ES215: Semester I [2024-2025]
Assignment 1
22110296

1. Implement a program(s) to list the first 50 fibonacci numbers preferably in C/C++ in the following manner:

a. Using recursion
b. Using loop
c. Using recursion and memoization
d. Using loop and memoization

Code:

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

using namespace std;
using namespace std::chrono;

// Recursive Fibonacci function
long long fibRec(int n) {
    if (n <= 1) {
        return n;
    }
    return fibRec(n - 1) + fibRec(n - 2);
}

// Iterative Fibonacci function
```

```
long long fibLoop(int n) {
    long long a = 0, b = 1, c;
    if (n == 0) {
        return a;
    }
    if (n == 1) {
        return b;
    }
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}

// Recursive Fibonacci function with memoization
long long fibRecMemo(int n, vector<long long>
&memo) {
    if (n <= 1) {
        return n;
    }
    if (memo[n] != -1) {
        return memo[n];
    }
```

```
    memo[n] = fibRecMemo(n - 1, memo) + fibRecMemo(n
- 2, memo);
    return memo[n];
}

// Iterative Fibonacci function with memoization
void fibLoopMemo(int n, vector<long long> &memo) {
    memo[0] = 0;
    memo[1] = 1;
    for (int i = 2; i <= n; i++) {
        memo[i] = memo[i - 1] + memo[i - 2];
    }
}

// Function to measure execution time
void measureTime(int n) {
    vector<long long> memo(n + 1, -1);
    vector<long long> arr(n + 1);

    // Measure time for recursion
    auto start = high_resolution_clock::now();
    fibRec(n);
    auto end = high_resolution_clock::now();
    double timeRec = duration_cast<nanoseconds>(end
- start).count() / 1e9;
```

```
    // Measure time for loop
    start = high_resolution_clock::now();
    fibLoop(n);
    end = high_resolution_clock::now();
    double timeLoop = duration_cast<nanoseconds>(end
- start).count() / 1e9;

    // Measure time for recursion with memoization
    start = high_resolution_clock::now();
    fibRecMemo(n, memo);
    end = high_resolution_clock::now();
    double timeRecMemo =
duration_cast<nanoseconds>(end - start).count() /
1e9;

    // Measure time for loop with memoization
    start = high_resolution_clock::now();
    fibLoopMemo(n, arr);
    end = high_resolution_clock::now();
    double timeLoopMemo =
duration_cast<nanoseconds>(end - start).count() /
1e9;

    // Calculate speedups
```

```
    double speedupLoop = timeRec / timeLoop;
    double speedupRecMemo = timeRec / timeRecMemo;
    double speedupLoopMemo = timeRec / timeLoopMemo;

    // Print results
    cout << "Time for recursion: " << timeRec << "
seconds" << endl;
    cout << "Time for loops: " << timeLoop << "
seconds, Speedup: " << speedupLoop << "x" << endl;
    cout << "Time for recursion with memo: " <<
timeRecMemo << " seconds, Speedup: " <<
speedupRecMemo << "x" << endl;
    cout << "Time for loops with memo: " <<
timeLoopMemo << " seconds, Speedup: " <<
speedupLoopMemo << "x" << endl;
}

int main() {
    int n = 50; // First 50 Fibonacci numbers
    measureTime(n);
    return 0;
}
```

Output:

Time for recursion: 67.5069 seconds
Time for loops: 1.25e-07 seconds, Speedup: 5.40055e+08x
Time for recursion with memo: 1.1458e-05 seconds, Speedup: 5.89168e+06x
Time for loops with memo: 2.91e-07 seconds, Speedup: 2.31982e+08x

2. Write a simple Matrix Multiplication program for a given NxN matrix in any two of your preferred Languages from the following listed buckets, where N is iterated through the set of values 64, 128, 256, 512 and 1024. N can either be hardcoded or specified as input.

Code:

```
#include <iostream>
#include <vector>
#include <ctime>
#include <sys/time.h>
#include <sys/resource.h>

void MatrixMultiplicationInt(int N) {
   std::vector<std::vector<int>> A(N,
std::vector<int>(N, 1));
   std::vector<std::vector<int>> B(N,
std::vector<int>(N, 1));
   std::vector<std::vector<int>> C(N,
std::vector<int>(N, 0));

   struct timeval start, end;
   gettimeofday(&start, NULL);

   for(int i = 0; i < N; ++i) {
```

```
        for(int j = 0; j < N; ++j) {
            for(int k = 0; k < N; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    gettimeofday(&end, NULL);
    double real_time_taken = (end.tv_sec -
start.tv_sec) + (end.tv_usec - start.tv_usec) /
1e6;

    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    double program_time_taken =
usage.ru_utime.tv_sec + usage.ru_utime.tv_usec /
1e6;
    double system_time_taken = usage.ru_stime.tv_sec
+ usage.ru_stime.tv_usec / 1e6;

    std::cout << "Time taken by integer
multiplication for N = " << N << ":" << std::endl;
    std::cout << "Real-Time: " << real_time_taken <<
" seconds" << std::endl;
```

```
    std::cout << "Program-Time: " <<
program_time_taken << " seconds" << std::endl;
    std::cout << "System-Time: " <<
system_time_taken << " seconds" << std::endl;
}

void MatrixMultiplicationDouble(int N) {
    std::vector<std::vector<double>> A(N,
std::vector<double>(N, 1.0));
    std::vector<std::vector<double>> B(N,
std::vector<double>(N, 1.0));
    std::vector<std::vector<double>> C(N,
std::vector<double>(N, 0.0));

    struct timeval start, end;
    gettimeofday(&start, NULL);

    for(int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            for(int k = 0; k < N; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

```
    gettimeofday(&end, NULL);
    double real_time_taken = (end.tv_sec -
start.tv_sec) + (end.tv_usec - start.tv_usec) /
1e6;

    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    double program_time_taken =
usage.ru_utime.tv_sec + usage.ru_utime.tv_usec /
1e6;
    double system_time_taken = usage.ru_stime.tv_sec
+ usage.ru_stime.tv_usec / 1e6;

    std::cout << "Time taken by double
multiplication for N = " << N << ":" << std::endl;
    std::cout << "Real-Time: " << real_time_taken <<
" seconds" << std::endl;
    std::cout << "Program-Time: " <<
program_time_taken << " seconds" << std::endl;
    std::cout << "System-Time: " <<
system_time_taken << " seconds" << std::endl;
}

int main() {
    for (int N : {64, 128, 256, 512, 1024}) {
```

```
        MatrixMultiplicationInt(N);
        MatrixMultiplicationDouble(N);
    }
    return 0;
}
```

Output For C++:
Time taken by integer multiplication for N = 64: 0.003983 seconds
Time taken by double multiplication for N = 64: 0.004931 seconds
Time taken by integer multiplication for N = 128: 0.034854 seconds
Time taken by double multiplication for N = 128: 0.033288 seconds
Time taken by integer multiplication for N = 256: 0.205534 seconds
Time taken by double multiplication for N = 256: 0.276216 seconds
Time taken by integer multiplication for N = 512: 1.71537 seconds
Time taken by double multiplication for N = 512: 2.12655 seconds
Time taken by integer multiplication for N = 1024: 15.3496 seconds
Time taken by double multiplication for N = 1024: 17.1271 seconds

Time taken by integer multiplication for N = 64:
Real-Time: 0.002969 seconds
Program-Time: 0.002484 seconds
System-Time: 0.002484 seconds
Time taken by double multiplication for N = 64:
Real-Time: 0.003911 seconds
Program-Time: 0.0063 seconds
System-Time: 0.0027 seconds
Time taken by integer multiplication for N = 128:
Real-Time: 0.07299 seconds
Program-Time: 0.031106 seconds
System-Time: 0.002916 seconds
Time taken by double multiplication for N = 128:
Real-Time: 0.028259 seconds
Program-Time: 0.059699 seconds
System-Time: 0.002936 seconds
Time taken by integer multiplication for N = 256:
Real-Time: 0.406829 seconds
Program-Time: 0.266853 seconds

System-Time: 0.002987 seconds
Time taken by double multiplication for N = 256:
Real-Time: 0.681178 seconds
Program-Time: 0.510505 seconds
System-Time: 0.003996 seconds
Time taken by integer multiplication for N = 512:
Real-Time: 3.98062 seconds
Program-Time: 2.37805 seconds
System-Time: 0.012935 seconds
Time taken by double multiplication for N = 512:
Real-Time: 5.4965 seconds
Program-Time: 4.63141 seconds
System-Time: 0.025807 seconds
Time taken by integer multiplication for N = 1024:
Real-Time: 45.3995 seconds
Program-Time: 20.5248 seconds
System-Time: 0.061707 seconds
Time taken by double multiplication for N = 1024:
Real-Time: 58.7928 seconds
Program-Time: 38.7994 seconds
System-Time: 0.122317 seconds

Python Code:

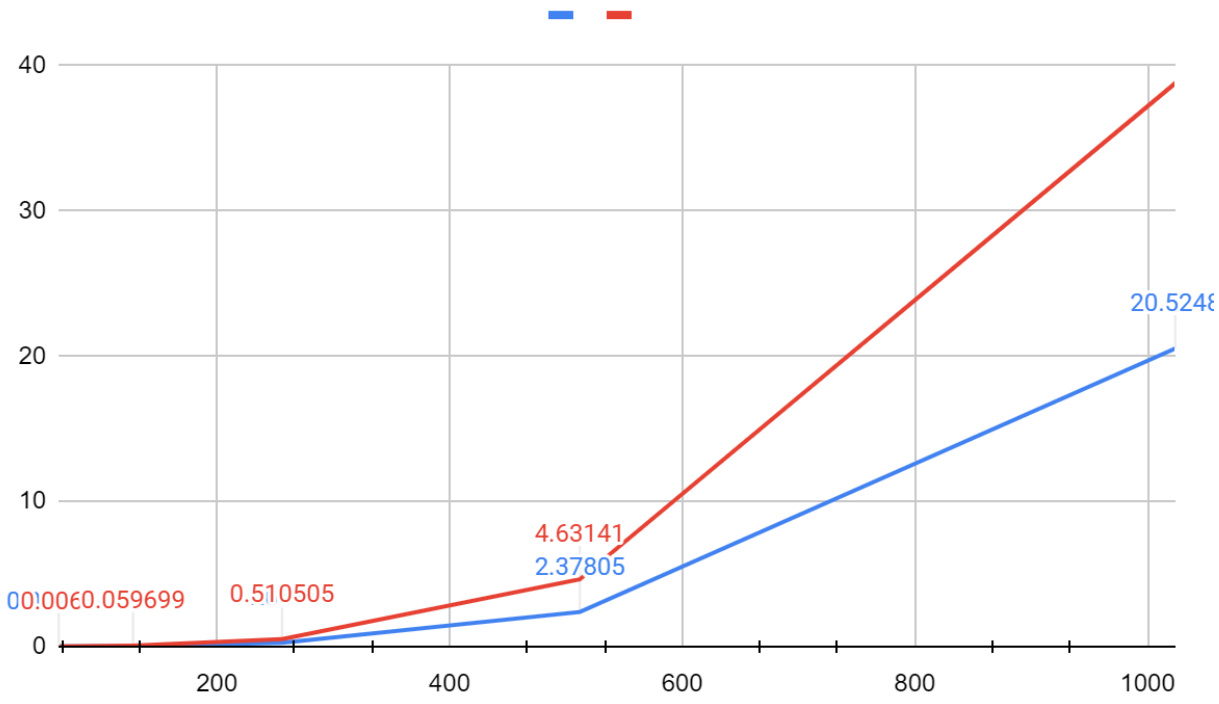
```
import time
import resource

def matrix_multiplication(N, dtype):
    A = [[1 if dtype == int else 1.0 for _ in range(N)] for _ in range(N)]
    B = [[1 if dtype == int else 1.0 for _ in range(N)] for _ in range(N)]
    C = [[0 if dtype == int else 0.0 for _ in range(N)] for _ in range(N)]

    start_real_time = time.time()
    start_program_time = time.process_time()
    start_system_time = resource.getrusage(resource.RUSAGE_SELF).ru_stime

    for i in range(N):
        for j in range(N):
            for k in range(N):
                C[i][j] += A[i][k] * B[k][j]
```

```
    end_real_time = time.time()
    end_program_time = time.process_time()
    end_system_time = resource.getrusage(resource.RUSAGE_SELF).ru_stime

    real_time_taken = round(end_real_time - start_real_time, 3)
    program_time_taken = round(end_program_time - start_program_time, 3)
    system_time_taken = round(end_system_time - start_system_time, 3)

    print(f"Time taken by {dtype.__name__} multiplication for N = {N}:")
    print(f"Real-Time: {real_time_taken} seconds")
    print(f"Program-Time: {program_time_taken} seconds")
    print(f"System-Time: {system_time_taken} seconds")

for N in [64, 128, 256, 512, 1024]:
    matrix_multiplication(N, int)
    matrix_multiplication(N, float)
```

Output:

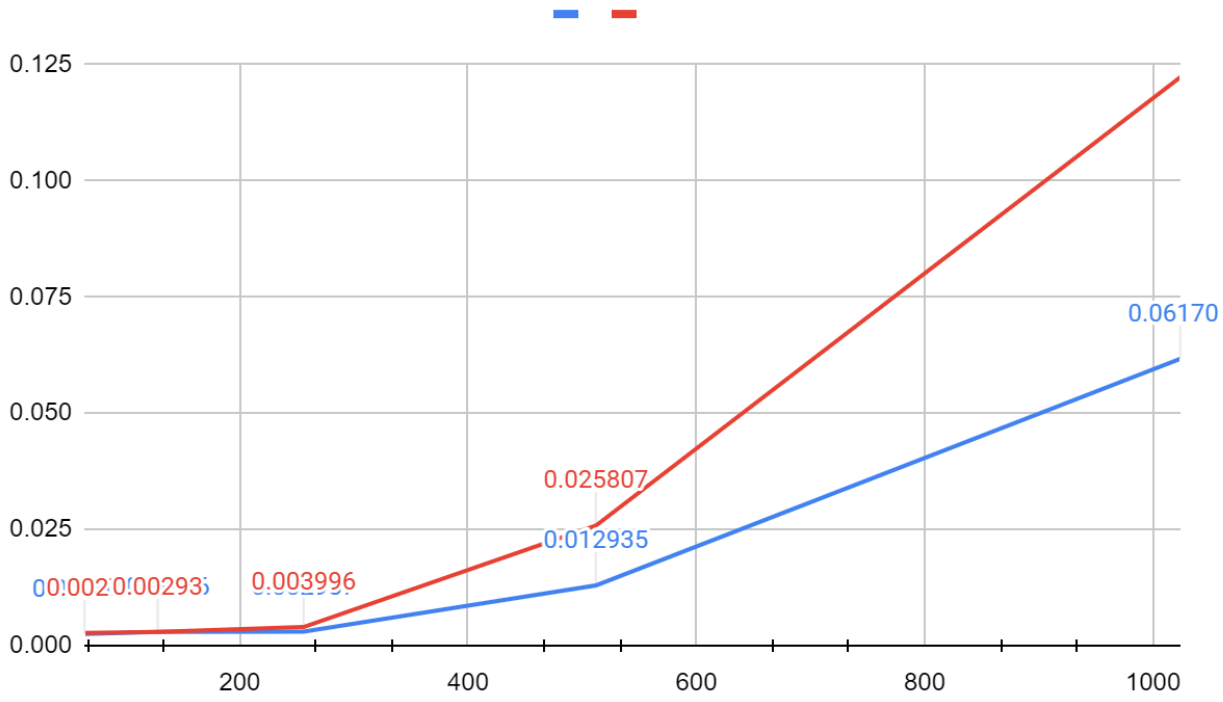
```
Time taken by int multiplication for N = 64:
Real-Time: 0.027 seconds
Program-Time: 0.027 seconds
System-Time: 0.0 seconds
Time taken by float multiplication for N = 64:
Real-Time: 0.03 seconds
Program-Time: 0.03 seconds
System-Time: 0.0 seconds
Time taken by int multiplication for N = 128:
Real-Time: 0.194 seconds
Program-Time: 0.194 seconds
System-Time: 0.0 seconds
Time taken by float multiplication for N = 128:
Real-Time: 0.204 seconds
Program-Time: 0.201 seconds
System-Time: 0.0 seconds
Time taken by int multiplication for N = 256:
Real-Time: 1.554 seconds
Program-Time: 1.493 seconds
System-Time: 0.0 seconds
Time taken by float multiplication for N = 256:
Real-Time: 1.548 seconds
Program-Time: 1.544 seconds
System-Time: 0.003 seconds
```

Time taken by int multiplication for N = 512:
Real-Time: 14.028 seconds
Program-Time: 13.706 seconds
System-Time: 0.019 seconds
Time taken by float multiplication for N = 512:
Real-Time: 14.584 seconds
Program-Time: 14.254 seconds
System-Time: 0.024 seconds
Time taken by int multiplication for N = 1024:
Real-Time: 127.01 seconds
Program-Time: 123.596 seconds
System-Time: 0.238 seconds
Time taken by float multiplication for N = 1024:
Real-Time: 129.471 seconds
Program-Time: 126.059 seconds
System-Time: 0.185 seconds



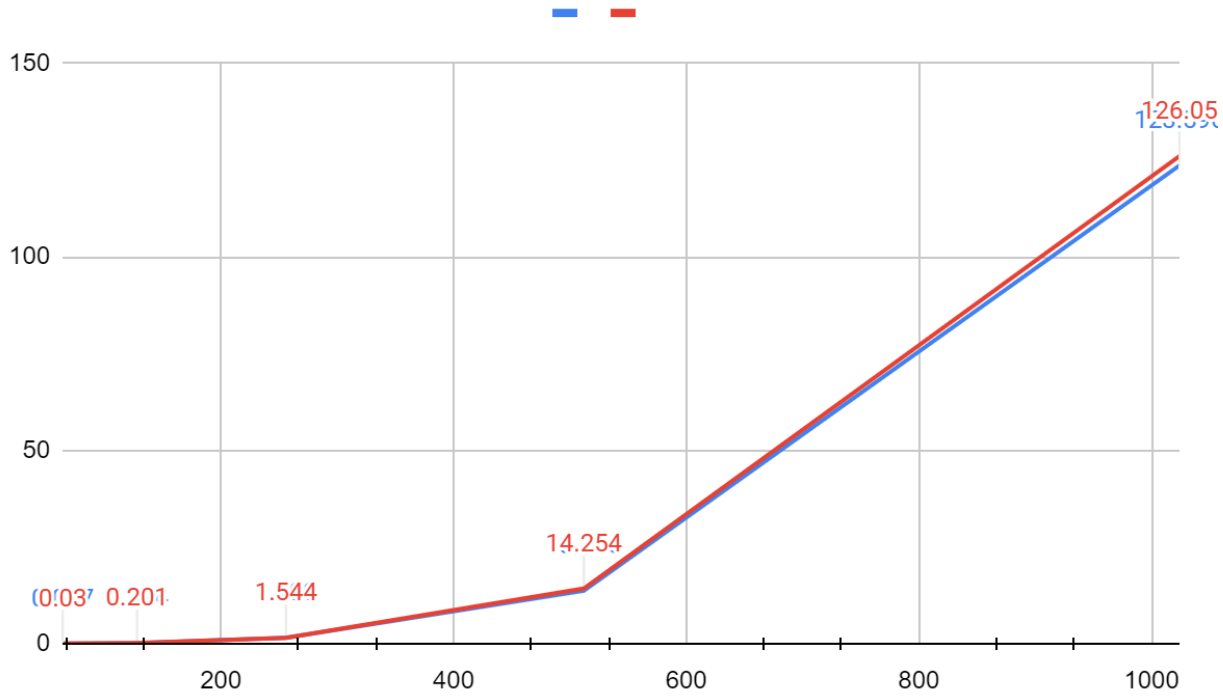
Graph 1. Graph between total execution time and value of N for C++

Where x axis represents value of N and y axis represents value of total execution time, where red line represents the double data structure and blue line represents the INT data structure.

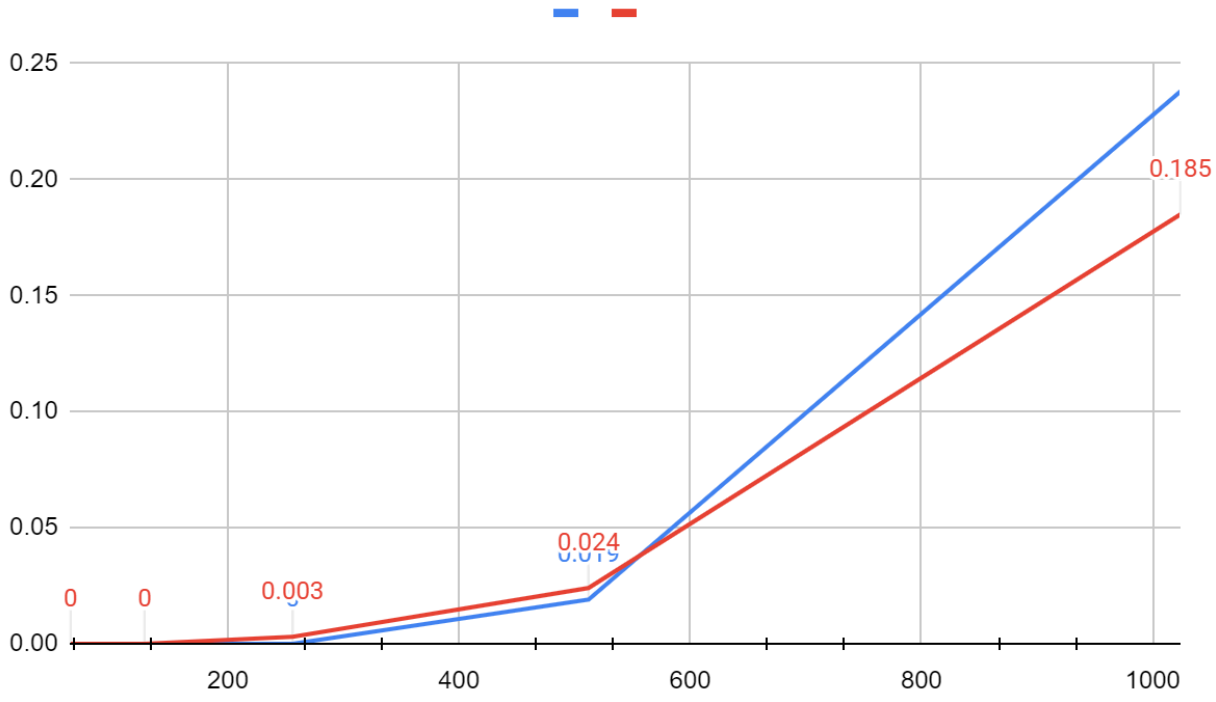


Graph 2. Graph between system time and value of N for C++

Where x axis represents value of N and y axis represents value of the system time where red line represents the double data structure and blue line represents the INT data structure.



Graph 3: Graph between total execution time and value of N for Python where red line represents the double data structure and blue line represents the INT data structure.



Graph 4: Graph between system time and value of N for Python

Where x axis represents value of N and y axis represents value of the system time where red line represents the double data structure and blue line represents the INT data structure.

Observations:

Performance Comparison: C++ vs Python

- C++ consistently outperforms Python in both integer and double-precision multiplication, regardless of matrix size.
- The performance gap between C++ and Python widens significantly as matrix size increases, with Python taking considerably longer for larger matrices.

Integer vs Double Multiplication

- Both C++ and Python take slightly longer for double multiplication compared to integer multiplication. However, this difference is more noticeable in Python.

Program Time

- Program execution time reflects the expected n3n^3n3 time complexity of the matrix multiplication algorithm, but the constant factor for Python is significantly larger than that for C++.

System Time

- System time remains relatively low for both C++ and Python, but Python exhibits slightly higher system time, particularly with larger matrices.