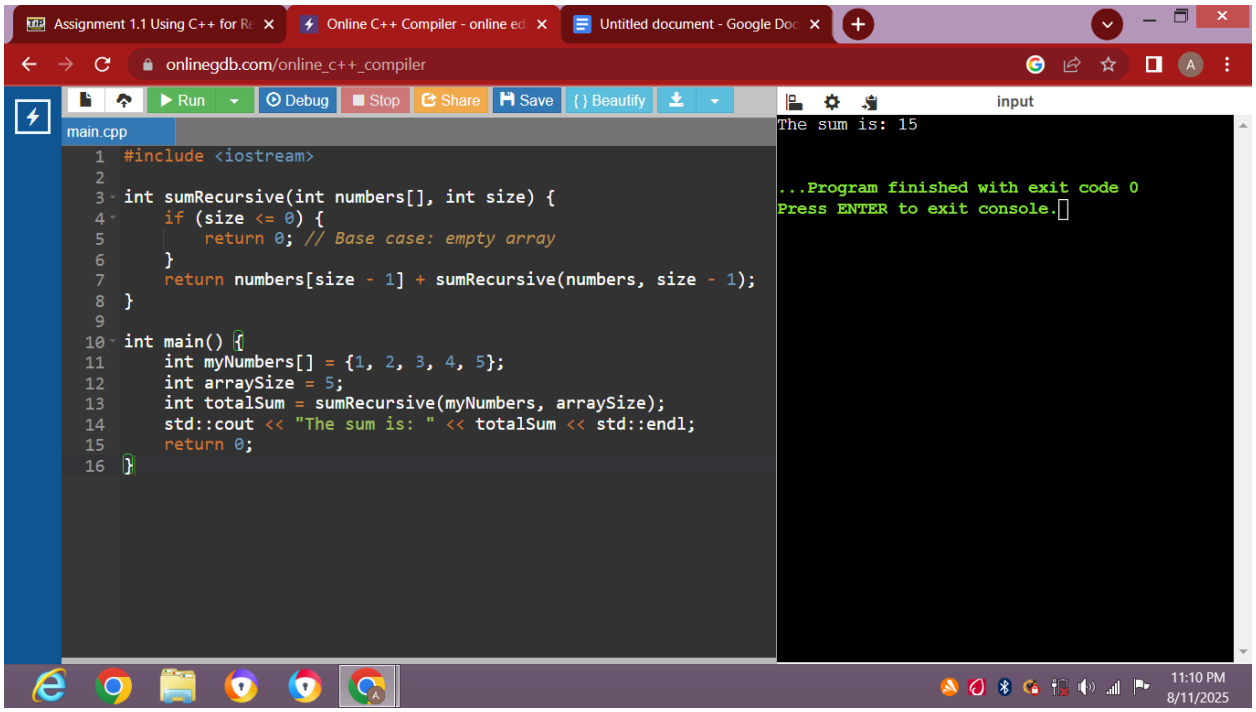
Task 1:

Recursive Solution

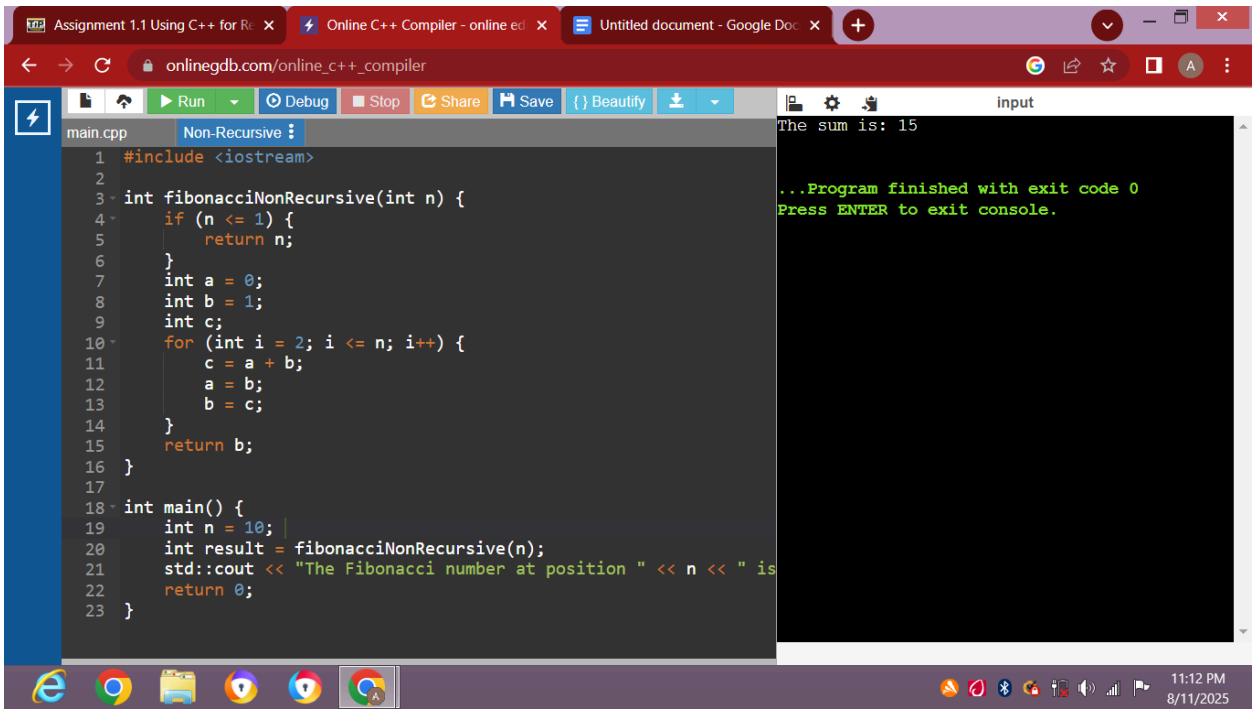



```cpp
#include <iostream>

int sumRecursive(int numbers[], int size) {
    if (size <= 0) {
        return 0; // Base case: empty array
    }
    return numbers[size - 1] + sumRecursive(numbers, size - 1);
}

int main() {
    int myNumbers[] = {1, 2, 3, 4, 5};
    int arraySize = 5;
    int totalSum = sumRecursive(myNumbers, arraySize);
    std::cout << "The sum is: " << totalSum << std::endl;
    return 0;
}
```

Non-Recursive Solution




```cpp
#include <iostream>

int fibonacciNonRecursive(int n) {
    if (n <= 1) {
        return n;
    }
    int a = 0;
    int b = 1;
    int c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

int main() {
    int n = 10;
    int result = fibonacciNonRecursive(n);
    std::cout << "The Fibonacci number at position " << n << " is
    return 0;
}
```

Analysis of Big-O Notation

Task 1: Addition of Numbers in List
Recursive Solution: For the list of numbers, this function calls itself once for each element in the list. Therefore, if the number of elements is N, then there will be N function calls. Each call does constant work (an addition and a return). So, the time will be O(N) and for space also it will be O(N), because of depth of call stack.

Non-Recursive Solution: The code has a simple for loop iterating through the element of the list once. For a list of size N, loop runs N times. This makes the time complexity O(N). As for space complexity, it is O(1), using very few variables to store the sum and loop counter for the computation, regardless of the list size.

Task 2: Fibonacci

Recursive solution: This is a classical example of an inefficient recursive algorithm. To compute fibonacci(n), it needs to compute fibonacci(n-1) and fibonacci(n-2). Now each of these makes further calls, resulting in the formation of a tree of function calls, which grows exponentially. Thus, time complexity is O(2^N). Space complexity is O(N), since N is the maximum depth of the recursion stack.

Non-Recursive Solution: This solution includes a loop that runs from 2 to N. Inside the loop, a fixed number of computations are done. For a given N, the loop is iterated N−1 times, thus giving O(N) time complexity. Space complexity is O(1) because it needs to store only a few variables irrespective of the size of input N.