

Assignment No. 1.1

Using C++ for Recursion

Course Code: CPE010

Program: Computer Engineering

Course Title: Data Structures and Algorithms

Date Performed: 8/12/25

Section: CPE21S4

Date Submitted: 8/12/25

Name(s): Crishen Luper S. Pulgado

Instructor: Jimlord Quejado

Output

Recursive:

The screenshot shows a code editor interface with a tab labeled "main.cpp". The code is as follows:

```
1 #include <iostream>
2
3 int sumNumbers(int arr[], int size){
4     if (size == 1) return arr[0];
5     return arr[size - 1] + sumNumbers(arr, size - 1);
6 }
7
8 int main() {
9     int listNumbers[] = {1,2,3,4,5,6};
10    int size = sizeof(listNumbers) / sizeof(listNumbers[0]);
11    std::cout << "The sum is: " << sumNumbers(listNumbers, size) << "\n";
12 }
13 return 0;
14 }
```

Below the code, there are several icons: a copy icon, a run icon, a share icon, and a "Run" button. To the right, the output window displays "The sum is: 21" and "==== Code Execution".

- The "if (size == 1) return arr[0];" runs in O(1) time. One element is processed at a time, followed by calling itself with size – 1. Then, we have n calls each doing O(1) work and O(n) total. For the space complexity, every recursive call is on the call stack until it finishes O(n) extra memory.

Non-recursive

The screenshot shows a code editor interface with a tab labeled "main.cpp". The code is as follows:

```
1 #include <iostream>
2
3 int sumNumbers(int arr[], int size){
4     int sumNumbers = 0;
5     for (int i = 0; i < size; i++) {
6         sumNumbers += arr[i];
7     }
8     return sumNumbers;
9 }
10
11 int main() {
12     int listNumbers[] = {1,2,3,4,5,6};
13     int size = sizeof(listNumbers) / sizeof(listNumbers[0]);
14     std::cout << "The sum is: " << sumNumbers(listNumbers, size) << "\n";
15
16     return 0;
17 }
```

Below the code, there are several icons: a copy icon, a run icon, a share icon, and a "Run" button. To the right, the output window displays "The sum is: 21" and "==== Code Execution S".

- The time complexity is O(n) since loop runs n times. The space complexity is O(1) since no recursion.

Recursive:

main.cpp				Run	Output
1 #include <iostream> 2 3+ int fibonacci(int n) { 4 if (n == 0) return 0; 5 if (n == 1) return 1; 6 7 return fibonacci(n - 1) + fibonacci(n - 2); 8 } 9 10+ int main() { 11 int n = 10; 12 std::cout << "Recursive Fibonacci (" << n << ") = " 13 << fibonacci(n) << "\n"; 14 } 15					Recursive Fibonacci (10) = 55 ==== Code Execution Successful =====

- Since every call makes a 2 or more calls, then the time complexity is $O(2^n)$. The space complexity is $O(n)$ since it is a recursion

Non-recursive:

main.cpp				Run	Output
1 #include <iostream> 2 3+ int fibonacci(int n) { 4 if (n == 0) return 0; 5 if (n == 1) return 1; 6 7 int a = 0, b = 1, next; 8+ for (int i = 2; i <= n; i++) { 9 next = a + b; 10 a = b; 11 b = next; 12 } 13 return b; 14 } 15 16+ int main() { 17 int n = 10; 18 std::cout << "Iterative Fibonacci (" << n << ") = " 19 << fibonacci(n) << "\n"; --					Iterative Fibonacci (10) = 55 ==== Code Execution Successful =====

- It runs n times in the loop, which is the time complexity is $O(n)$. The space complexity is $O(1)$ since it uses only few variables.