Kadane Max Sum Subarray C++

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

int maxSubArraySum(const int arr[], int n) {
    int currentSum = arr[0]; // Initialize current sum
and overall sum
    int overallSum = arr[0];

    for (int i = 1; i < n; i++) {
        if (currentSum >= 0) {
            currentSum += arr[i]; // Add current element
to current sum if positive
        } else {
            currentSum = arr[i]; // Start new subarray if
current sum is negative
        }

        if (currentSum > overallSum) {
            overallSum = currentSum; // Update overall
sum if current sum is greater
        }
    }

    return overallSum; // Return maximum sum found
}

int main() {
    const int arr[] = {5, 6, 7, 4, 3, 6, 4}; // Input array
    int n = sizeof(arr) / sizeof(arr[0]); // Determine the
number of elements in the array

    cout << maxSubArraySum(arr, n) << endl; //
Output maximum sum of subarray
    return 0;
}
```

Dry Run of the Program

Let's break down how the program works with the input array {5, 6, 7, 4, 3, 6, 4}.

Input:

- arr[] = {5, 6, 7, 4, 3, 6, 4}
- n = 7 (the size of the array)

Initialization:

- currentSum = arr[0] = 5 (initialize current sum with the first element)
- overallSum = arr[0] = 5 (initialize overall sum with the first element)

Now, we iterate over the array starting from index 1.

Iteration:

1. **i = 1** (element = 6):
   - currentSum = 5, which is positive.
   - Add 6 to currentSum: currentSum = 5 + 6 = 11.
   - Since currentSum = 11 is greater than overallSum = 5, update overallSum = 11.
2. **i = 2** (element = 7):
   - currentSum = 11, which is positive.
   - Add 7 to currentSum: currentSum = 11 + 7 = 18.
   - Since currentSum = 18 is greater than overallSum = 11, update overallSum = 18.
3. **i = 3** (element = 4):
   - currentSum = 18, which is positive.
   - Add 4 to currentSum: currentSum = 18 + 4 = 22.
   - Since currentSum = 22 is greater than overallSum = 18, update overallSum = 22.
4. **i = 4** (element = 3):
   - currentSum = 22, which is positive.
   - Add 3 to currentSum: currentSum = 22 + 3 = 25.
   - Since currentSum = 25 is greater than overallSum = 22, update overallSum = 25.
5. **i = 5** (element = 6):
   - currentSum = 25, which is positive.
   - Add 6 to currentSum: currentSum = 25 + 6 = 31.
   - Since currentSum = 31 is greater than overallSum = 25, update overallSum = 31.
6. **i = 6** (element = 4):
   - currentSum = 31, which is positive.

	<ul><li>Add 4 to currentSum: currentSum = 31 + 4 = 35.</li><li>Since currentSum = 35 is greater than overallSum = 31, update overallSum = 35.</li></ul>**Final Result:**<ul><li>The maximum sum of the subarray is 35.</li></ul>
Output:- 35	