

The screenshot shows a C++ development environment with two tabs open: '69.cpp' and '69.exe'. The '69.cpp' tab contains C++ code for two tasks. Task 1 is a sum function using both recursive and iterative versions. Task 2 is a Fibonacci sequence function using both recursive and iterative versions. The '69.exe' tab shows the output of the program, which compares the execution times of the different methods. The recursive versions are significantly slower than their iterative counterparts.

```
2 #include <vector>
3 using namespace std;
4
5 // Task 1: Sum of a List of Numbers
6
7 // Recursive version
8 int sumRecursive(const vector<int>& nums, int n) {
9     if (n == 0) return 0;
10    return nums[n] + sumRecursive(nums, n - 1);
11 }
12
13 // Iterative version
14 int sumIterative(const vector<int>& nums) {
15     int total = 0;
16     for (int num : nums) {
17         total += num;
18     }
19     return total;
20 }
21
22 // Task 2: Fibonacci
23
24 // Recursive version
25 int fibonacciRecursive(int n) {
26     if (n <= 1) return n;
27     return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
28 }
29
30 // Iterative version
31 int fibonacciIterative(int n) {
32     if (n <= 1) return n;
33     int a = 0, b = 1;
34     for (int i = 2; i <= n; i++) {
35         c = a + b;
36         a = b;
37         b = c;
38     }
39     return b;
40 }
41
42 // Main
43 int main() {
44     vector<int> nums = {1, 2, 3, 4, 5};
45     cout << "Sum Recursive: " << sumRecursive(nums, nums.size()) << endl;
46     cout << "Sum Iterative: " << sumIterative(nums) << endl;
47
48     int n = 10;
49     cout << "Fibonacci Recursive (" << n << ")": " << fibonacciRecursive(n) << endl;
50     cout << "Fibonacci Iterative (" << n << ")": " << fibonacciIterative(n) << endl;
51
52     return 0;
53 }
54
```

C:\Users\Joshua\Documents\69.exe
Sum Recursive: 15
Sum Iterative: 15
Fibonacci Recursive (10): 55
Fibonacci Iterative (10): 55
Process exited after 0.0926 seconds with return value 0
Press any key to continue . . .

Explanation:

The Big-O-Notation shows how quick and how much memory an algorithm uses as the input gets larger, focusing on the worst-case-scenario.

Task 1: Summing a List of Numbers

- In the recursive version, the time complexity is O(n) because the function runs once for each item in the list, But the space complexity is O(1) because it uses only a fixed amount of memory.

Task 2: Fibonacci Sequence

- The recursive version, the time complexity is O(2^n) because each number calls the function twice.
- The non recursive version or Iterative version since it loops through numbers once, and the space complexity is O(1) as it only uses one or two variables.