Maximum Sum Circular Subarray

Given a circular integer array, find a subarray with the largest sum in it.

For example,

```
Input:  {2, 1, -5, 4, -3, 1, -3, 4, -1}

Output: Subarray with the largest sum is {4, -1, 2, 1} with sum 6.


Input:  {-3, 1, -3, 4, -1, 2, 1, -5, 4}

Output: Subarray with the largest sum is {4, -1, 2, 1} with sum 6.
```

Practice this problem

The problem differs from the problem of finding the maximum sum circular subsequence.

Unlike subsequences, subarrays are required to occupy consecutive positions within the original array.

The idea is to find the sequence which will have a maximum negative value. If we remove that minimum sum sequence from the input sequence, we will be left with the maximum sum circular sequence. Finally, return the maximum of the maximum-sum circular sequence (includes corner elements) and maximum-sum non-circular sequence.

For example, consider array `{2, 1, -5, 4, -3, 1, -3, 4, -1}`. The sequence having maximum negative value is `{-5, 4, -3, 1, -3}`, i.e., `-6`. If we remove this minimum sum sequence from the array, we will get the maximum sum circular sequence, i.e., `{2, 1, 4, -1}` having sum `6`. Since the maximum sum circular sequence is greater than the maximum sum non-circular sequence, i.e., `{4}` for the given array, it is the answer.

We can find the maximum-sum non-circular sequence in linear time by using Kadane's algorithm. We can find a maximum-sum circular sequence by inverting the sign of all array elements and then applying Kadane's algorithm.

For example, if we invert signs of array `{2, 1, -5, 4, -3, 1, -3, 4, -1}`, we get `{-2, -1, 5, -4, 3, -1, 3, -4, 1}` which has maximum sum sequence `{5, -4, 3, -1, 3}` having sum `6`. Now inverting the signs back, we get a minimum sum sequence `{-5, 4, -3, 1, -3}` having sum `-6`. The algorithm can be implemented as follows in C++, Java, and Python:

C++

```
#include <iostream>
#include <numeric>
#include <algorithm>
using namespace std;

// Function to find contiguous subarray with the largest sum
// in a given set of integers
int kadane(int arr[], int n)
{
    // stores the sum of maximum subarray found so far
    int max_so_far = 0;

```

```
    // stores the maximum sum of subarray ending at the current position
    int max_ending_here = 0;

    // traverse the given array
    for (int i = 0; i < n; i++)
    {
        // update the maximum sum of subarray "ending" at index `i` (by ad
        // current element to maximum sum ending at previous index `i-1`)
        max_ending_here = max_ending_here + arr[i];

        // if the maximum sum is negative, set it to 0 (which represents
        // an empty subarray)
        max_ending_here = max(max_ending_here, 0);

        // update result if the current subarray sum is found to be greate
        max_so_far = max(max_so_far, max_ending_here);
    }

    return max_so_far;
}

// Function to find the maximum sum circular subarray in a given array
int runCircularKadane(int arr[], int n)
{
    // empty array has sum of 0
    if (n == 0) {
        return 0;
    }

    // find the maximum element present in a given array
    int max_num = *max_element(arr, arr + n);

    // if the array contains all negative values, return the maximum eleme
    if (max_num < 0) {
        return max_num;
    }

    // negate all the array elements
    for (int i = 0; i < n; i++) {
        arr[i] = -arr[i];
    }

    // run Kadane's algorithm on the modified array
    int neg_max_sum = kadane(arr, n);

    // restore the array
    for (int i = 0; i < n; i++) {
        arr[i] = -arr[i];
    }

    /* Return the maximum of the following:
        1. Sum returned by Kadane's algorithm on the original array.
        2. Sum returned by Kadane's algorithm on the modified array +
           the sum of all the array elements.
    */

```

```
69      return max(kadane(arr, n), accumulate(arr, arr + n, 0) + neg_max_sum);
70  }
71
72  int main()
73  {
74      int arr[] = { 2, 1, -5, 4, -3, 1, -3, 4, -1 };
75      int n = sizeof(arr)/sizeof(arr[0]);
76
77      cout << "The sum of the subarray with the largest sum is " <<
78              runCircularKadane(arr, n);
79
80      return 0;
81  }
```

Download Run Code

Output:

```
The sum of the subarray with the largest sum is 6
```

Java

Python

The time complexity of the above solution is `O(n)` and doesn't require any extra space, where `n` is the size of the input.

Array

Algorithm, Hard