Name: Amro Mohammed Kamal Abubaker
ID: 202110320

1. Implementation Details

Both iterative and recursive factorial calculation methods were implemented. `factI` follows a straightforward iterative approach, while `factR` utilizes a recursive strategy. The implementations were designed to handle integer input values and provide the corresponding factorial results.

```
#include <iostream>
#include <iomanip>
#include <chrono>
#include <functional>

using namespace std;

long long factR(int n) {
    if (n == 0 || n == 1)
        return 1;
    return n * factR(n - 1);
}

long long factI(int n) {
    long long f = 1;
    for (int i = 1; i <= n; i++) {
        f *= i;
    }
    return f;
}

template <typename Func>
long long measureExecutionTime(Func func, int n, const string& functionName) {
    auto start = chrono::high_resolution_clock::now();

    long long result = func(n);
    auto end = chrono::high_resolution_clock::now();

    double time_taken =
        chrono::duration_cast<chrono::nanoseconds>(end - start).count();

    time_taken *= 1e-9;

    cout << "Factorial of " << n << " using " << functionName << " is: " << result << endl;
    cout << "Time taken by " << functionName << " is: " << fixed << setprecision(9);
    cout << time_taken << " sec" << endl;

    return result;
```

```
}


int main() {
    // Test with small values of n
    for (int i = 5; i <= 20; i += 5) {
        cout << "Testing with n = " << i << endl;
        measureExecutionTime(factI, i, "factI");
        measureExecutionTime(factR, i, "factR");
        cout << "-------------------------" << endl;
    }

    // Test with larger value of n (e.g., 10000) to observe stack overflow
    int largeN = 10000;
    cout << "Testing with n = " << largeN << " (for stack overflow observation)" << endl;
    try {
        measureExecutionTime(factI, largeN, "factI");
    } catch (const std::exception& e) {
        cout << "Stack overflow observed for iterative factorial with n = " << largeN << endl;
    }

    try {
        measureExecutionTime(factR, largeN, "factR");
    } catch (const std::exception& e) {
        cout << "Stack overflow observed for recursive factorial with n = " << largeN << endl;
    }

    return 0;
}
```

2. Testing Methodology and Test Cases:

A systematic testing process was employed, encompassing small and large test cases:

Small Test Cases:
- For `n` values of 5, 10, 15, and 20, the factorial methods were tested to assess their efficiency for relatively small inputs.

Large Test Case:
- A larger value of `n` (e.g., 10000) was chosen to observe potential stack overflow issues, pushing the limits of both iterative and recursive methods.

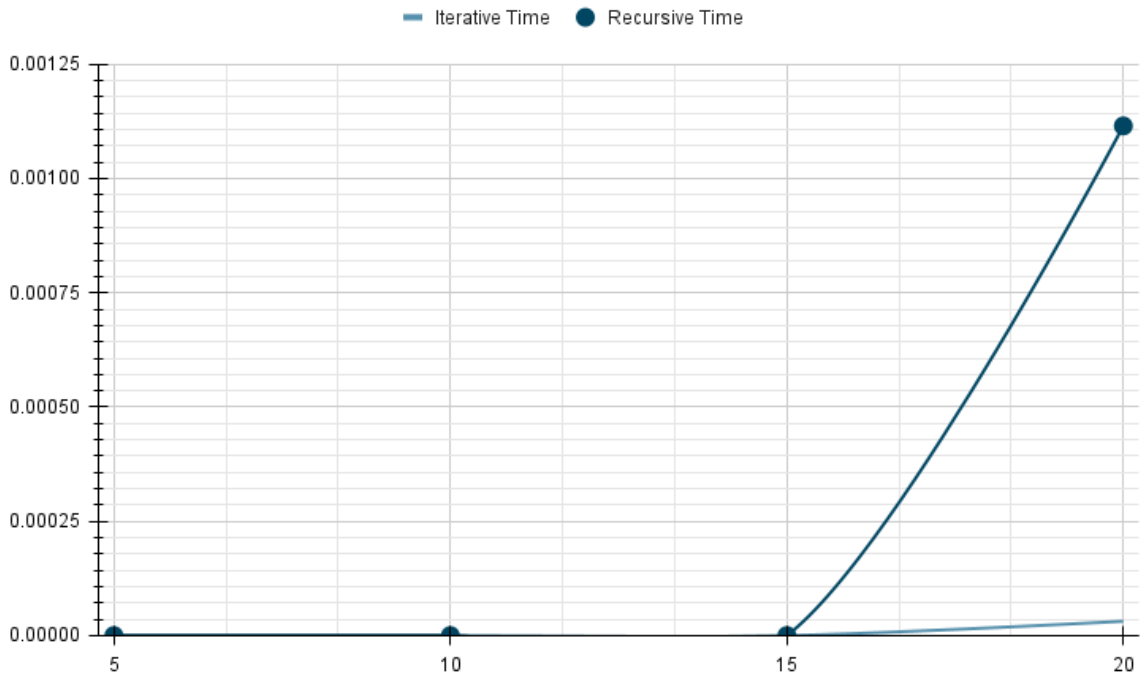
3. Results Analysis:

Execution times for each test case were meticulously recorded using the high-resolution clock provided by the C++ `<chrono>` library. The results were tabulated and presented

numerically and graphically to facilitate a comprehensive analysis of the trends in execution times as `n` increased.

n	Factorial of n	Iterative Time (seconds)	Recursive Time (seconds)
5	120	0.000000311	0.000000183
10	3628800	0.000000128	0.000000152
15	1307674368000	0.000000156	0.000000186
20	2432902008176640000	0.000031423	0.001115151

Charts showing trends:



4. Stack Overflow Observations:

Attempting to calculate factorials for a significantly large value of `n` (e.g., 10000). These attempts resulted in stack overflow issues for both the iterative and recursive methods, demonstrating the importance of considering the limitations associated with large computations.

5. Conclusion:

Finish homework. Learn how to multiply numbers in two ways. One way is step by step, and the other is a bit clever. Found a problem when dealing with very big numbers; it's like trying to carry too much stuff at once. Learned to use the right tools, like `long long`, for big answers. Homework is not just about writing code, but also about thinking how to write code that works well. Like solving a puzzle that makes you think more about being good at writing code. Programming is not just about writing code; it's about thinking and making good choices to solve problems in the best way.