Kadane's Algorithm.

[4, -1, 2, -7, 3, 4]

↳ Find a non-empty Sub-array with the largest sum.

Sub-Array → By meaning is contiguous.

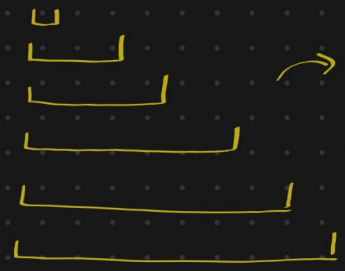
→ Query What happen if the Array is all negative like [-4, -1, -2, -7, -3, -4] ↳ in this Case.

→ Generally we can't return 0 as it given we need find non-empty subarray so we return -1, it's negative but it's the largest.

→ So Answer the original Question

1) Method - 1) Brute force

[4, -1, 2, -7, 3, 4]



→ keep the track of sum of each Sub Array

↳ $O(n^2)$

```
vector<int> findMaxSumSubArray(vector<int> arr){
    int sum = 0;
    for(int i = 0; i < arr.size(); i++){
        int temp = 0;
        for(int j = i; j < arr.size(); j++){
            temp += arr[j];
            if (temp > sum){
                sum = temp;
            }
        }
    }
```

```
        }
        cout << sum << "  " ;
    }
```

→ if you want to return a sub string →

```
int findMaxSumSumArray(vector<int> arr){
    int sum = INT_MIN;
    int start = 0;
    int end = 0;
    for( int i = 0; i < arr.size(); i++){
        int temp = 0;
        for (int j = i ; i < arr . size(); j++){
            temp += arr[j];
            if( temp > sum){
                sum = temp;
                start = i;
                end = j;
            }
        }
    }
}
vector<int> maxSubArray (arr.begin()+start ,arr.begin()+end+1){
    return maxSubArray;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int bruteforce (vector<int> arr){
    int maxSum = arr[0];
    for(int i=0; i<arr.size(); i++){
        int curSum = 0;
        for (int j=i ; j < arr.size(); j++){
            curSum += arr[j];
            maxSum = max(maxsum, curSum);
        }
    }
    return maxSum;
}
```

Kadane's Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;

int KadaneAlgo( vector<int> arr ){
    int maxSum = arr[0];
    int curSum = 0;
    for(auto n : arr){
        curSum = (maxSum(curSum, 0)
        CurSum += n,
        maxSum= max(maxSum,curSum);
    }
    return maxSum;
}
```

```cpp
curSum= (curSum >0?curSum:0)+n;
```

```
// Return the left and Right index of the max SubArray.
// assuming there's exacty one result (no ties).
// Sliding window variation of Kadane's : O(N).
vector<int> slidingWindowKadane( vector<int> arr){
    int maxSum= arr[0];
    int curSum = 0;
    int maxL =0 , maxR = 0;
    int L= 0;
    for (int R=0;  R< arr.size(); R++){
        if ( curSum < 0){
            curSum= 0;
            L = R;
        }
        CurSum += arr[R];
        if ( curSum > maxSum){
            maxSum = CurSum;
            MaxL = L;
            MaxR = R;
        }
    }
    return vector<int>{MaxL , MaxR};
}
```

918. Maximum Sum Circular Subarray

Medium | Topics | Companies | Hint

Given a **circular integer array** `nums` of length `n`, return *the maximum possible sum of a non-empty* ***subarray*** *of* `nums`.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A **subarray** may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i], nums[i + 1], ..., nums[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % n == k2 % n`.

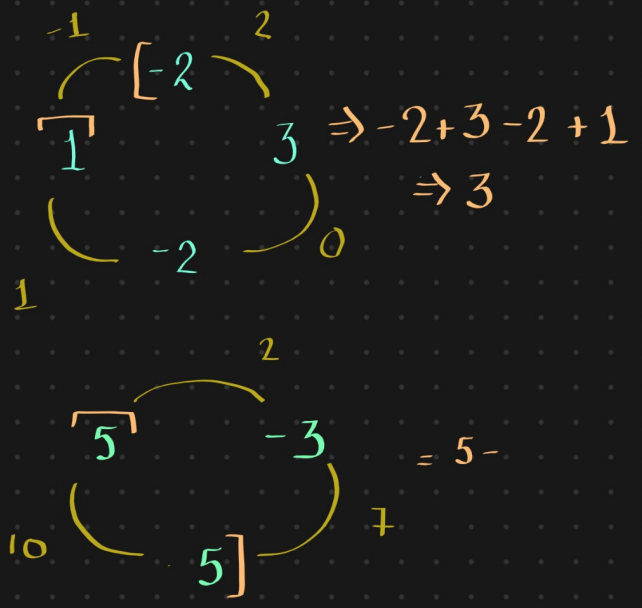
Example 1:

```
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.
```

Example 2:

```
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
```

Example 3:



practice (8th April; 2025)

```
# include < bits/stdc++.h>
using namespace std;

int KadaneAlgorithms (vector<int> nums){
    int maxSum = nums[0];
    int curSum = 0;

    for(int n : nums){
        curSum = ((curSum > 0)? curSum : 0) + n;
        maxSum = max (maxSum, curSum);
    }

    return maxSum;
}
```

```
vector<int> KadaneAlgoReturnPointer(vector<int> nums){
    int maxSum = nums[0];
    int curSum = 0;
    int maxL = 0;
    int maxR = 0;
    int L = 0;
    for(int R=0; R < nums.size(), R++){
        if(curSum < 0){
            curSum = 0;
            L = R;
        }
        curSum = num[R];
        if(curSum > maxSum){
            maxSum = curSum;
            maxL = L;
            maxR = R;
        }
        return vector<int>{maxL, maxR};
    }
```

918. Maximum Sum Circular Subarray

Medium | Topics | Companies | Hint

Given a **circular integer array** `nums` of length `n`, return *the maximum possible sum of a non-empty* ***subarray*** *of* `nums`.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A **subarray** may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i], nums[i + 1], ..., nums[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % n == k2 % n`.

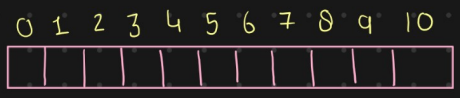
Example 1:

```
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.
```

Example 2:

```
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
```

Example 3:



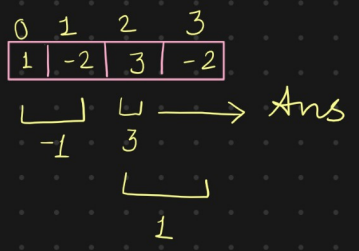
given

nums[i] in a circular array

is nums[i+1] % n and the previous element of nums[i] is nums[i-1+n] % n.

Example - 1)

curSum = (curSum > 0 ? curSum : 0) + n.



Example - 2)

maxSum = 5

curSum = 2.

maxSum = 7

curSum = 7.

↳ if n == size - 1.

curSum = nums[size-1]