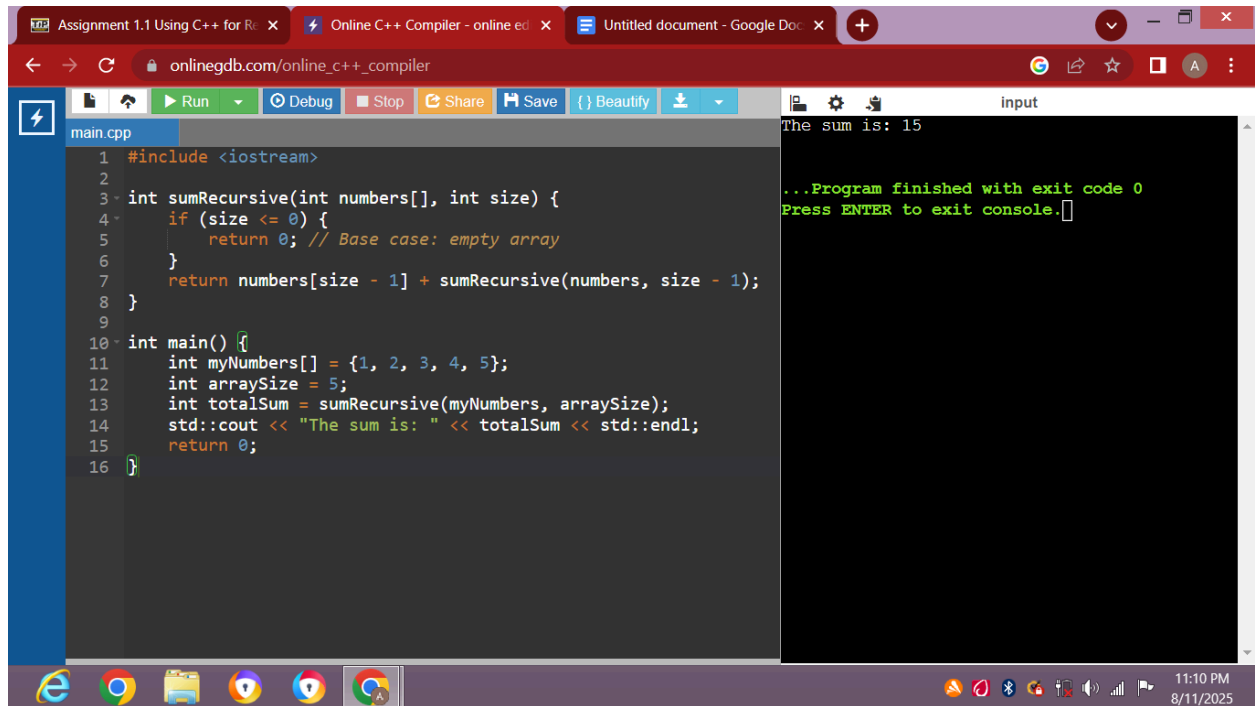


Task 1:

Recursive Solution



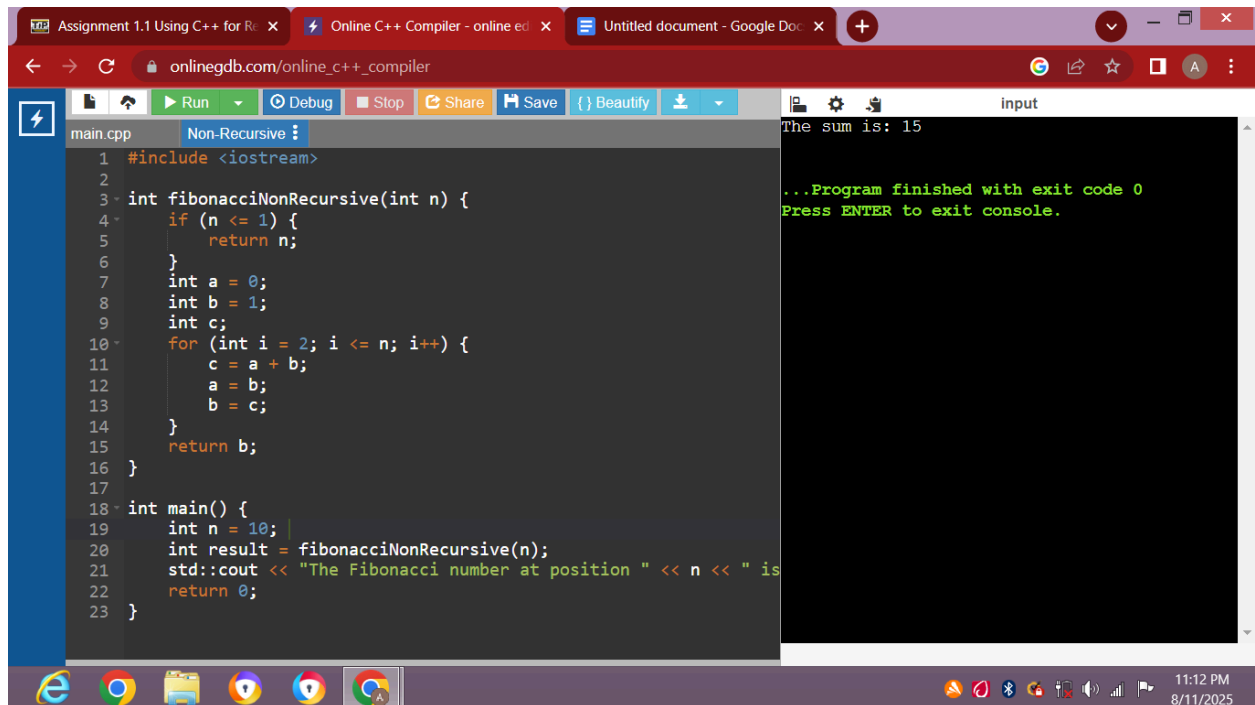
The screenshot shows a web browser with three tabs: "Assignment 1.1 Using C++ for Re...", "Online C++ Compiler - online ed...", and "Untitled document - Google Doc...". The address bar shows "onlinegdb.com/online_c++_compiler". The compiler interface has a toolbar with buttons for Run, Debug, Stop, Share, Save, and Beautify. The code editor on the left shows a C++ program in "main.cpp" that uses a recursive function to calculate the sum of an array. The output window on the right shows the result of the program execution.

```
1 #include <iostream>
2
3 int sumRecursive(int numbers[], int size) {
4     if (size <= 0) {
5         return 0; // Base case: empty array
6     }
7     return numbers[size - 1] + sumRecursive(numbers, size - 1);
8 }
9
10 int main() {
11     int myNumbers[] = {1, 2, 3, 4, 5};
12     int arraySize = 5;
13     int totalSum = sumRecursive(myNumbers, arraySize);
14     std::cout << "The sum is: " << totalSum << std::endl;
15     return 0;
16 }
```

The output window displays the following text:

```
The sum is: 15
...Program finished with exit code 0
Press ENTER to exit console.
```

Non-Recursive Solution



The screenshot shows the same online C++ compiler interface as the previous one, but with a different C++ program. The code editor shows a non-recursive function to calculate the Fibonacci number at a given position. The output window shows the result of the program execution.

```
1 #include <iostream>
2
3 int fibonacciNonRecursive(int n) {
4     if (n <= 1) {
5         return n;
6     }
7     int a = 0;
8     int b = 1;
9     int c;
10    for (int i = 2; i <= n; i++) {
11        c = a + b;
12        a = b;
13        b = c;
14    }
15    return b;
16 }
17
18 int main() {
19     int n = 10;
20     int result = fibonacciNonRecursive(n);
21     std::cout << "The Fibonacci number at position " << n << " is " << result << std::endl;
22     return 0;
23 }
```

The output window displays the following text:

```
The sum is: 15
...Program finished with exit code 0
Press ENTER to exit console.
```

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 $\text{fibonacci}(n)$, it needs to compute $\text{fibonacci}(n-1)$ and $\text{fibonacci}(n-2)$. 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 .