Kadane's Algorithm

Difficulty: **Medium** Accuracy: **36.28%** Submissions: **1.2M** Points: **4**

Average Time: **20m**

You are given an integer array **arr[].** You need to find the **maximum** sum of a subarray (containing at least one element) in the array **arr[]**.

Note : A **subarray** is a continuous part of an array.

Examples:

Input: arr[] = [2, 3, -8, 7, -1, 2, 3]
Output: 11
Explanation: The subarray [7, -1, 2, 3] has the largest sum 11.

Input: arr[] = [-2, -4]
Output: -2
Explanation: The subarray [-2] has the largest sum -2.

C++ (12) | Start Timer

```cpp
class Solution {
  public:
    int maxSubarraySum(vector<int> &arr) {
        // Code here
        int n=arr.size();
        int ans=arr[0];
        int large=arr[0];

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

        return ans;
    }
};
```

Custom Input | Compile & Run

Maximum Product Subarray

Difficulty: **Medium** Accuracy: **18.09%** Submissions: **476K+** Points: **4**

Given an array **arr[]** that contains positive and negative integers (may contain 0 as well). Find the **maximum** product that we can get in a subarray of **arr[]**.

Note: It is guaranteed that the answer fits in a 32-bit integer.

Examples

Input: arr[] = [-2, 6, -3, -10, 0, 2]
Output: 180
Explanation: The subarray with maximum product is [6, -3, -10] with product = 6 * (-3) * (-10) = 180.

Input: arr[] = [-1, -3, -10, 0, 6]
Output: 30
Explanation: The subarray with maximum product is [-3, -10] with product = (-3) * (-10) = 30.

C++ (12) Start Timer

```cpp
class Solution {
  public:
    int maxProduct(vector<int> &arr) {
        // code here
        int n=arr.size();
        int ans=arr[0];
        int large=arr[0];
        int small=arr[0];

        for(int i=1;i<n;i++)
        {
            //used templ,temps to store old value before upda
            int templ=large;
            int temps=small;
            large=max(templ*arr[i],max(arr[i],temps*arr[i]));
            small=min(templ*arr[i],min(arr[i],temps*arr[i]));
            ans=max(large,ans);
        }

        return ans;
    }
};
```

Custom Input Compile & Run Sub

Max Circular Subarray Sum

Difficulty: **Hard** Accuracy: **29.37%** Submissions: **179K+** Points: **8**

Average Time: **25m**

You are given a circular array **arr[]** of integers, find the **maximum** possible sum of a non-empty **subarray**. In a circular array, the subarray can start at the end and wrap around to the beginning. Return the maximum non-empty subarray sum, considering both non-wrapping and wrapping cases.

Examples:

Input: 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.

Input: arr[] = [10, -3, -4, 7, 6, 5, -4, -1]
Output: 23
Explanation: Maximum sum of the circular subarray is 23. The

C++ (12) Start Timer

```cpp
class Solution {
  public:
    int maxCircularSum(vector<int> &arr) {
        // code here
        int n=arr.size();
        int l = arr[0],x=0;
        int s = arr[0],y=0;
        int total=0;

        for(int i=0;i<n;i++)
        {
            total+=arr[i];

            x=max(arr[i],x+arr[i]);
            l=max(x,l);

            y=min(arr[i],y+arr[i]);
            s=min(y,s);
        }

        if(l<0){return l;}

        return max(l,total-s);

    }
};
```

Custom Input Compile & Run Sub