Arrays: Sequence of homogeneous data.

In arrays, elements are stored at contiguous memory locations.

	0	1	2	3	4	5	6
arr[] =	2	1	3	5	7	2	3
	x	x+4	x+8	x+12	-	-	-

10

O(1) lookup time for any index in the array.

a + (i)*4

arr[0] → x + 0*4 = x

arr[1] → x + 1*4 = x+4

arr[2] → x + 2*4 = x+8

Sorting-based approach:

[0, -10, 1, 3, -20] → [-20, -10, 0, 1, 3]

result = ~~1~~ 2

O(nlogn + n) = O(nlogn)

Smallest ans: 1

n Largest ans: n+1

[0, -10, 1, 3, -20]

Use the ith index of the array as a bucket to store the information that whether (i+1) occurs in the array or not.

n+1

[-3, -1, -2, 5, -7] n=5 [1,6]

1✓ 2✓ 3✓

return 3

[-3, -1, -2, 5, -7]

1✓ 2✓ 3✓

[0, -10, 1, 3, 20] n+1 [1,n]

Replace all elements which are outside of the range [1, n] with (n+1).

[1,n]

[-6, 6, -1, 3, 6] ✓

return 2

0

neg 1, 1

[x0 0 x0
1 x0 1
1 x0 1]

	0	1	2
0	x0	0	0x
1	1	0x	1
2	x0	0	x0

rows: [T | F | T]

cols: [F | T | F]

TC: O(n*m)

Aux. Space: O(n+m)

row-~~0~~ 1

col-~~0~~ ~~2~~ 0 1

	0	1	2
0	x0	0	x0
1	1	x0	1
2	x0	0	0x

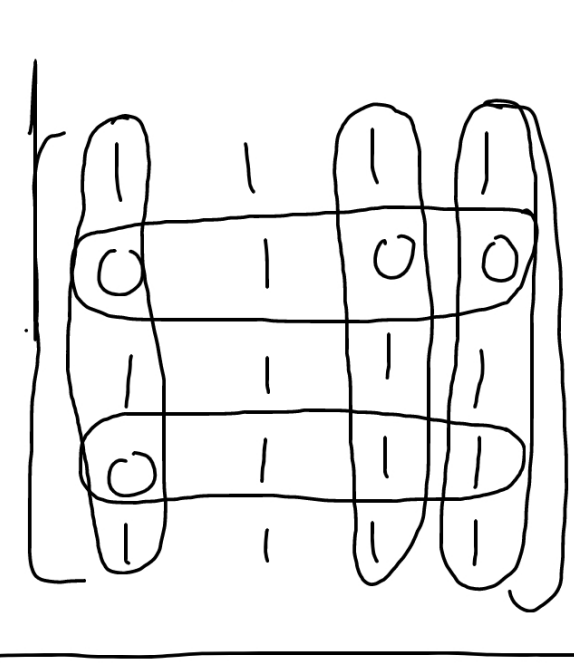
first_row = True

first_col = False ✓

x0 1 x0 x0
0 x0 0 0
x0 1 x0 x0
0 x0 x0 x0
x0 1 x0 x0

first_row = False

first_col = True



⇒

0 1 0 0
0 0 0 0
0 1 0 0
0 0 0 0
0 1 0 0

TC: O(n*m)

Aux Space: O(1) ✓

[2, -3, 1, 4, 7, -1] = 12

[4, -3, 1, 4, 7, -1] = 13

[1, 2, 3, -2, 5] = 9

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

n

no. of subarrays of length = 1 : n

" " " = 2 : n-1

" " " = 3 : n-2

= n : 1

Total no. of subarrays = 1+2+3+ ... +n = $\left[\frac{n(n+1)}{2}\right]$

Let's find the maximum sum of any subarray ending at index i.

	0	1	2	3	4	5
	2	-3	1	4	7	-1
max_till_here:	2	-1	1	5	12	11
start:	0	1	2	2	2	2

max_till_here[0] = arr[0]

max_till_here[i] = max(arr[i], max_till_here[i-1] + arr[i]) ∀ i>0

(start = ~~0~~ 2)

[2, -3, 1,

mth: 2, -1,

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

n = 6

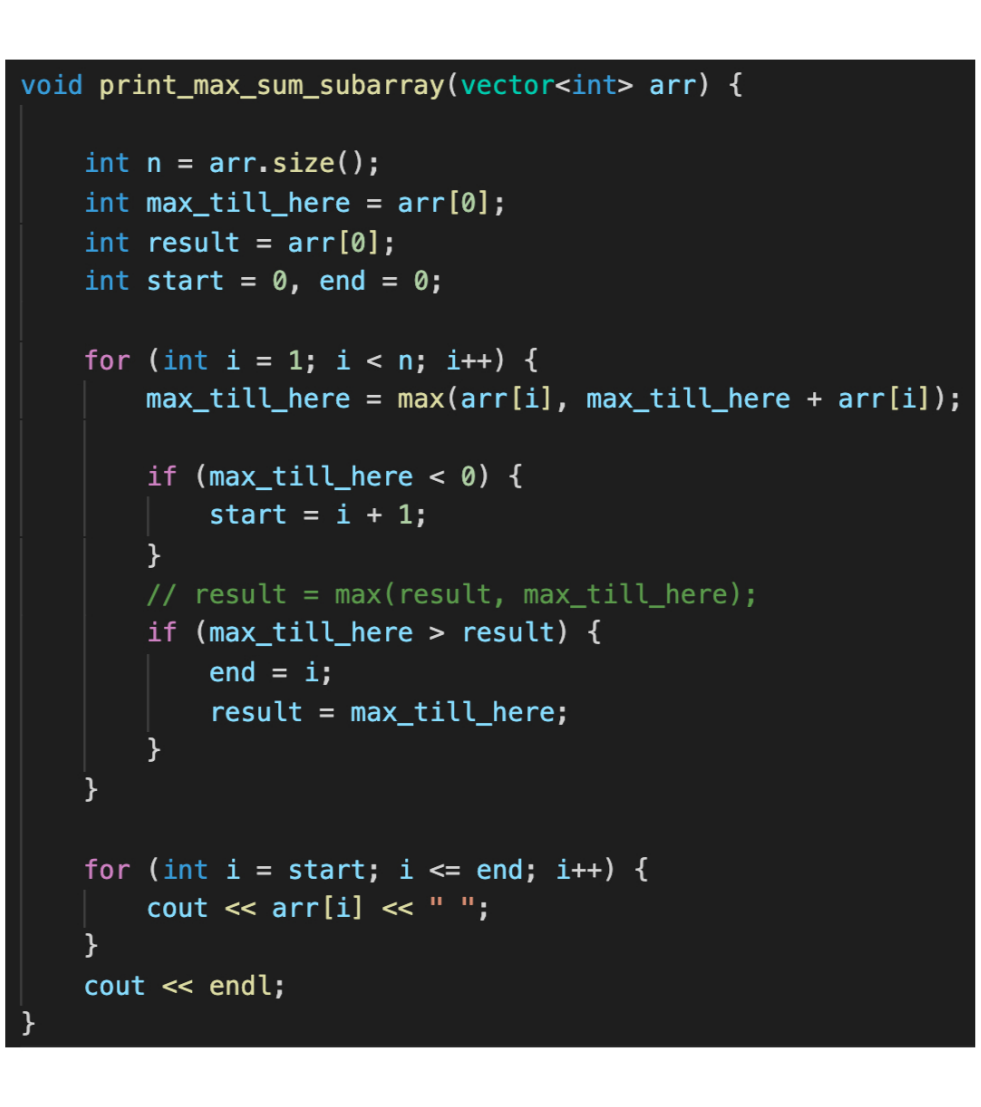
mth = ~~-2~~ -3

start = ~~0~~ 2

end = 0

result: -2

(-3) -2+-3



```
void print_max_sum_subarray(vector<int> arr) {

    int n = arr.size();
    int max_till_here = arr[0];
    int result = arr[0];
    int start = 0, end = 0;

    for (int i = 1; i < n; i++) {
        max_till_here = max(arr[i], max_till_here + arr[i]);

        if (max_till_here < 0) {
            start = i + 1;
        }
        // result = max(result, max_till_here);
        if (max_till_here > result) {
            end = i;
            result = max_till_here;
        }
    }

    for (int i = start; i <= end; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

[2, 3, -10, -20, -30]

mth: 2 5 -5 -20 -30

`{8,-8,9,-9,10,-11,12}`

[8, -8, 9, -9, 10, -11, 12, 8, -8, 9, -9, 10, -11, 12]

[1,2,3] → [1, 2, 3, 1, 2, 3] → 12

[1, -3, 1, 2] ④ ③ ✗

[2, 1, -3, 1] ③ ✗

[-1, 2, 3, -1] ⑤

[8, -8, 9, -9, 10, -11, 12]

-[2, -1, -2, 3] -3

NOTE: To find the minimum subarray sum, multiply all the elements in array with (-1) and apply Kadane's algo on upd. array. orig. array.

[2, -1, -3, 3] → [-2, 1, 3, -3]

Max. subarray sum of the upd. array = 4

∴ Min. subarray sum of the orig array = -4

Now, the total sum of the orig. array = 1

⇒ Max subarray sum considering the circular property: 1 - (-4) = 5

Maximum Circular Subarray Sum:

① Apply Kadane's algo & find the max. subarray sum. Let it be 'x'.

② Compute the sum of the overall array. Let it be 'sum'.

③ Multiply all the values in the array with (-1).

④ Apply Kadane's algo on the modified array. Let the result be 'y'.

⑤ return max (x, sum-y)

	0	1	2	3	4	5
	2	-3	1	4	7	-1
mth:	2	-1	1	5	12	11

c_start = ~~0~~ 2

c_end = ~~0~~ ~~3~~ 4

result = ~~2~~ ~~5~~ 12

start = ~~0~~ 2