benchmark.cpp

```cpp
/**
 * Benchmarker
 * Author: Arnab Ghosh
 * Date: 11/8/2023
 *
 * Profiles the results of the experiment according to the methodology described in the manuscript
 * Usage: ./benchmark ./eccKEM 1gb_test
*/

#include <iostream>
using std::cin;
using std::cout;

#include <string>
using std::string;

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <ctime>
using std::time;
using std::time_t;

#include <thread>
using std::thread;

#include <future>
using std::async;

#include <chrono>
using std::chrono::milliseconds;


// TODO: find the include header for C++ asynchronous code

/**
 * Generates a performance report, and outputs it to a file.
 * TODO: Finish this function.
*/
void generatePerformanceReport(string filename, int elapsedTime, int totalTimeIntervals, int timeIntervals[], double memoryUsage[], double cpuUsage[]) {
    // Create buffer for report
    ofstream report(filename);

    // Create headers
    report << "Time_Interval;Memory_Usage;CPU_Usage;";
}


/**
 * Returns the current memory usage for the system in megabytes.
 * TODO: Finish this function.
```

```
*/
int currentMemoryUsage() {
    return 0;
}


/**
 * Entry point of program
*/
int main(int argc, char* argv[]) {
    // Parsing arguments
    // We want to join all of the arguments together, and build that as the command
    string command {""};

    // Loop through arguments and append to commandToProfile
    for(int i = 1 /* Skipping the first argument as it will be ./benchmark */; i < argc; i++) {
        command.append(argv[i]);
        command.append(" ");
    }

    // Now we run
    // 1) PERFORMANCE METRIC 1 - TIME
    // Initial time
    time_t initialTime = time(NULL);

    // 2) PERFORMANCE METRIC 2 - MEMORY
    // TODO: implement performance benchmarking metrics
    // Asynchronously call command
    auto future = async(std::launch::async,
        [command] { return system(command.c_str()); // This is an anonymous lambda
    });

    // Profile future
    while( future.wait_for( milliseconds(500) /* Profile for data every 1/2 second*/ )  !=
std::future_status::ready ) {
        // Time to store some data!

    }

    // Calculating time usage
    time_t finalTime = time(NULL);
    time_t deltaTime = finalTime - initialTime;
    cout << "Elapsed (s): " << deltaTime << std::endl;
}
```