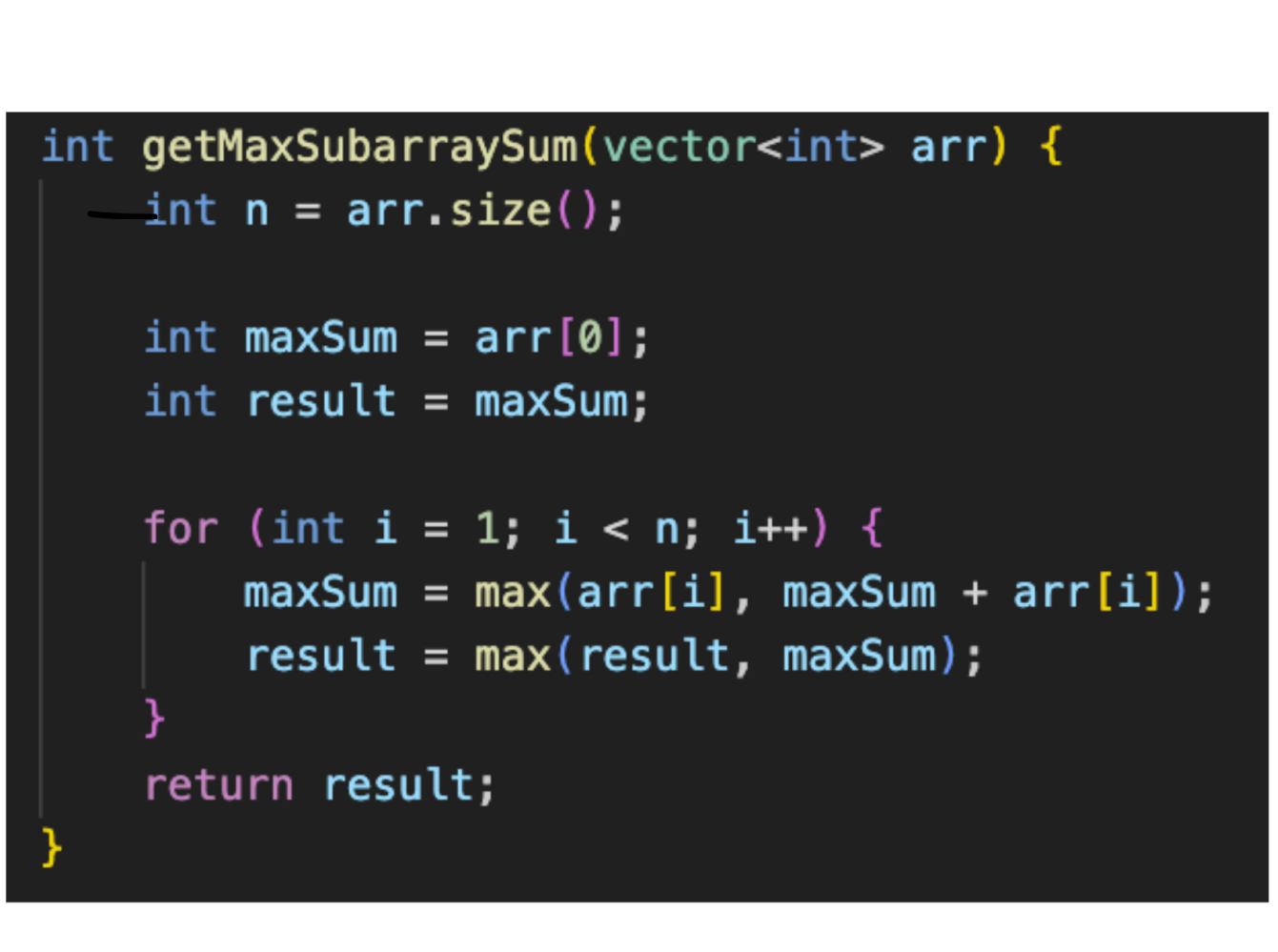


```
int getMaxSubarraySum(vector<int> arr) {
    int n = arr.size();

    int maxSum = arr[0];
    int result = maxSum;

    for (int i = 1; i < n; i++) {
        maxSum = max(arr[i], maxSum + arr[i]);
        result = max(result, maxSum);
    }
    return result;
}
```

[-10, 1, 3, 6, -2, 5, -20, -11, 10].
-10 1 4 10 8 13 -7 -11 10

ms = ~~-10~~. ~~1~~ ~~4~~ ~~10~~ ~~8~~ ~~13~~ ~~-7~~ ~~-11~~ 10
res = ~~-10~~ ~~1~~ ~~4~~ ~~10~~ 13

0 1 2 3 4 5 6 7 8
[-10, 1, 3, 6, -2, 5, -20, -11, 10]
ms: -10 1 4 10 8 13 -7 -11 10
s: 0 1 1 1 1 1 1 7 8
e: ~~0~~

[1, 5].

[1, 2, 3, 4].

[-10, 1, 5, -6].

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

case-1. Max sum subarray is not circular.
- Apply Kadane's → sum = S1

case-2. Max sum subarray is circular
- [Find the min sum. = S2] 2.2.

sum = Array-sum - S2.

return: max (S1, Array-sum - S2).

[1, 3, 2, 5, 6].

Prefix. [1]. [1, 3] [1, 3, 2] ✓
[3, 2] ×

Suffix: [6] [5, 6] [2, 5, 6] ✓
[2, 5] ×.

L = [1, 4, 3, 1]
R: [15, 8, 5, 3].

[1, 15]: [1, 2, 3, 4, 5, 6, 7, 8 - - - 15].
[4, 8] . [4, 5, 6, 7, 8].
[3, 5] : [3, 4, 5].
[1, 3] : [1, 2, 3].

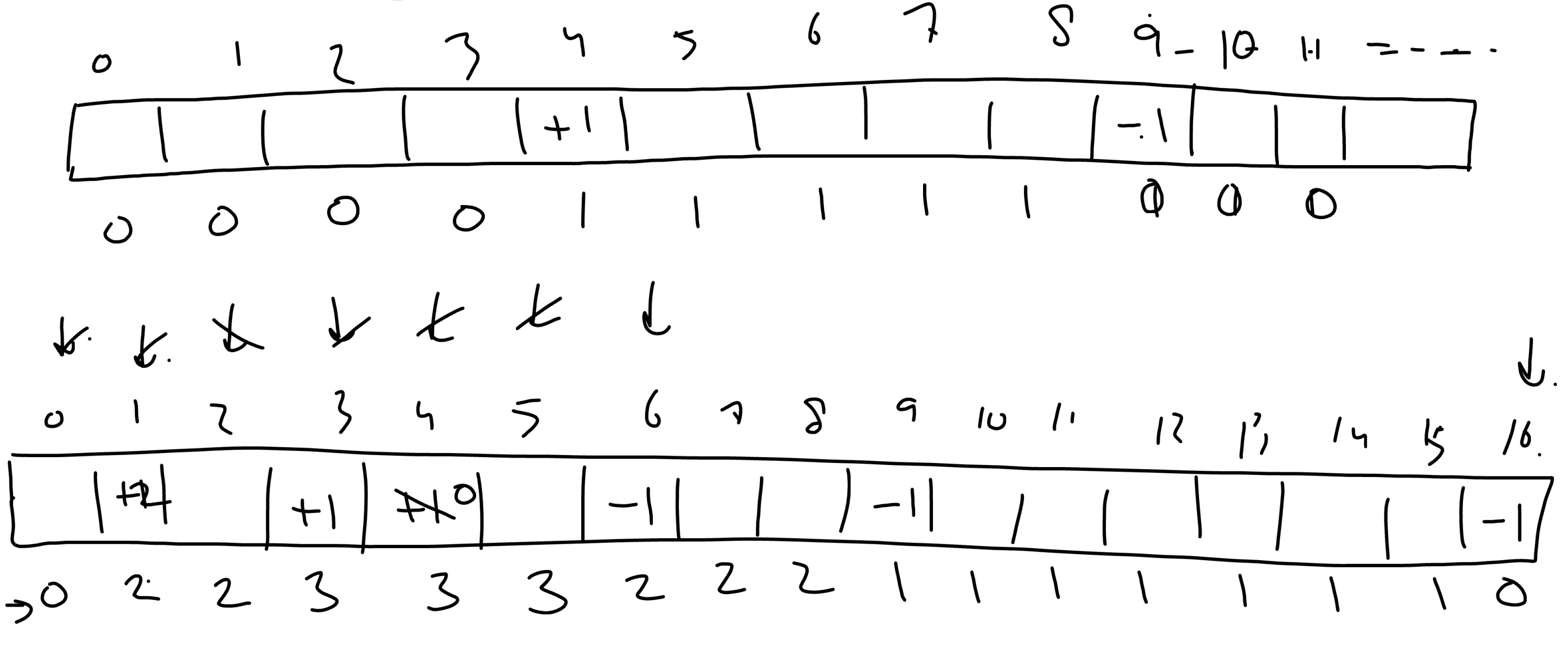
O/P: 3.

TC: O(n* maxx). ← max value in R[].
AS: O(maxx).

$10^6 \times 10^6 = 10^{12}$.

10^9 operations → 1sec.

⇒ [4, 8].



~~ans~~ =
result = ~~2~~ 3

TC: O(n + maxx).
AS: O(maxx).

---

nums: [1, 2, 3, 4].
ans: [24, 12, 8, 6]
x p y

xy fits in a 32-bit int.

nums: [-1, 1, 0, -3, 3].
ans: [0, 0, 9, 0, 0]

[-1, 1, 0, -3, 0].
ans. [0, 0, 0, 0, 0].

TC: O(n²).
AS: O(1)] BF.

[1, 2, 3, 4]
PP: [1, 2, 6, 24].
SP: [24, 24, 12, 4.]
ans: [24 12 8 6].

PP[i] = PP[i-1] × arr[i]
SP[i] = arr[i] × SP[i+1].
ans[i] = PP[i-1] × SP[i+1].

⇓

PP[i] : $a_0 \times a_1 \times a_2$ - - - - × a_{i-1}
→ SP[i] : $a_{i+1} \times a_{i+2}$ - - - - × a_{n-1}
ans[i] = PP[i] × SP[i].

TC: O(n + n + n) = O(n).
AS: O(n)

- . . - i-2 i-1, i - - . .
PP:

PP[i] = PP[i-1]. × arr(i-1]

SP[i+1] = $a_{i+2} \times a_{i+3}$ - - - . × a_{n-1}.
SP[i] = $a_{i+1} \times a_{i+2} \times a_{i+3}$ - - - - × a_{n-1}.
SP[i] = a_{i+1} × SP[i+1].