Hailstone Sequence

In the Hailstone Sequence, also known as the Collatz Conjecture, you start with a positive integer and repeatedly apply the following rules:

- If the number is even, divide it by 2.
- If the number is odd, multiply it by 3 and add 1.

The conjecture states that no matter what positive integer you start with, the sequence will always reach 1 eventually.

Write a C program to generate the Hailstone Sequence for a given positive integer using **recursion**. Your program should:

- Take a positive integer input from the user.
- Use recursion to generate the Hailstone Sequence for that input.
- Print out each number in the sequence until you reach 1.
- You should use struct for this question

Here are some additional requirements and tips:

- Your program should handle input values up to at least 2^31-1 (the maximum value of a 32-bit signed integer).
- Make sure your program handles invalid input gracefully (e.g., negative integers, zero, or non-integer input).
- Consider using a helper function to implement the recursion.
- Test your program with different input values to make sure it works correctly.

Test Case:

Input: 5 Expected Output: 5 16 8 4 2 1

Input: 17 Expected Output: 17 52 26 13 40 20 10 5 16 8 4 2 1

Input: 10 Expected Output: 10 5 16 8 4 2 1

Input: 1 Expected Output: 1

Input: -3 Expected Output: Error: Invalid input. Please enter a positive integer.

Input: 0 Expected Output: Error: Invalid input. Please enter a positive integer.

Input: 2147483648 (one more than the maximum 32-bit signed integer value) Expected Output: Error: Invalid input. Please enter a positive integer.

Maximum Subarray

Write a function in C that takes an array of integers and the length of the array as input, and returns the maximum sum of any subarray of the array. A subarray of an array is a contiguous sequence of elements.

For example, if the input array is {1, -2, 3, 5, -3, 2}, the maximum sum of any subarray is 8, which corresponds to the subarray {3, 5}.

Your function should have the following prototype: int max_subarray_sum(int arr[], int n); where `arr` is a pointer to the first element of the array, and `n` is the number of elements in the array. Here's a test case for the `max_subarray_sum` function with the input array {1, -2, 3, 5, -3, 2}:

- {1} (sum = 1)
- {-2} (sum = -2)
- {3} (sum = 3)
- {5} (sum = 5)
- {-3} (sum = -3)
- {2} (sum = 2)
- {1, -2} (sum = -1)
- {-2, 3} (sum = 1)
- {3, 5} (sum = 8)
- {5, -3} (sum = 2)
- {-3, 2} (sum = -1)
- {1, -2, 3} (sum = 2)
- {-2, 3, 5} (sum = 6)
- {3, 5, -3} (sum = 5)
- {5, -3, 2} (sum = 4)
- {1, -2, 3, 5} (sum = 7)

- {-2, 3, 5, -3} (sum = 3)
- {3, 5, -3, 2} (sum = 7)
- {1, -2, 3, 5, -3} (sum = 4)
- {-2, 3, 5, -3, 2} (sum = 5)

So the answer is the maximum: - {3, 5} (sum = 8)

Note 1: Please keep in mind that we're looking for the contiguous subarray within the given array that has the largest sum. This means that the elements of the subarray must be adjacent to each other in the original array.

Note 2: A recursive implementation, along with comments and explanations, can earn bonus points.