Assignment 1

Computer Organization and Architecture

Dewansh Singh Chandel 22110072

1. Implement a program(s) to list the first 50 Fibonacci numbers, preferably in C/C++, in the following manner: (Total: 50 points)
   a. Using recursion (10 points)
   b. Using loop (10 points)
   c. Using recursion and memoization (10 points)
   d. Using loop and memoization (10 points)
   Find the speedup of all the programs on your machine by keeping program (1) as the baseline. (10 points).

Solution:

First 50 Fibonacci numbers

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
6765 10946 17711 28657 46368 75025 121393 196418 317811 514229
832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817
39088169 63245986 102334155 165580141 267914296 433494437
701408733 1134903170 1836311903 2971215073 4807526976 7778742049
```

Code:

```
#include <iostream>
#include <ctime>
#include <vector>

using namespace std;


double time_diff(struct timespec start, struct timespec end) {
    return (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
}


long long fibonacciRecursion(int n) {
    if (n <= 1) return n;
```

```
    return fibonacciRecursion(n - 1) + fibonacciRecursion(n - 2);
}


void fibonacciLoop(int n, vector<long long>& fib_sequence) {
    fib_sequence[0] = 0;
    if (n > 1) fib_sequence[1] = 1;
    for (int i = 2; i < n; i++) {
        fib_sequence[i] = fib_sequence[i - 1] + fib_sequence[i - 2];
    }
}


vector<long long> memo(50, -1);
long long fibonacciRecursionMemo(int n) {
    if (n <= 1) return n;
    if (memo[n] != -1) return memo[n];
    return memo[n] = fibonacciRecursionMemo(n - 1) + fibonacciRecursionMemo(n
}


void fibonacciLoopMemo(int n, vector<long long>& fib_sequence) {
    fib_sequence[0] = 0;
    if (n > 1) fib_sequence[1] = 1;
    for (int i = 2; i < n; i++) {
        fib_sequence[i] = fib_sequence[i - 1] + fib_sequence[i - 2];
    }
}

int main() {
    struct timespec start, end;
    double time_taken;


    clock_gettime(CLOCK_MONOTONIC, &start);
    for (int i = 0; i < 50; i++) {
        cout <<fibonacciRecursion(i)<<" ";
    }

    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "\nTime taken by Recursion: " << time_taken << " seconds\n";
    double baseline_time = time_taken;
```

```
    vector<long long> fib_sequence(50);
    clock_gettime(CLOCK_MONOTONIC, &start);
    fibonacciLoop(50, fib_sequence);
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "Time taken by Loop: " << time_taken << " seconds\n";
    cout << "Speedup (Loop vs Recursion): " << baseline_time / time_taken <<


    memo.assign(50, -1);
    clock_gettime(CLOCK_MONOTONIC, &start);
    for (int i = 0; i < 50; i++) {
        cout << fibonacciRecursionMemo(i) << " ";
    }

    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "\nTime taken by Recursion with Memoization: " << time_taken
    << " seconds\n";
    cout << "Speedup (Recursion with Memoization vs Recursion): "
    << baseline_time / time_taken << "x\n";


    clock_gettime(CLOCK_MONOTONIC, &start);
    fibonacciLoopMemo(50, fib_sequence);
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "Time taken by Loop with Memoization: " << time_taken
     << " seconds\n";
    cout << "Speedup (Loop with Memoization vs Recursion): " <<
    baseline_time / time_taken << "x\n";

    return 0;
}
```

Time taken by programs

a. Using recursion - 133.447 seconds
b. Using loop - 5e-07 seconds
c. Using recursion and memoization - 0.0017616 seconds
d. Using loop and memoization - 6e-07 seconds seconds

Speed up is calculate as:

Speedup = T(BaseLine)/ T(Program)

Speed Ups :

Using loop	Using recursion and memoization	Using loop and memoization
2.66894e+08	75753.4	2.22412e+08

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. Consider two cases (a) Elements of matrix are of data type Integer and (b) Double In each case, (i.e. Bucket 1 for (a) and (b) + Bucket 2 for (a) and(b)) (Total: 100 points)
   Bucket1: C, C++, Go
   Bucket2: Python, Java.
   a. Report the output of the 'time' describing the system and CPU times. (25 points)
   b. Using the 'language hooks' evaluate the execution time for the meat portions of the program and how much proportion is it w.r.t. total program execution time. (25 points)
   c. Plot the (a) and (b) execution times for each of the iterations. And compare the performance (System and Program execution times) of the program for given value of N for the languages in both the buckets. –Illustrate your observations. (50 points)

Solution:

Note: Time proportion is calculated as Time taken by the Program/Real Time

Note: In the code below the matrix size (N) is taken as input and the time of program is calculated as the total time of matrix multiplication for that specific input.

Bucket 1: Language used C++

a. For Integer Data Type

Code:

```
void multiplyMatrixInteger(int N) {
    vector<vector<int>> A(N, vector<int>(N, 1));
    vector<vector<int>> B(N, vector<int>(N, 2));
    vector<vector<int>> C(N, vector<int>(N, 0));

    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];
            }
        }
}
```

```
int main() {
    struct timespec start, end;
    double time_taken;
        int N ;
        cin>>N;



    clock_gettime(CLOCK_MONOTONIC, &start);
    multiplyMatrixInteger(N);
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "Time taken for Integer Matrix Multiplication (N = " << N << "
    << time_taken << " seconds\n";



    return 0;
}
```

N=64

```
Time taken for Integer Matrix Multiplication (N = 64): 0.00137217 seconds

real    0m2.551s
user    0m0.002s
sys     0m0.000s
```

Proportion = (0.00137 seconds /2.551 seconds) = 0.000537

N=128

```
Time taken for Integer Matrix Multiplication (N = 128): 0.0116224 seconds

real    0m3.498s
user    0m0.007s
sys     0m0.007s
```

N= 256

```
Time taken for Integer Matrix Multiplication (N = 256): 0.0869183 seconds

real    0m2.912s
user    0m0.089s
sys     0m0.000s
```

N= 512

```
Time taken for Integer Matrix Multiplication (N = 512): 0.665133 seconds

real    0m6.686s
user    0m0.666s
sys     0m0.000s
```

N= 1024

```
Time taken for Integer Matrix Multiplication (N = 1024): 6.59933 seconds

real    0m9.663s
user    0m6.590s
sys     0m0.010s
```

b. For double data type

Code:

```
void multiplyMatrixDouble(int N) {
    vector<vector<double>> A(N, vector<double>(N, 1.0));
    vector<vector<double>> B(N, vector<double>(N, 2.0));
    vector<vector<double>> C(N, vector<double>(N, 0.0));

    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];
            }
        }
    }
}

int main() {
```

```
    struct timespec start, end;
    double time_taken;
    int N ;
    cin>>N;



    clock_gettime(CLOCK_MONOTONIC, &start);
    multiplyMatrixDouble(N);
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_taken = time_diff(start, end);
    cout << "Time taken for Double Matrix Multiplication (N = " << N << "): "
    << time_taken << " seconds\n";



    return 0;
}
```

N=64

```
Time taken for Double Matrix Multiplication (N = 64): 0.00156874 seconds

real    0m1.971s
user    0m0.001s
sys     0m0.000s
```

N=128

```
Time taken for Double Matrix Multiplication (N = 128): 0.0111214 seconds

real    0m3.101s
user    0m0.011s
sys     0m0.000s
```

N=256

```
Time taken for Double Matrix Multiplication (N = 256): 0.0868746 seconds

real    0m2.979s
user    0m0.088s
sys     0m0.000s
```

N=512

```
Time taken for Double Matrix Multiplication (N = 512): 0.699298 seconds

real    0m5.181s
user    0m0.701s
sys     0m0.000s
```

N=1024

```
Time taken for Double Matrix Multiplication (N = 1024): 10.5702 seconds

real    0m13.643s
user    0m10.561s
sys     0m0.010s
```

Bucket 2: Language used Python

a. Data type Integer

Code:

```
import time
import numpy as np

def multiply_matrix_integer(N):
    A = np.ones((N, N), dtype=int)
    B = np.full((N, N), 2, dtype=int)
    C = np.zeros((N, N), dtype=int)

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


N = int(input())

start_time = time.time()
multiply_matrix_integer(N)

print(f"Time taken for Integer Matrix Multiplication (N = {N}):
{time.time() - start_time:.6f} seconds")
```

N=64

```
Time taken for Integer Matrix Multiplication (N = 64): 0.107664 seconds

real    0m2.414s
user    0m1.169s
sys     0m0.047s
```

N=128

```
Time taken for Integer Matrix Multiplication (N = 128): 0.789819 seconds

real    0m3.037s
user    0m1.907s
sys     0m0.092s
```

N=256

```
Time taken for Integer Matrix Multiplication (N = 256): 6.269397 seconds

real    0m9.579s
user    0m7.411s
sys     0m0.139s
```

N=512

```
Time taken for Integer Matrix Multiplication (N = 512): 76.796486 seconds

real    1m19.363s
user    1m17.934s
sys     0m0.110s
```

N= 1024

```
Time taken for Integer Matrix Multiplication (N = 1024): 710.851472 second

real    11m54.169s
user    11m51.951s
sys     0m0.170s
```

b. Data type Double

Code:

```
import time
import numpy as np

def multiply_matrix_double(N):
    A = np.ones((N, N), dtype=float)
    B = np.full((N, N), 2.0, dtype=float)
    C = np.zeros((N, N), dtype=float)

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

N = int(input())

start_time = time.time()
multiply_matrix_double(N)
print(f"Time taken for Double Matrix Multiplication (N = {N}):

{time.time() - start_time:.6f} seconds")
```

N = 64

```
Time taken for Double Matrix Multiplication (N = 64): 0.165979 seconds

real    0m4.417s
user    0m1.394s
sys     0m0.083s
```

N =128

```
Time taken for Double Matrix Multiplication (N = 128): 1.533681 seconds

real    0m4.435s
user    0m2.729s
sys     0m0.060s
```

N =256

```
Time taken for Double Matrix Multiplication (N = 256): 10.940854 seconds

real    0m13.762s
user    0m12.147s
sys     0m0.111s
```

N= 512

```
Time taken for Double Matrix Multiplication (N = 512): 88.248821 seconds

real    1m30.726s
user    1m29.538s
sys     0m0.050s
```

N = 1024

```
Time taken for Double Matrix Multiplication (N = 1024): 765.818712 seconds

real    12m49.879s
user    12m46.996s
sys     0m0.151s
```

C++ (Integer Data Type)

N	Time (seconds)	Real Time (seconds)	Proportion
64	0.00137217	2.551	0.000537
128	0.0116224	3.498	0.003322
256	0.0869183	2.912	0.029852
512	0.665133	6.686	0.099484

1024	6.59933	9.663	0.682922

C++ (Double Data Type)

N	Time (seconds)	Real Time (seconds)	Proportion
64	0.00156874	1.971	0.000796
128	0.0111214	3.101	0.003588
256	0.0868746	2.979	0.029153
512	0.699298	5.181	0.134974
1024	10.5702	13.643	0.774835

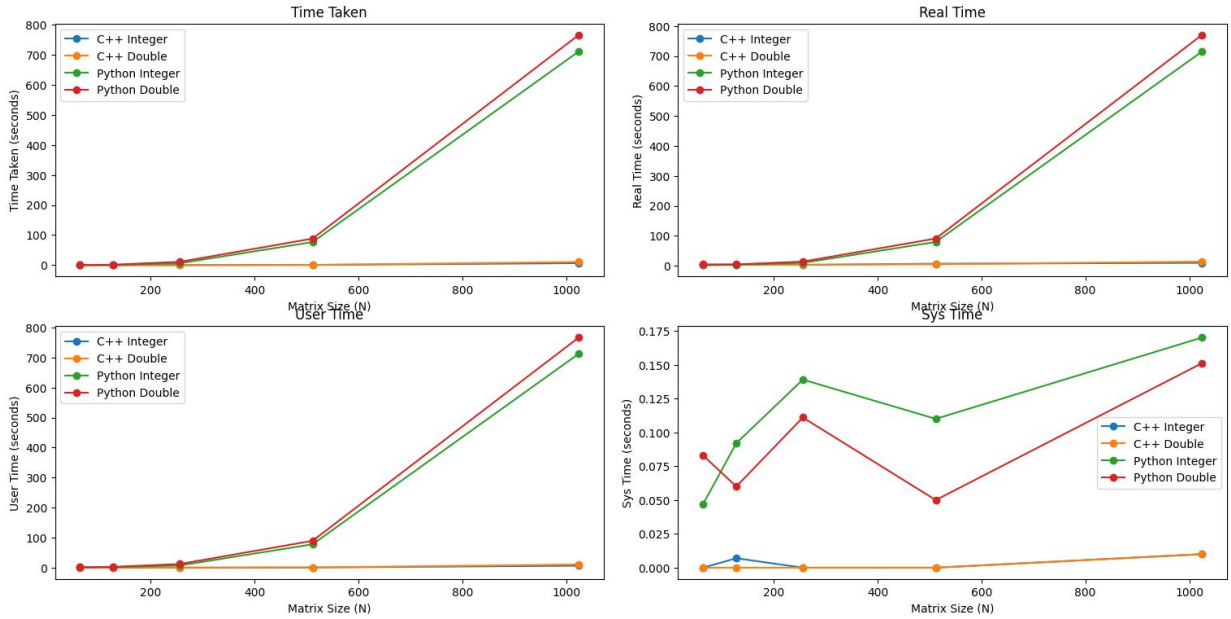
Python (Integer Data Type)

N	Time (seconds)	Real Time (seconds)	Proportion
64	0.107664	2.414	0.044590
128	0.789819	3.037	0.259976
256	6.269397	9.579	0.654462
512	76.796486	79.363	0.967670
1024	710.851472	714.169	0.995352

Python (Double Data Type)

N	Time (seconds)	Real Time (seconds)	Proportion
64	0.165979	4.417	0.037588
128	1.533681	4.435	0.345796
256	10.940854	13.762	0.794996
512	88.248821	90.726	0.972617
1024	765.818712	769.879	0.994716

Plots :



Plots of Matrix Size vs Time for all the time categories (Time taken by program, Real Time, User time, System Time)

Observations:

1. As value of the N increase the time required for computation also increases.
2. C++ programs generally have significantly lower time taken for matrix multiplication compared to Python.
3. The difference in time taken between integer and double data types is less pronounced in C++ compared to Python. In Python, the time taken with doubles is significantly higher than with integers.