**Computer Organization and Architecture**Assignment 1  
Mayank Gulati - 22110146

Date: 16th August, 2024

**Machine specifications:**CPU: Apple M3 Pro 11-core  
Memory: 18 GB

**Question 1**

In this question, the objective is to implement the Fibonacci sequence in different scenarios and observe the difference in time taken for execution of the programs.

It has been assumed that the monotonic clock must be used for measuring the execution time.

Since output of all the four programs is the same, it has been stated once here and not separately for each approach.  
  
Output: 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  
  
(a). Fibonacci sequence using recursion | Source Code:

Recursion is not the best approach for obtaining the Fibonacci sequence, and the same is evident in the execution time for the recursive program.

Time taken by recursion = seconds.

(b). Fibonacci sequence using loop | Source Code:

An iterative approach is better than a recursive approach for a computation as large as calculating the Fibonacci sequence up to 50 positions.

Time taken by loop = seconds.

The below stated parts of the question require to implement memorization. Memoization greatly helps in reducing the time taken by a program by storing the previously computed results. This helps us to calculate the next number in the Fibonacci sequence with ease. However, this has an impact on the memory being used by the program when storing the computed results.

(c). Fibonacci sequence using memoized recursion | Source Code:

Using memorization when recursively computing the Fibonacci sequence has such a big impact on the execution time, that now the program is even faster than using the iterative approach.

Time taken by memoized recursion = seconds.

(d). Fibonacci sequence using memoized loop | Source Code:

It was observed that the iterative approach took way lesser time than the recursive approach. However, in the case of memoization, the memoized recursion took less time than the memoized loop, although the difference is negligible.

Time taken by memoized loop = seconds.

To calculate the speedup of each program considering program (a) as the base, we need to divide execution time of program (a) by the execution time of the specific program. Since the speedup is a ratio of two time durations, it does not have any units.

Speedup of program (a) = = 1

Speedup of program (b) = = =

Speedup of program (c) = = =

Speedup of program (d) = = =

As it can be observed, the maximum value of speedup is obtained in the case of program (c) which is the memoized recursion approach. This means that program (c) is the fastest in this case.

**Question 2**

In this question, the objective is to implement a matrix multiplication program for an N x N matrix, where N can vary between the values: 64, 128, 256, 512, 1024. Two different cases also need to be considered, for when elements of the matrices are integers and doubles.

From this problem, it will be clear that the efficiency of languages in both the buckets varies, and it becomes more evident as the size of N increases.

Bucket 1: C++ | Source Code  
Bucket 2: Python | Source Code

It has been assumed that, in this case, the time taken for multiplying two numbers does not depend on the numbers themselves. So, time taken for computing 100 x 100 will be the same as the time taken for computing 0 x 0. Due to this assumption, both the matrices have been initialized with all elements as zero.

(a). Using the ‘**time**’ command from the terminal, we can calculate the system and CPU times of the programs. The user time is the time taken by the CPU for the actual computation (matrix multiplication) or execution of the program. The system time represents the time take by the CPU for system-level operations such as memory allocation.

Command: **time ./matrix**Output: **./matrix 11.89s user 0.01s system 99% cpu 11.928 total**

From this, we can observe that in the case of C++:

CPU time = 11.89 seconds.  
System time = 0.01 seconds.

Command: **time python3 matrix.py**Output: **python3 matrix.py 116.16s user 0.12s system 99% cpu 1:56.63 total**

From this, we can observe that in the case of Python:

CPU time = 116.16 seconds.  
System time = 0.12 seconds.

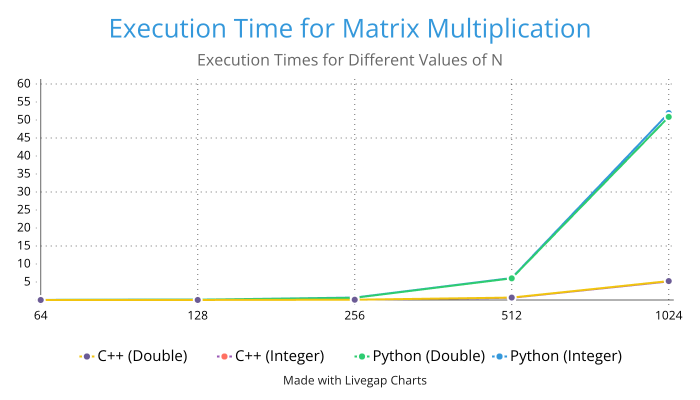
(b). In the source code mentioned above, the time taken for each value of N in both cases when the elements of the matrices are integers and doubles has been calculated, and the same is tabulated below.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **C++** | | | | **Python** | | | |
| **Integer** | | **Double** | | **Integer** | | **Float (Double)** | |
| **Time (seconds)** | **Proportion (%)\*** | **Time (seconds)** | **Proportion (%)\*** | **Time (seconds)** | **Proportion (%)\*** | **Time (seconds)** | **Proportion (%)\*** |
| 64 |  | 0.011 % |  | 0.010 % |  | 0.008 % |  | 0.008 % |
| 128 |  | 0.081 % |  | 0.081 % |  | 0.074 % |  | 0.065 % |
| 256 |  | 0.656 % |  | 0.653 % | 0.678 | 0.584 % | 0.612 | 0.527 % |
| 512 | 0.616 | 5.180 % | 0.673 | 5.660 % | 6.112 | 5.262 % | 6.017 | 5.180 % |
| 1024 | 5.151 | 43.322 % | 5.256 | 44.205 % | 51.927 | 44.703 % | 50.858 | 43.783 % |

TABLE 1: Execution times for each N for integer and double datatypes  
\*The proportions has been calculated as a percentage of the CPU time for each language.

As mentioned before, the difference in execution time of both the languages is evident, and the gap increases as the value of N increases. For N = 1024, the difference is significant, and can hamper user experience and power efficiency in a real-life application.

(c). The execution times for different values for both cases (integer and double) are plotted below.

  
Y-axis: seconds, X-axis: N

We can see that the execution time increases as N increases, and the biggest jump is observed in case of Python when N goes from 512 to 1024, giving us maximum slope in the graph.

Even though the execution times in the two languages vary, especially for higher values of N, the relative percentage proportions have similar values.

It can be concluded that although Python code is user friendly, it is in no way efficient when it comes to large computations. C++ is much faster, even for higher values of N in this case of matrix multiplication.