Assignment 1.1	
Using C++ for Recursion	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 08/11/2025
Section: CPE21S4	**Date Submitted:** 08/11/2025
Name(s): Gabriel A. Cabrera	**Instructor:** Engr. Jimlord Quejado

6. Output

1. Create C++ code to implement a recursive and non- recursive solution for the following tasks:
Task 1: Summing a List of Numbers
Task 2: Fibonacci
2. Analyse the above algorithms using Big-O Notation.

Task 1: Summing a List of Numbers

Recursive

main.cpp

```cpp
#include <iostream>
#include <vector>

int sum1(const std::vector<int>& n, int j = 0) {
    if (j >= n.size()) {
        return 0;
    }
    return n[j] + sum1(n, j + 1);
}

int main() {
    std::vector<int> list1 = {45, 32, 63, 74, 85};
    int total_sum = sum1(list1);

    std::cout << "The sum of the list is: " << total_sum << std::endl;

    return 0;
}
```

Output

```
The sum of the list is: 299


=== Code Execution Successful ===
```

Since the function only uses one if statement, the Big-O notation is O(n).

Explanation:
The function takes the list n and takes every element of it first, then iterates depending on the size of the list by using n.size, every iteration adds the previous value of n[j] and adds it to the current value of n[j]. Recursion occurs on the return line of the function, where the sum1 is called by itself.

Non-recursive

main.cpp

```cpp
#include <iostream>
#include <vector>


int sum2(const std::vector<int>& n) {
    int sum = 0;
    for (int j : n) {
        sum += j;
    }
    return sum;
}
int main() {
    std::vector<int> list2 = { 45, 32, 63, 74, 85 };
    int total_sum = sum2(list2);

    std::cout << "The sum of the list is: " << total_sum << std::endl;

    return 0;
}
```

Output

```
The sum of the list is: 299


=== Code Execution Successful ===
```

Since the program only uses one for statement, the Big-O notation is O(n).

Explanation:
Similar to the recursive solution, this solution iterates over the list taken by the function, the int j : n goes over the range which is j to the end of the list n, which is list2 in the main function, and adds it on the same function then prints the total sum at the end of the program. The difference between this and the recursive solution is this one goes over the list at once but the recursive solution repeats the function multiple times depending on the size of n.

Task 2: Fibonacci

Recursive

main.cpp

```cpp
#include <iostream>

long long fib1(int n) {
    if (n <= 1) {
        return n;
    }
    return fib1(n - 1) + fib1(n - 2);
}
int main() {
    int j=0;
    std::cout<<"Please enter the element index for the Fibonacci
        sequence: ";
    std::cin>>j;
    std::cout<<"Element "<<j<<" of the sequence is: "<<fib1(j-1);

    return 0;
}
```

Output

```
Please enter the element index for the Fibonacci sequence: 10
Element 10 of the sequence is: 34

=== Code Execution Successful ===
```

Even though there is only one if statement, depending on the value of the element to be computed, it calls itself two times every execution(fib1(n-1)+fib1(n-2)), therefore increasing the number of O exponentially. Therefore, the Big-O notation of this solution is $O(2^n)$.

Explanation:
Firstly, for the fib1 function, I intended to use int, but when I saw that the number could go up quickly, I used long instead so it can contain bigger numbers. The function calls itself, reducing the value of n by one and two, respectively, and then adds the two new values together. This happens until the base cases are met, which is when n is less than or equal to 1. For the printing, using j-1 instead of j computes for the actual number in the series since sequences start at 1 and not 0.

Non-recursive

main.cpp — Share — Run

```cpp
#include <iostream>
#include <vector>

long long fib2(int n) {
    if (n <= 1) {
        return n;
    }
    std::vector<long long> fib(n + 1);
    fib[0] = 0;
    fib[1] = 1;

    for (int i = 2; i <= n; ++i) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }

    return fib[n];
}
int main() {
    int j = 0;
    std::cout << "Please enter the element index for the Fibonacci sequence: ";
    std::cin >> j;
    std::cout << "Element " << j << " of the sequence is: " << fib2(j - 1);

    return 0;
}
```

Output — Clear

```
Please enter the element index for the Fibonacci sequence: 13
Element 13 of the sequence is: 144

=== Code Execution Successful ===
```

Since there's only one if statement and one for statement in the program, the notation can be written as O(n)+O(n), which can be simplified to O(n).

Explanation:
Similar to the recursive solution, the function is declared as long long so it can store and compute larger numbers. It initiates the fib[] vector to store the initial values for the 0 and 1 index, it then continues to the for part of the function to compute for the fibonacci element given by the user when the program starts. It stops on the condition that n is less than or equal to 1.

7. Conclusion

Through this activity, I realized that problems can be solved through recursive and non-recursive ways, both of them have pros and cons. For a recursive solution, it looks cleaner and more elegant in a way, but it may be harder to execute and in some ways, gives the hardware or the program more load to bear in computing numbers that require multiple iterations. The non-recursive solution is a more direct approach to problems and can easily be executed. Overall, both of these can be useful depending on the case and it's always up to the skill and knowledge of the developer to know which one to use and how to use one over the other more efficiently.

9. Assessment Rubric