Maximum Circular Subarray Sum

In this problem, there can be two cases i.e. either the subarray having maximum sum is obtained in a circular fashion or the subarray having maximum sum is obtained in a non-circular fashion.

The non-circular maximum sum subarray can be obtained directly by Kadane's Algorithm.

But the subarray with circular fashion cannot be solved by Kadane's algorithm.

So to solve this problem we will first calculate the maximum sum subarray using Kadane's Algorithm and store it in a variable, say candidate1. Then we will invert the sign of every element of the array and again apply Kadane's Algorithm to calculate the maximum sum subarray of this new inverted array.

subtracting is like, removing the array with the smallest sum from the original array

But we should keep in mind the fact that the maximum sum of the inverted subarray is actually the minimum sum subarray for the original array. So if we subtract the negative value of this new sum from the cumulative sum of the original array we will get another candidate for the answer and name it candidate2. Then compare both the candidates and return the larger number.

```
// C/C++ program for maximum contiguous circular sum problem
#include<stdio.h>
using namespace std;
// Standard Kadane's algorithm to find maximum subarray
// sum
int kadane(int a[], int n);

// The function returns maximum circular contiguous sum
// in a[]
int maxCircularSum(int a[], int n)
{
// Case 1: get the maximum sum using standard kadane'
// s algorithm
int max_kadane = kadane(a, n);

// Case 2: Now find the maximum sum that includes
// corner elements.
int max_wrap = 0, i;
for (i=0; i<n; i++)
{
        max_wrap += a[i]; // Calculate array-sum
        a[i] = -a[i]; // invert the array (change sign)
}

// max sum with corner elements will be:
// array-sum - (-max subarray sum of inverted array)
max_wrap = max_wrap + kadane(a, n);

// The maximum circular sum will be maximum of two sums
```

```
return (max_wrap > max_kadane)? max_wrap: max_kadane;
}

// Standard Kadane's algorithm to find maximum subarray sum
// See https://www.geeksforgeeks.org/archives/576 for details
int kadane(int a[], int n)
{
    int max_so_far = 0, max_ending_here = 0;
    int i;
    for (i = 0; i < n; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_ending_here < 0)
            max_ending_here = 0;
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}

/* Driver program to test maxCircularSum() */
int main()
{
    //int a[] = {10, -3, -4, 7, 6, 5, -4, -1};
    //int n = sizeof(a)/sizeof(a[0]);
    int t;
    scanf("%d", &t);
    //cin>>t;
    for(int i=0;i<t;i++)
    {
       int n;
       scanf("%d", &n);
       //cin>>n;
       int a[n];
       for(int j=0;j<n;j++)
       {
        scanf("%d", &a[j]);
        //cin>>a[j];
       }
       printf("%d \n",
                              maxCircularSum(a, n));
    }

    return 0;
}
```

Let us take an example and dry run our code.
For an array, say 10, -3, -4, 7, 6, 5, -4, -1
Cumulative Sum of the original array is 16
candidate1 using Kadane's algorithm on this array is 21 i.e from index 0 to 5
Now inverting this array it becomes -10, 3, 4, -7, -6, -5, 4, 1

Now applying Kadane's algorithm on this new array we get the sum as 7 i.e from index 1 to 2.
So candidate2 will be cumulative_sum - (-7) i.e 16+7 which is 23
So candidate2 is greater than candidate1 so answer is 23
Here we can conclude that this algorithm works because it explains the fact the array's sum could have been maximum if the minimum sum elements sum were not present so we should exclude it.