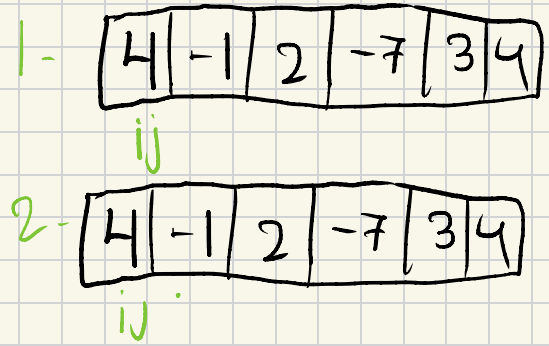
Brute Force For Finding non-empty subarray with largest sum.

```python
import time
a = list(map(int, input().split()))

def bruteforce(a): #(On^2)
    maxSum = a[0]

    for i in range(len(a)):
        curSum = 0
        for j in range(i , len(a)):
            curSum += a[j]
            maxSum = max(maxSum, curSum)
    return maxSum

st = time.process_time()
bruteforce(a)

et = time.process_time()
print(f"Running Time: {et-st:.6f}")
print(bruteforce(a))
```

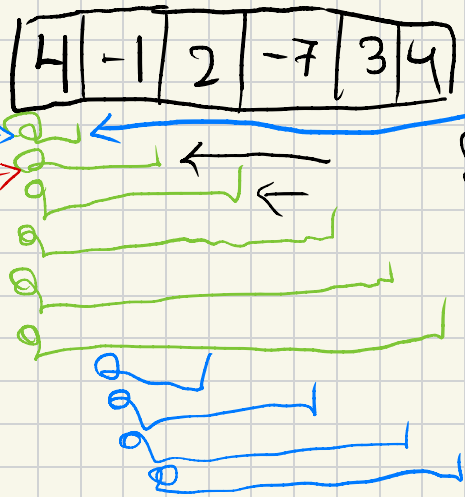
be non-empty

can set to $-\infty$

Go through every possible sub array and keep track of what the largest sum caculated



to get every sub array starting at the first position is going to be O(n)



and you have starting point every single element so it's O(n) for every element

$O(n \cdot n)$

$O(n^2)$

have a lot of redundant and inefficiency

this contribute so necessary bcuz its positive and first digit.

negative doesn't contribute to finding the maximum number
- would be nice to ignore the value but can't bcuz its continuous.

KADANE

- Why would we ever add a negative value if we don't have to and doesn't contribute to largest sum?
- We shouldn't add "negative **previous** sum" to our **current** value.
- Make sure **current_sum** value isn't negative. If become negative, reset back to 0.

not even keeping track of pointers
there is still window

Linear Time Algorithm
One Loop (no nested)
O(n)

```python
def kadanes(nums):
    maxSum = nums[0]
    curSum = 0
    for n in nums:
        curSum = max(curSum, 0)
        curSum += n
        maxSum = max(maxSum, curSum)
    return maxSum
```

Sliding Window O(n)

- The L and R pointers can be at the same position (that just mean we have sub array of length one)
- Left Pointer should never cross the right pointer

```python
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 = L
            maxR = R
    return maxSum, maxL, maxR
```