ES215: Assignment 1

Name: Prey Patel
Roll No: 20110132

Q1.

The programs are uploaded on this [github repo.]

a) The recursive approach to generate Fibonacci numbers involves defining a function that calls itself to calculate the Fibonacci sequence. Program using recursion will take a very long time to calculate the higher fibonacci numbers as it needs to calculate each fibonacci number from beginning using recursion.

Time taken also follows approximately a fibonacci sequence and thus calculating with recursion will become impossible.

```
fib(17) = 1597    Time taken: 0 ms
fib(18) = 2584    Time taken: 0 ms
fib(19) = 4181    Time taken: 0 ms
fib(20) = 6765    Time taken: 0 ms
fib(21) = 10946           Time taken: 0 ms
fib(22) = 17711           Time taken: 0 ms
fib(23) = 28657           Time taken: 0 ms
fib(24) = 46368           Time taken: 1 ms
fib(25) = 75025           Time taken: 2 ms
fib(26) = 121393          Time taken: 4 ms
fib(27) = 196418          Time taken: 7 ms
fib(28) = 317811          Time taken: 10 ms
fib(29) = 514229          Time taken: 9 ms
fib(30) = 832040          Time taken: 10 ms
fib(31) = 1.34627e+06     Time taken: 12 ms
fib(32) = 2.17831e+06     Time taken: 20 ms
fib(33) = 3.52458e+06     Time taken: 30 ms
fib(34) = 5.70289e+06     Time taken: 47 ms
fib(35) = 9.22746e+06     Time taken: 73 ms
fib(36) = 1.49304e+07     Time taken: 110 ms
fib(37) = 2.41578e+07     Time taken: 164 ms
fib(38) = 3.90882e+07     Time taken: 292 ms
fib(39) = 6.3246e+07      Time taken: 502 ms
fib(40) = 1.02334e+08     Time taken: 687 ms
fib(41) = 1.6558e+08      Time taken: 1108 ms
fib(42) = 2.67914e+08     Time taken: 1942 ms
fib(43) = 4.33494e+08     Time taken: 3447 ms
fib(44) = 7.01409e+08     Time taken: 4685 ms
fib(45) = 1.1349e+09      Time taken: 7526 ms
fib(46) = 1.83631e+09     Time taken: 12203 ms
fib(47) = 2.97122e+09     Time taken: 19697 ms
```

b)

```
fib(76) = 3.41645e+15    Time taken: 2 micro seconds
fib(77) = 5.52794e+15    Time taken: 2 micro seconds
fib(78) = 8.94439e+15    Time taken: 3 micro seconds
fib(79) = 1.44723e+16    Time taken: 2 micro seconds
fib(80) = 2.34167e+16    Time taken: 2 micro seconds
fib(81) = 3.78891e+16    Time taken: 3 micro seconds
fib(82) = 6.13058e+16    Time taken: 3 micro seconds
fib(83) = 9.91949e+16    Time taken: 3 micro seconds
fib(84) = 1.60501e+17    Time taken: 3 micro seconds
fib(85) = 2.59695e+17    Time taken: 3 micro seconds
fib(86) = 4.20196e+17    Time taken: 3 micro seconds
fib(87) = 6.79892e+17    Time taken: 3 micro seconds
fib(88) = 1.10009e+18    Time taken: 3 micro seconds
fib(89) = 1.77998e+18    Time taken: 2 micro seconds
fib(90) = 2.88007e+18    Time taken: 3 micro seconds
fib(91) = 4.66005e+18    Time taken: 3 micro seconds
fib(92) = 7.54011e+18    Time taken: 3 micro seconds
fib(93) = 1.22002e+19    Time taken: 3 micro seconds
fib(94) = 1.97403e+19    Time taken: 3 micro seconds
fib(95) = 3.19404e+19    Time taken: 3 micro seconds
fib(96) = 5.16807e+19    Time taken: 3 micro seconds
fib(97) = 8.36211e+19    Time taken: 3 micro seconds
fib(98) = 1.35302e+20    Time taken: 3 micro seconds
fib(99) = 2.18923e+20    Time taken: 3 micro seconds
total time: 1419 micro seconds
```

The iterative approach uses a loop to calculate Fibonacci numbers without recursion. It initializes variables to represent the first two numbers in the sequence and iterates through a loop to compute subsequent numbers by summing the previous two.

This approach is way better than recursion as it will need to calculate lower fibonacci numbers just once.

c)

Memoization optimizes the recursive approach by storing previously computed Fibonacci numbers array. Before computing a Fibonacci number, the function checks if it has already been calculated and retrieves it from the memoization table if available.

```
fib(77) = 5.52794e+15    Time taken: 1 micro seconds
fib(78) = 8.94439e+15    Time taken: 1 micro seconds
fib(79) = 1.44723e+16    Time taken: 1 micro seconds
fib(80) = 2.34167e+16    Time taken: 1 micro seconds
fib(81) = 3.78891e+16    Time taken: 1 micro seconds
fib(82) = 6.13058e+16    Time taken: 1 micro seconds
fib(83) = 9.91949e+16    Time taken: 1 micro seconds
fib(84) = 1.60501e+17    Time taken: 1 micro seconds
fib(85) = 2.59695e+17    Time taken: 1 micro seconds
fib(86) = 4.20196e+17    Time taken: 1 micro seconds
fib(87) = 6.79892e+17    Time taken: 1 micro seconds
fib(88) = 1.10009e+18    Time taken: 1 micro seconds
fib(89) = 1.77998e+18    Time taken: 1 micro seconds
fib(90) = 2.88007e+18    Time taken: 1 micro seconds
fib(91) = 4.66005e+18    Time taken: 1 micro seconds
fib(92) = 7.54011e+18    Time taken: 1 micro seconds
fib(93) = 1.22002e+19    Time taken: 1 micro seconds
fib(94) = 1.97403e+19    Time taken: 1 micro seconds
fib(95) = 3.19404e+19    Time taken: 1 micro seconds
fib(96) = 5.16807e+19    Time taken: 1 micro seconds
fib(97) = 8.36211e+19    Time taken: 1 micro seconds
fib(98) = 1.35302e+20    Time taken: 1 micro seconds
fib(99) = 2.18923e+20    Time taken: 1 micro seconds
total time: 1002 micro seconds
```

d)

Similarly also for loop with memoization will remember previously calculated fibonacci numbers and if available it will not need to be calculated from start.

```
fib(77) = 5.52794e+15    Time taken: 1 micro seconds
fib(78) = 8.94439e+15    Time taken: 0 micro seconds
fib(79) = 1.44723e+16    Time taken: 0 micro seconds
fib(80) = 2.34167e+16    Time taken: 0 micro seconds
fib(81) = 3.78891e+16    Time taken: 0 micro seconds
fib(82) = 6.13058e+16    Time taken: 1 micro seconds
fib(83) = 9.91949e+16    Time taken: 0 micro seconds
fib(84) = 1.60501e+17    Time taken: 0 micro seconds
fib(85) = 2.59695e+17    Time taken: 0 micro seconds
fib(86) = 4.20196e+17    Time taken: 1 micro seconds
fib(87) = 6.79892e+17    Time taken: 0 micro seconds
fib(88) = 1.10009e+18    Time taken: 0 micro seconds
fib(89) = 1.77998e+18    Time taken: 0 micro seconds
fib(90) = 2.88007e+18    Time taken: 1 micro seconds
fib(91) = 4.66005e+18    Time taken: 0 micro seconds
fib(92) = 7.54011e+18    Time taken: 0 micro seconds
fib(93) = 1.22002e+19    Time taken: 1 micro seconds
fib(94) = 1.97403e+19    Time taken: 1 micro seconds
fib(95) = 3.19404e+19    Time taken: 0 micro seconds
fib(96) = 5.16807e+19    Time taken: 1 micro seconds
fib(97) = 8.36211e+19    Time taken: 1 micro seconds
fib(98) = 1.35302e+20    Time taken: 0 micro seconds
fib(99) = 2.18923e+20    Time taken: 0 micro seconds
total time: 347 micro seconds
```

e)

The base program has a time complexity of $O(2^n)$.
The second program has a time complexity of O(n).
The third program has a time complexity of O(n).
The forth program has a time complexity of O(n). But by using memoization this becomes even faster then second once numbers start to get filled in memo_table. The forth program is almost 4 times faster than the second.
The third programme is also 1.5 times faster than the second program.
And all later three programs are approximately 2^{92} times faster than the first program.

Q2.

a)

C language programs:

```
└─$ time ./2_1
Matrix multiplication 512 x 512 of Double type.
CPU Time used: 0.790934

real    0.81s
user    0.81s
sys     0.00s
cpu     99%
```

```
└─$ time ./2_2
Matrix multiplication 512 x 512 of Int type.
CPU Time used: 0.846726

real    0.88s
user    0.87s
sys     0.01s
cpu     99%
```

Python language programs:

```
└─$ time python 2_1.py
Matrix multiplication 512 x 512 of float type.
CPU Time: 51.81101322174072
Process Time: 52.954429463


real    52.99s
user    52.96s
sys     0.02s
cpu     99%
```

```
└─$ time python 2_2.py
Matrix multiplication 512 x 512 of int type.
CPU Time: 50.017064571380615
Process Time: 51.183273423


real    51.21s
user    51.17s
sys     0.04s
cpu     99%
```

When running the programs using the time command, the output indicated that the total time was largely composed of CPU time, with a minimal portion as system time. The time taken by python language is more than C language, the CPU time accounted for approximately 99% of the total time (real), while system-level operations (sys) constituted around 1%.

b)

```
└─$ gcc -pg -o 2_1 2_1.c

┌──(kali㉿kali)-[~/COA/COA]
└─$ ./2_1
Matrix multiplication 512 x 512 of Double type.
CPU Time used: 1.759964

┌──(kali㉿kali)-[~/COA/COA]
└─$ gprof 2_1 gmon.out > analysis.txt
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 69.80      0.46     0.46        1   460.68   630.93  matrixMultiplication
 25.80      0.63     0.17 134217728     0.00     0.00  multiply
  4.55      0.66     0.03        2    15.02    15.02  initializeRandomMatrix
```

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.66                 main [1]
                0.46    0.17       1/1           matrixMultiplication [2]
                0.03    0.00       2/2           initializeRandomMatrix [4]
-----------------------------------------------
                0.46    0.17       1/1           main [1]
[2]     95.5    0.46    0.17       1         matrixMultiplication [2]
                0.17    0.00 134217728/134217728     multiply [3]
-----------------------------------------------
                0.17    0.00 134217728/134217728     matrixMultiplication [2]
[3]     25.8    0.17    0.00 134217728         multiply [3]
-----------------------------------------------
                0.03    0.00       2/2           main [1]
[4]      4.5    0.03    0.00       2         initializeRandomMatrix [4]
-----------------------------------------------
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 78.45      0.47     0.47        1   470.69   590.87  matrixMultiplication
 20.03      0.59     0.12 134217728     0.00     0.00  multiply
  1.67      0.60     0.01        2     5.01     5.01  initializeRandomMatrix
```

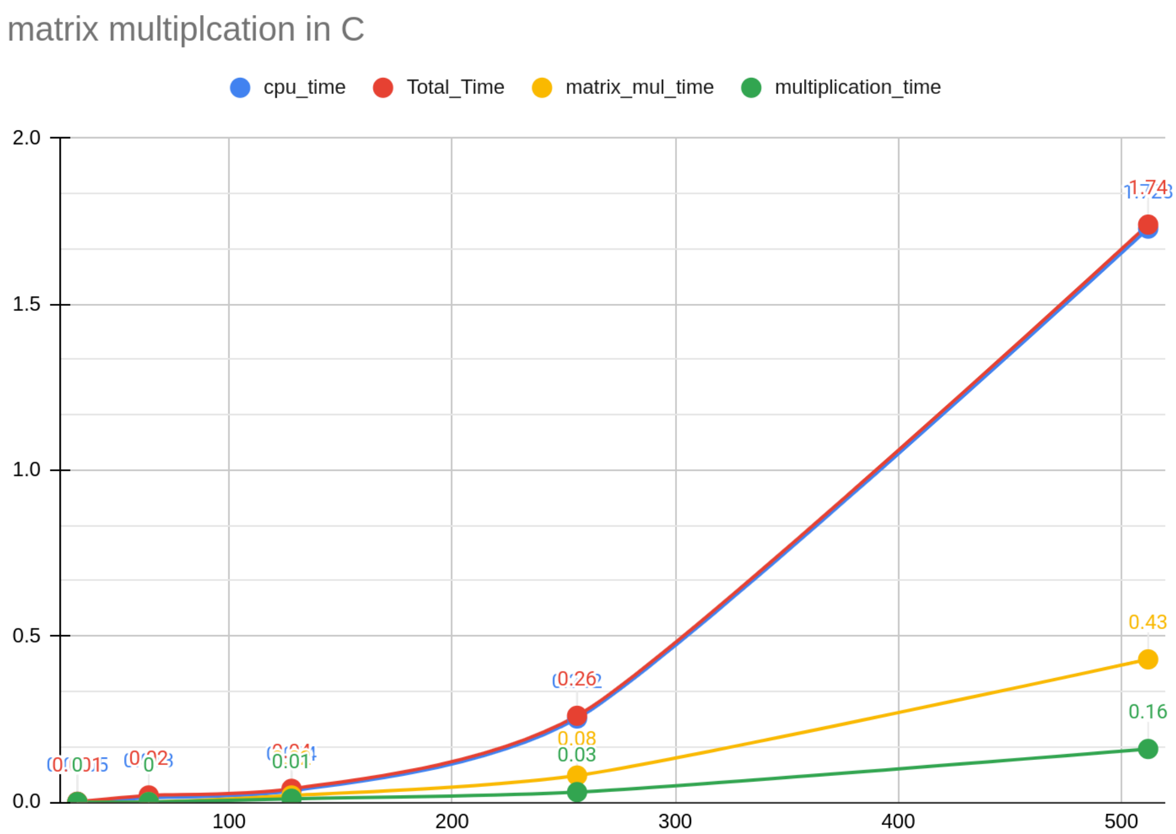
```
granularity: each sample hit covers 2 byte(s) for 1.66% of 0.60 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.60                 main [1]
                0.47    0.12       1/1           matrixMultiplication [2]
                0.01    0.00       2/2           initializeRandomMatrix [4]
-----------------------------------------------
                0.47    0.12       1/1           main [1]
[2]     98.3    0.47    0.12       1         matrixMultiplication [2]
                0.12    0.00 134217728/134217728     multiply [3]
-----------------------------------------------
                0.12    0.00 134217728/134217728     matrixMultiplication [2]
[3]     20.0    0.12    0.00 134217728         multiply [3]
-----------------------------------------------
                0.01    0.00       2/2           main [1]
[4]      1.7    0.01    0.00       2         initializeRandomMatrix [4]
-----------------------------------------------
```

```
└─$ python 2_1.py
Matrix multiplication 512 x 512 of float type.
CPU Time: 48.84976601600647
Process Time: 50.004731773

         138935891 function calls in 49.997 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1   38.891   38.891   48.850   48.850 2_1.py:12(matrixMultiplication)
        1    0.003    0.003    0.003    0.003 2_1.py:15(<listcomp>)
        2    0.154    0.077    1.143    0.572 2_1.py:26(random_init)
        1    0.001    0.001    0.001    0.001 2_1.py:31(<listcomp>)
        1    0.003    0.003    0.003    0.003 2_1.py:32(<listcomp>)
134217728    9.956    0.000    9.956    0.000 2_1.py:9(multiply)
   524288    0.274    0.000    0.377    0.000 random.py:235(_randbelow_with_getrandbits)
   524288    0.372    0.000    0.828    0.000 random.py:284(randrange)
   524288    0.162    0.000    0.989    0.000 random.py:358(randint)
  1572864    0.079    0.000    0.079    0.000 {built-in method _operator.index}
     1026    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {built-in method time.process_time}
        4    0.000    0.000    0.000    0.000 {built-in method time.time}
   524288    0.027    0.000    0.027    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
  1047109    0.076    0.000    0.076    0.000 {method 'getrandbits' of '_random.Random' objects}
```

```
$ python 2_2.py
Matrix multiplication 512 x 512 of int type.
CPU Time: 46.76811242103577
Process Time: 47.87381985

         138938639 function calls in 47.848 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1   37.103   37.103   46.768   46.768 2_2.py:12(matrixMultiplication)
        1    0.002    0.002    0.002    0.002 2_2.py:15(<listcomp>)
        2    0.141    0.070    1.076    0.538 2_2.py:26(random_init)
        1    0.001    0.001    0.001    0.001 2_2.py:31(<listcomp>)
        1    0.003    0.003    0.003    0.003 2_2.py:32(<listcomp>)
134217728    9.663    0.000    9.663    0.000 2_2.py:9(multiply)
   524288    0.254    0.000    0.353    0.000 random.py:235(_randbelow_with_getrandbits)
   524288    0.359    0.000    0.785    0.000 random.py:284(randrange)
   524288    0.150    0.000    0.935    0.000 random.py:358(randint)
  1572864    0.073    0.000    0.073    0.000 {built-in method _operator.index}
     1026    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {built-in method time.process_time}
        4    0.000    0.000    0.000    0.000 {built-in method time.time}
   524288    0.026    0.000    0.026    0.000 {method 'bit_length' of 'int' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
  1049857    0.073    0.000    0.073    0.000 {method 'getrandbits' of '_random.Random' objects}
```

Replaced the multiplication portion with a function call to multiply to evaluate the execution time for multiplying two numbers. Approximately 20% to 40% of the total execution time for matrix multiplication is attributed to the multiply function. 134,217,728 multiplications are required for 512*512 matrix multiplication.

c)



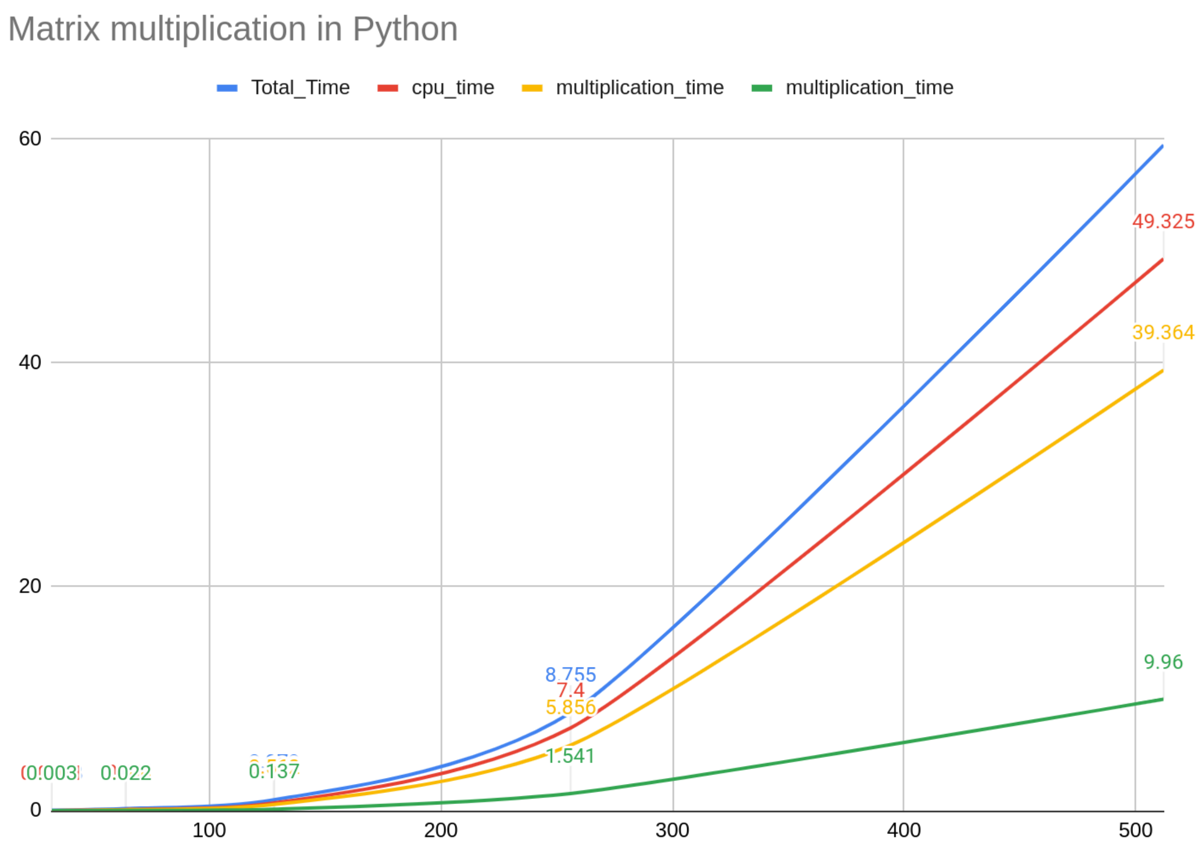
Time Difference:

For a 512x512 matrix multiplication, C requires 1.74 seconds (total time) with 1.728 seconds of CPU time, while Python requires 59.478 seconds (total) and 49.325 seconds of CPU time.

Despite the overall time difference, the ratio of CPU time to total time remains similar for both languages over N.

Curve Similarity:
The curves for CPU time, total time, matrix multiplication time, and multiplication function time versus N for N*N matrices exhibit similarity between C and Python.

The patterns suggest that the behavior of the matrix multiplication and multiplication function times is comparable in both languages.



[Github Repository](#)
[Google sheets](#)

Resources:
[Timespec](#)
[CPU Time Inquiry](#)
[gprof](#)