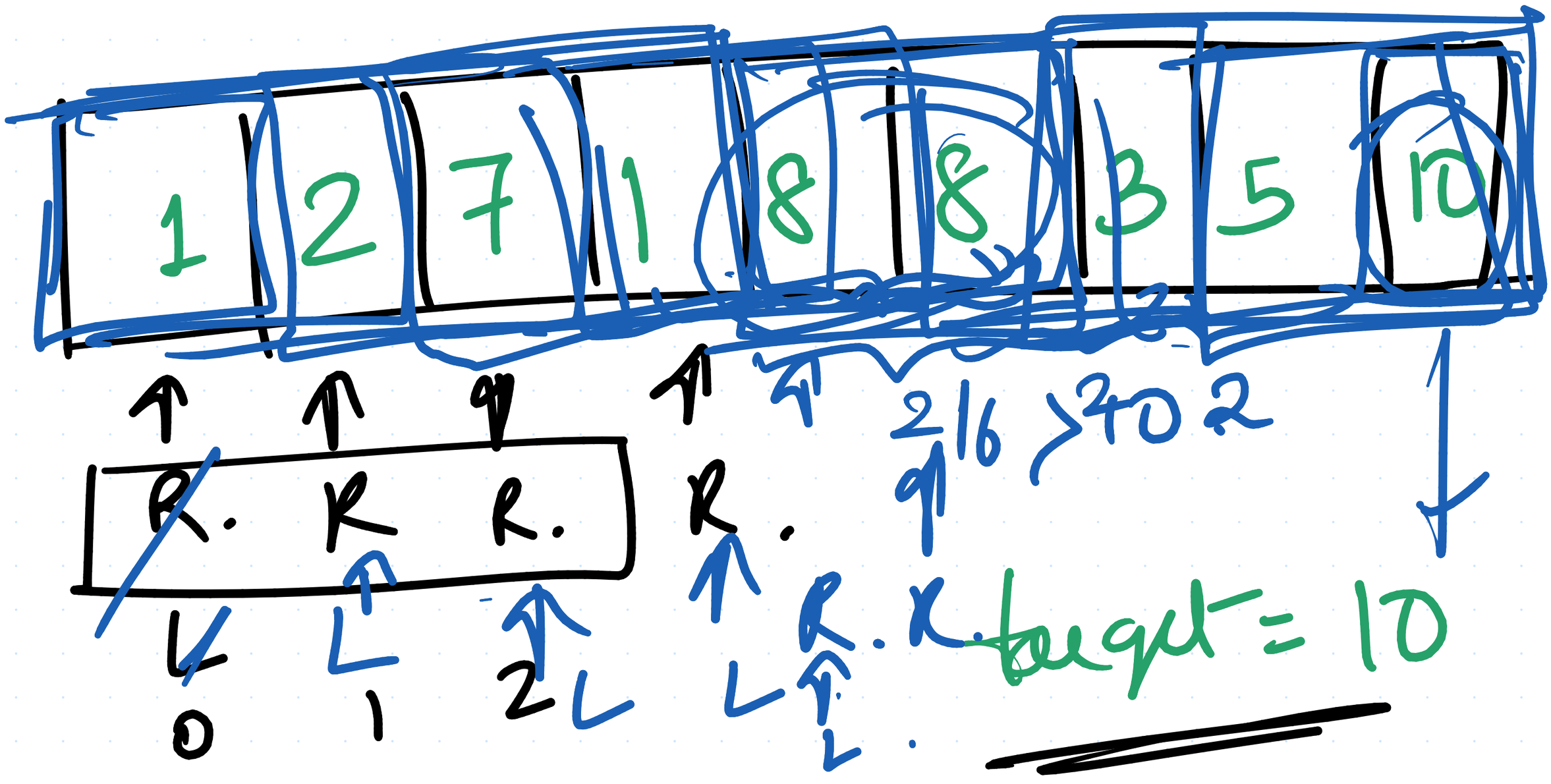
Assume all values are +ve.



Target = 6.

```
def shortestSubarray(nums, target):
    L, total = 0, 0
    length = float("inf") # ∞
    for R in range(len(nums)):
        total += nums[R]
        while total >= target:
            length = min(R-L+1,
                         length)
            total -= nums[L]
            L += 1
    return 0 if length == float("inf")
```

else length.

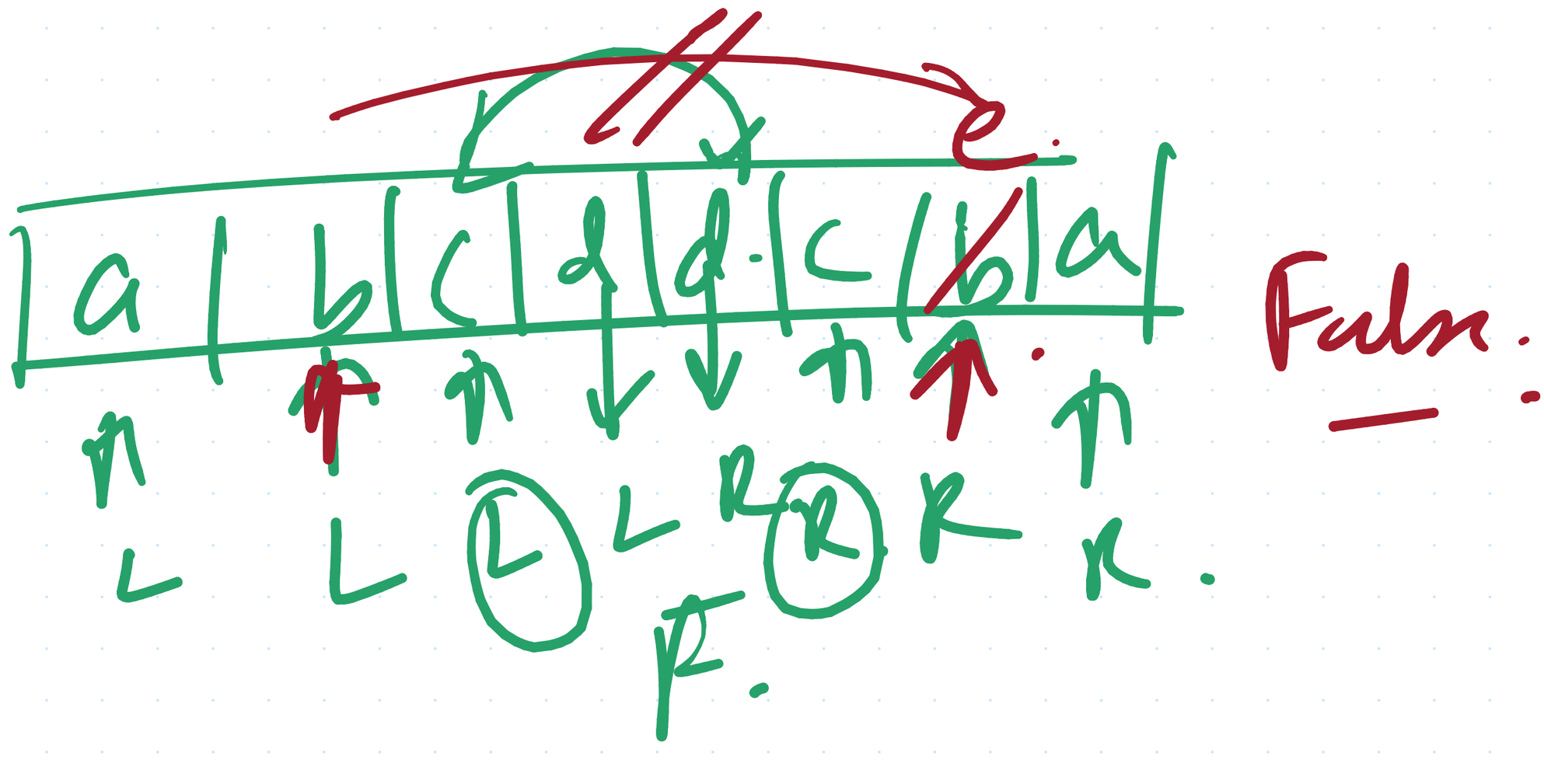


total = X 3 10 · 11;

R - L + 1

O(n)

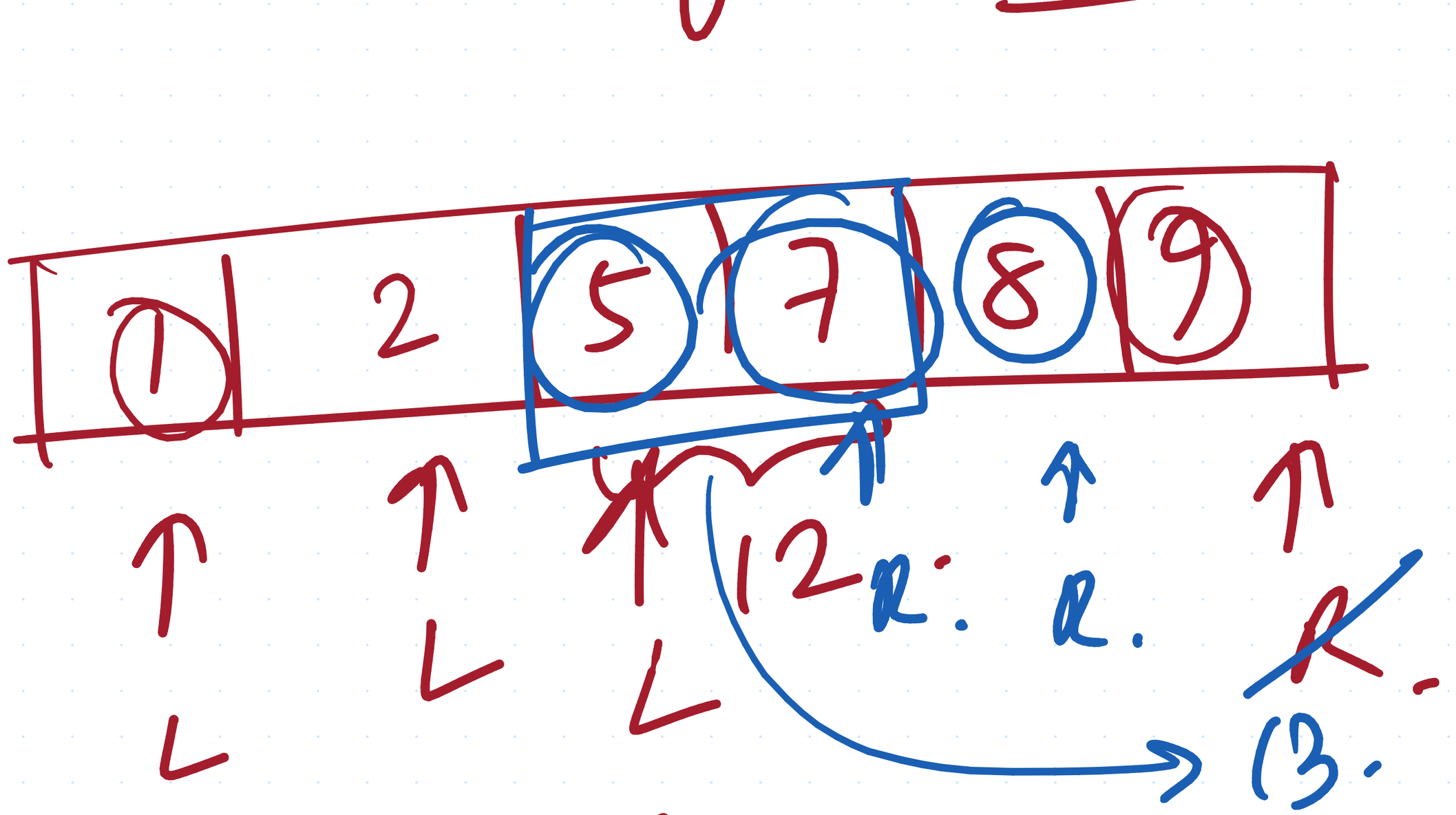
O(1) → 1.



check if an array is palindrome.

```
def isPal(word):
    L, R = 0, len(word) - 1
    while L < R:
        if word[L] != word[R]:
            return False
        L += 1; R -= 1
    return True
```

Q/ Given a sorted i/p array, return the two indices of the two elements which sum up to the target value. Assume there's exactly 1 soln.



```
def targetSum(nums, target):
    L, R = 0, len(nums)-1
    while L < R:
        if nums[L] + nums[R] > target:
            R -= 1
        elif nums[L] + nums[R] < target:
            L += 1
        else
            return [L, R]
```