Why is it O(n)?

The function has a single loop that iterates over the array elements. Inside the loop, we perform a constant number of operations:

1. We compare the current element with the maximum value found so far (if (arr[i] > MaxValue)).

2. If the current element is greater, we update the maximum value (MaxValue = arr[i]).

These operations are performed for each element in the array, and the number of operations grows linearly with the size of the input array (n).

Key factors contributing to O(n) complexity:

1. Single loop: The function has only one loop, which iterates over the array elements.

2. Linear iteration: The loop visits each element exactly once, resulting in a linear increase in the number of operations with respect to the input size (n).

3. Constant operations: The number of operations performed inside the loop is constant and does not depend on the input size (n).

Why not O(1), O(log n), or O(n^2)?

1. O(1): The function's runtime does not remain constant, as it depends on the size of the input array (n).

2. O(log n): The function does not use a divide-and-conquer approach or any other technique that would reduce the problem size by a factor of 2 with each iteration.

3. O(n^2): The function has only a single loop, and the number of operations does not grow quadratically with the input size (n).

In summary, the findMax function has a time complexity of O(n) because it has a single loop that iterates over the array elements, performing a constant number of operations for each element.

