24. Max Circular Subarray Sum

Hard Accuracy: 45.16% Submissions: 18928 Points: 8

Given an array **arr[]** of **N** integers arranged in a **circular** fashion. Your task is to find the **maximum contiguous subarray sum**.

Example 1:

Input:
N = 7
arr[] = {8,-8,9,-9,10,-11,12}
Output:
22
Explanation:
Starting from the last element of the array, i.e, 12, and moving in a circular fashion, we have max subarray as 12, 8, -8, 9, -9, 10, which gives maximum sum as 22.

Example 2:

Input:
N = 8
arr[] = {10,-3,-4,7,6,5,-4,-1}
Output:
23
Explanation: Sum of the circular subarray with maximum sum is 23

Your Task:
The task is to complete the function **circularSubarraySum()** which returns a sum of the circular subarray with maximum sum.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(1).

Constraints:
$1 <= N <= 10^6$
$-10^6 <= Arr[i] <= 10^6$

① Max sum subarray
need on a contiguous form
that could be found using
Kadane's algo

② ① Since the array is in
circular manner, therefore
other the case could be
② : max sum sub
array from beginning
and ending

Kadane's Algorithm

```
//Standard Kadane's algorithm to find maximum suba
int kadane(int a[], int n)
{
    int max_so_far = 0, max_ending_here = 0;
    int i;
    for (i = 0; i < n; i++)
    {
        //Storing max sum till current index.
        max_ending_here = max_ending_here + a[i];
        //If max sum till current index is negativ
        if (max_ending_here < 0)
            max_ending_here = 0;
        //Storing the max sum so far.
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}
```

① Abandoning baggage like of some negative in duration. This will only reduce the max sum end.

To maintain the max sum throughout the array that has been found.

```cpp
//Function to find maximum circular subarray sum.
int circularSubarraySum(int a[], int n)
{
    bool flag  = false;
    int count =0;int maxx = INT_MIN;
    for(int i = 0; i < n; i++)
    {
        //Storing the maximum element in the array.
        if(a[i] > maxx)
          maxx = a[i];
        //Counting total number of negative numbers in the array.
        if(a[i] < 0)
          count++;
    }
    if(count == n)
      return maxx;

    //Case 1:We get the maximum sum using standard Kadane's algorithm.
    int max_kadane = kadane(a, n);

    //Case 2:We now find the maximum sum that includes corner elements.
    int max_wrap = 0, i;
    for (i = 0; i < n; i++)
    {
            //Calculating total sum of array elements.
            max_wrap += a[i];
            //Inverting the sign of array elements.
            a[i] = -a[i];
    }

    //Maximum sum with corner elements will be:
    //Total sum of array elements-(-max subarray sum after changing
    //sign of array elements).
    max_wrap = max_wrap + kadane(a, n);

    //The maximum circular subarray sum will be maximum of two sums.
    return (max_wrap > max_kadane)? max_wrap: max_kadane;
}
```

① when all the array elements are one, then on addition would uneven the sum simply return the max value.

② for the case ①

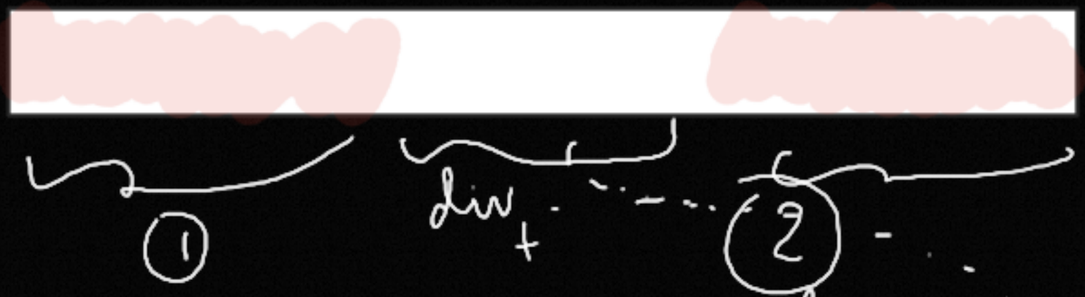
③ Approach to find sum of max circular array i.e., case ②

next page

```
//Case 2:We now find the maximum sum that includes corner elements.
int max_wrap = 0, i;
for (i = 0; i < n; i++)
{
        //Calculating total sum of array elements.
        max_wrap += a[i];
        //Inverting the sign of array elements.
        a[i] = -a[i];
}

//Maximum sum with corner elements will be:
//Total sum of array elements-(-max subarray sum after changing
//sign of array elements).
max_wrap = max_wrap + kadane(a, n);

//The maximum circular subarray sum will be maximum of two sums.
return (max_wrap > max_kadane)? max_wrap: max_kadane;
```



① This could be calculated by

Total array sum - Min sum possible for part dividing it

Run kadane on the array with all the elements sign inverted → Subarr chosen in this case would be smallest in original array