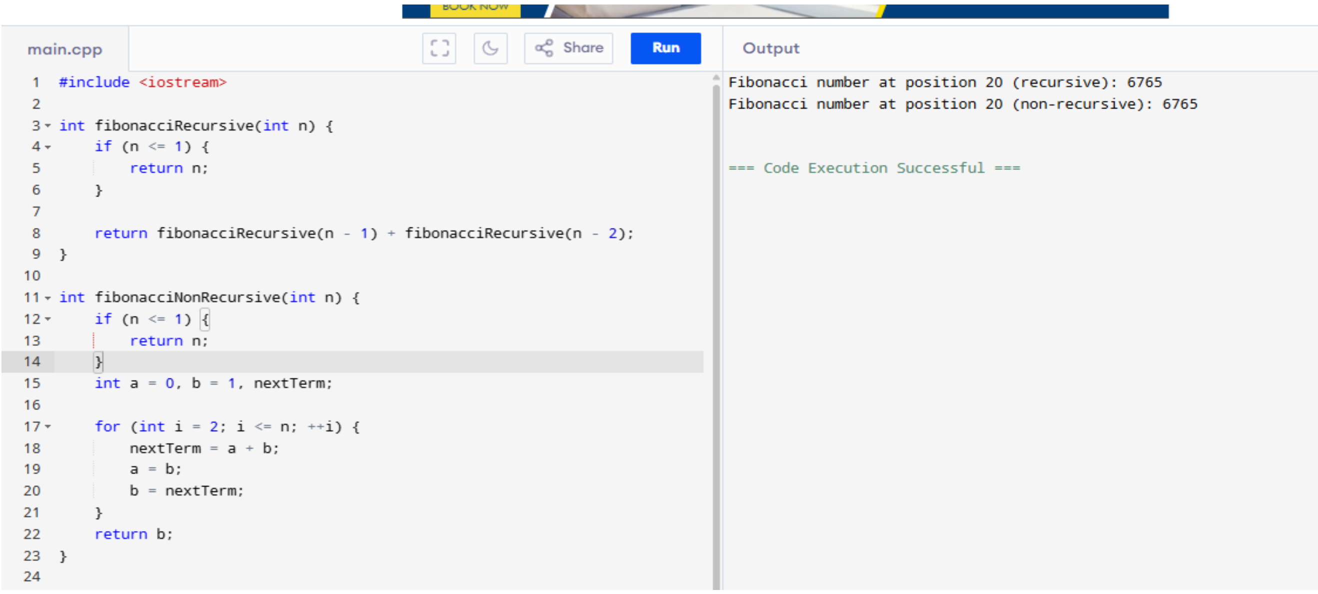
Assignment 1.1 Using C++ for Recursion



Analyzing using the Big-O Notation:

Task 1: Summing a List of numbers

1. The first task the recursive solution can be explained by whereas the 'n' is the number of elements in the list, is the time complexity. This is due to the fact that the function executes a fixed amount of work which means the addition at each step, calling itself once for every element. The recursive call stacks, which holds 'n' function frames, contributes to the space complexity being O(n).

2. The second task the non-recursive solution can be explained for each element in the list undergoes a fixed number of operations during the algorithm's single iteration. Because the sum variable is only stored in a fixed amount of memory, independent of the size of the input, the space complexity is O(1).

Task 2: Fibonacci

1. In this in recursive solution, it is an exponential complexity because the function makes two recursive calls for each number, leading to a tree-like structure of calls that grows exponentially. The space complexity is O(n), which is the maximum depth of the recursive call stack. This approach is highly inefficient for large values of 'n'.

2. In non-recursive solution, the Fibonacci sequence works as the algorithm uses a simple loop that runs 'n' times, performing a constant number of operations in each iteration. The space complexity is **O(1)**, as it only uses a few variables to store the current and previous terms, irrespective of the value of 'n'.