INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR
PALAJ, GANDHINAGAR, GUJARAT 382 055

Name : Bhavik Patel
Roll no. : 22110047
Branch : CSE

IITGN

ES 215: COA
Assignment 1

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).
   Tips: Measure the time taken by the program on the CPU using timespec.

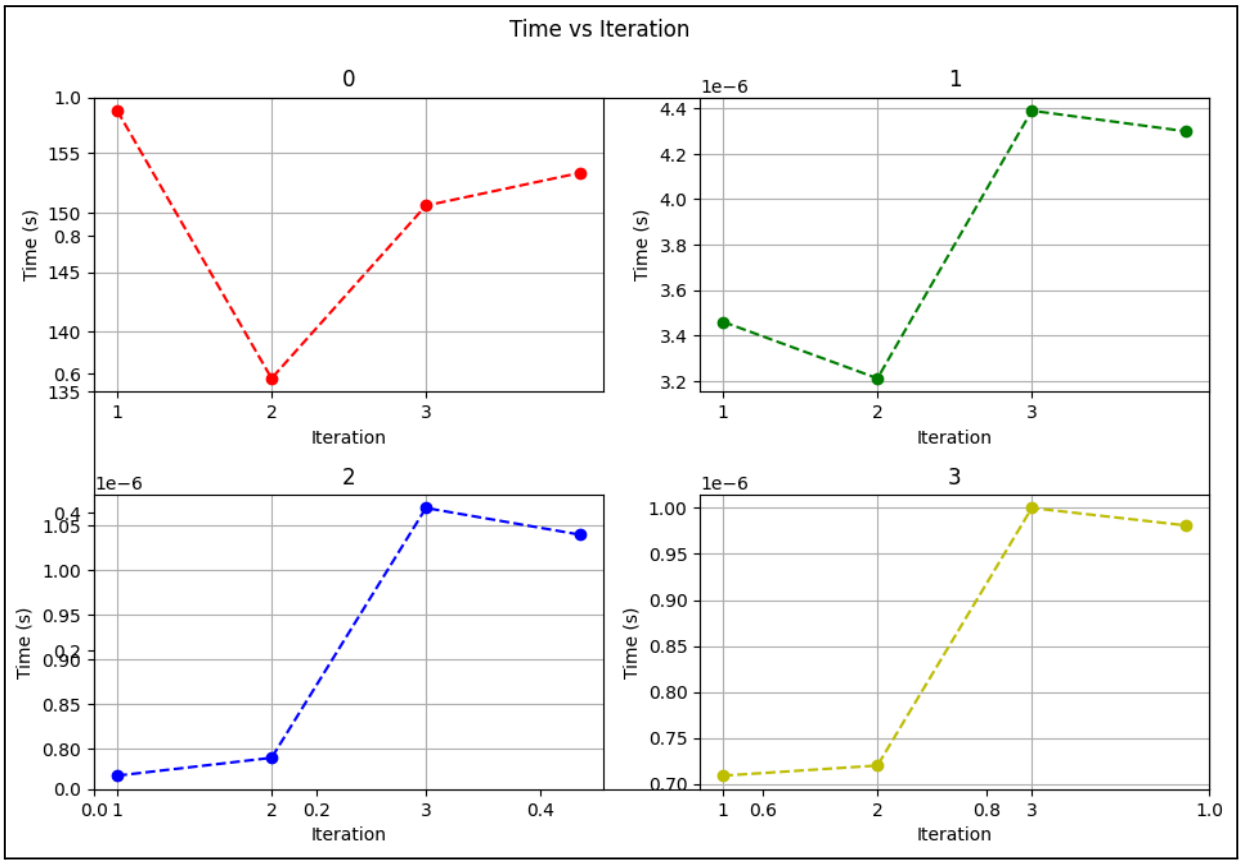
   Taking baseline to be the recursive program.

$$Speed\ Up\ program_1\ wrt\ program_2 = \frac{Execution\ time\ of\ program_2}{Execution\ time\ of\ by\ program_1}$$

Program	**Iteration 1**	**Iteration 2**	**Iteration 3**	**Iteration 4**	**Average Time Taken(sec)**
Recursion(Baseline)	158.54 sec	136.092 sec	150.579 sec	153.332 sec	149.6357 sec
Loop	3.46e-06 sec	3.211e-06 sec	4.39e-06 sec	4.299e-06 sec	3.84e-06 sec
Recursion+ Memoization	7.7e-07 sec	7.29e-07 sec	1.071e-06 sec	1.04e-06 sec	0.90225e-06 sec
Loop+Memoization	7.09e-07 sec	7.2e-07 sec	1e-06 sec	9.81e-07 sec	0.8525e-06 sec

For calculating speed-ups, we are considering the average time over 4 iterations of the program as the execution time of that program to get accurate results.

Speed up of recursion wrt recursion= 149.6357/149.6357=1 sec
Speed up of loop wrt to recursion = 149.6357/3.84e-06 = 3.88e07 sec
Speed up of (memoization+recursion) wrt recursion = 149.6357/9.0225e-07= 1.658e08 sec
Speed up of (loop+memoization) wrt recursion = 149.6357/8.525e-07 = 1.755e08 sec


Time vs Iteration
0
1
2
3
Time (s)
Iteration
1e−6

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 the 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)

   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 of a proportion it is w.r.t. of the total program execution time. (25 points)
   c. Plot the (a) and (b) execution times for each of the iterations. Compare the performance (System and program execution times) of the program for a given value of N for the languages in both buckets. –Illustrate your observations. (50 points)

Part a)

Bucket 1:

C++

Size of Matrix(NxN)	User CPU Time(sec)		System CPU Time	
	Integer	Float	Integer	Float
64	0.0 sec	0.01 sec	0.0	0.0
128	0.03 sec	0.04 sec	0.0	0.0
256	0.33 sec	0.17 sec	0.0	0.0
512	1.12 sec	0.88 sec	0.0	0.0
1024	10.68 sec	8.56 sec	0.01	0.02

Bucket 2:

Python

Size of Matrix(NxN)	User CPU Time(sec)		System CPU Time	
	Integer	Float	Integer	Float
64	0.04 sec	0.08 sec	0.02	0.02
128	0.17 sec	0.19 sec	0.02	0.02
256	1.05 sec	0.92 sec	0.03	0.02
512	8.06 sec	9.75 sec	0.03	0.02
1024	73.07 sec	69.71 sec	0.06	0.06

Part b) Using the 'language hooks', evaluate the execution time for the meat portions of the program and how much of a proportion it is w.r.t. of the total program execution time. (25 points)

C++(INTEGER)

Size of Matrix(NxN)	Meat Time	Total Execution Time	(Meat Time)/(Total execution time)
64	0.0	0.0	0
128	0.023045	0.03	0.7681
256	0.287672	0.33	0.8717
512	1.096072	1.12	0.9786
1024	10.5868	10.69	0.99034

C++(DOUBLE)

Size of Matrix(NxN)	Meat Time	Total Execution Time	(Meat Time)/(Total execution time)
64	0.002582	0.0	0
128	0.03764	0.04	0.941
256	0.151547	0.17	0.8914
512	0.83069	0.88	0.9439
1024	8.45014	8.58	0.9848

Python (INTEGER)

Size of Matrix(NxN)	Meat Time	Total Execution Time	(Meat Time)/(Total execution time)
64	0.02625	0.06	0.4375
128	0.11002	0.19	0.5790
256	0.88101	1.08	0.81575
512	7.96875	8.09	0.9850
1024	73.0810	73.13	0.99932

Python (DOUBLE)

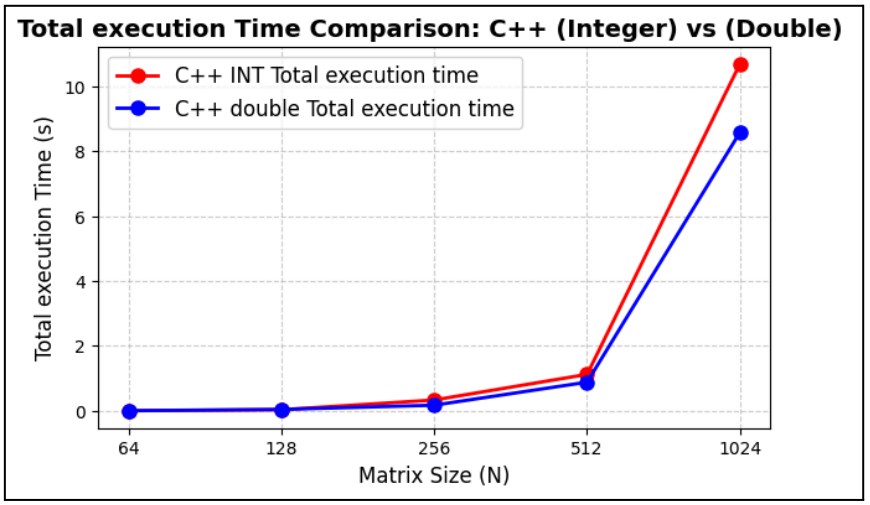
Size of Matrix(NxN)	Meat Time	Total Execution Time	(Meat Time)/(Total execution time)
64	0.05662	0.1	0.5662
128	0.109375	0.21	0.52083
256	0.8475	0.94	0.9015
512	9.44375	9.77	0.966
1024	69.6401	69.77	0.998

As the matrix size grows, the number of individual multiplications and additions increases. This means the core computation (meat time) dominates a larger portion of the total execution time. Therefore, the ratio of meat time to total execution time increases with an increase in matrix size.

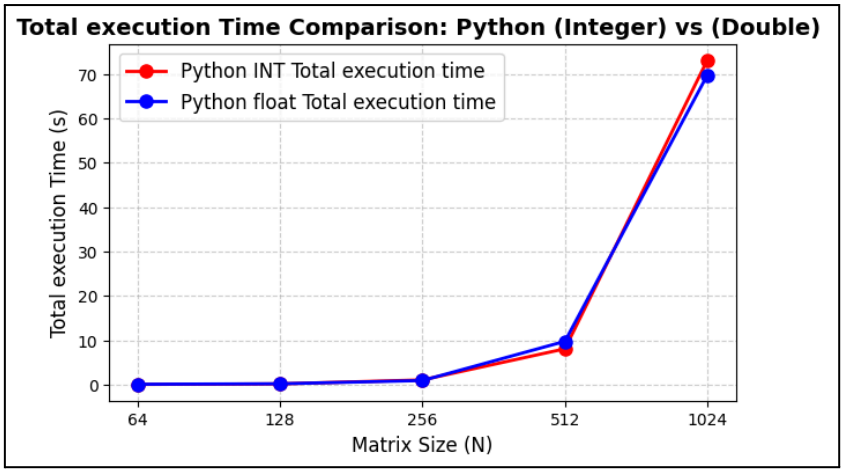
Assuming that "Meat time" refers to the time spent solely on the core computational work of a program, such as the actual multiplication and addition of matrix elements in a matrix multiplication task. It excludes any time taken for setup, initialization, or other peripheral tasks.

To calculate "meat time," I used **time.process_time()** in Python and **clock_gettime()** in C++. These functions measure the CPU time consumed by the process, focusing on the main computational work while excluding any I/O operations or time spent waiting for resources.

Comparing the Execution time of INTEGER and Double in C++



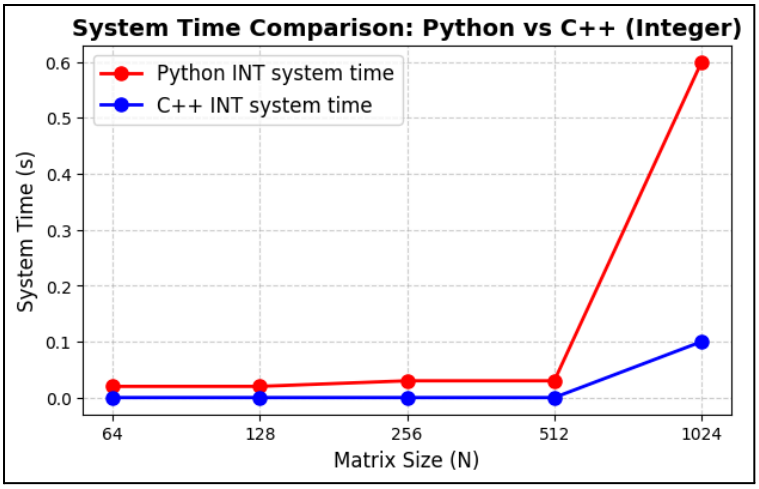
Comparing the Execution time of INTEGER and Double in Python

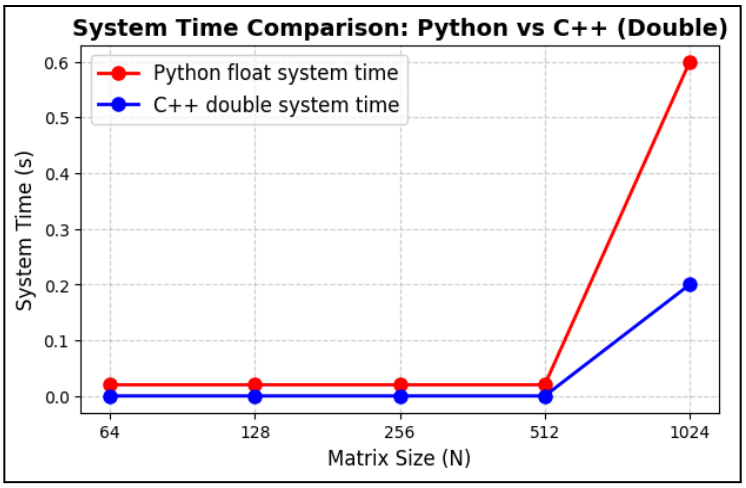


From the above plots, we can infer that matrix multiplication is computationally expensive for double data types than for integer data types for higher values of N(matrix size).

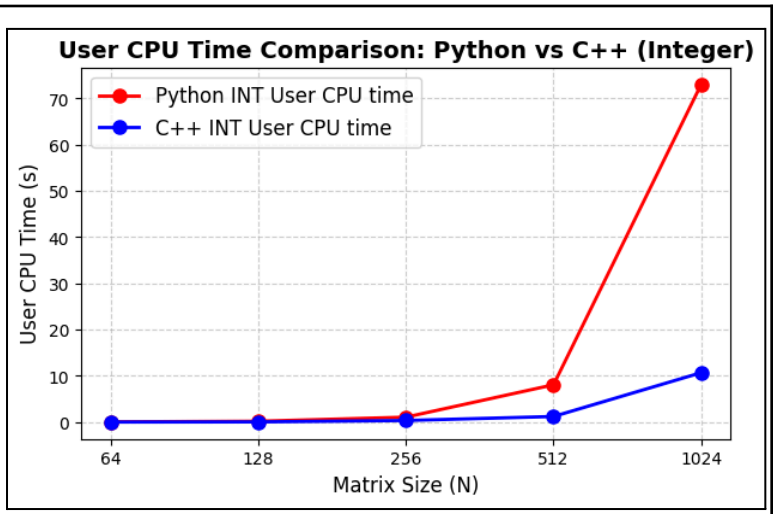
Now comparing the performance (System and Program execution times) of the program for given values of N for the languages in both the buckets.

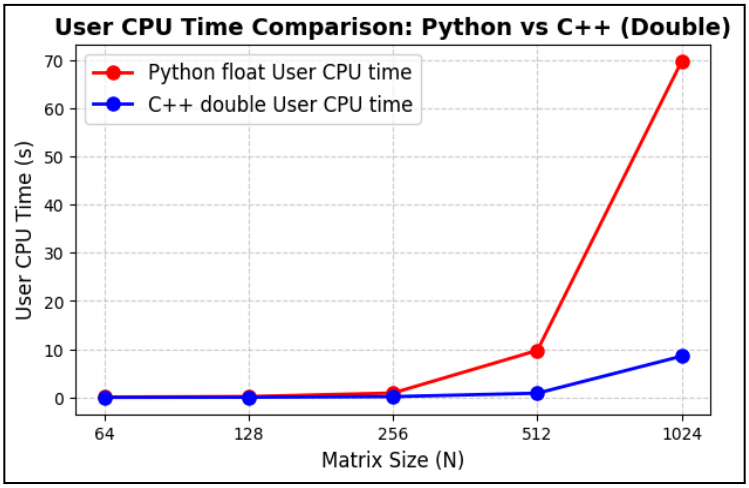
System CPU Time Comparision between both languages



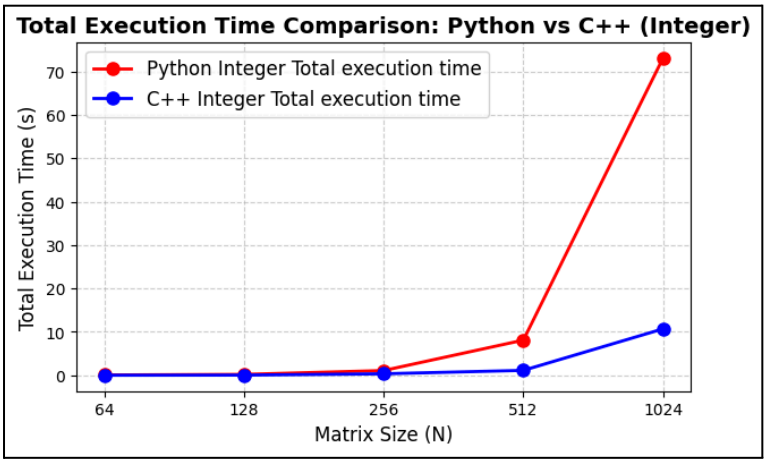


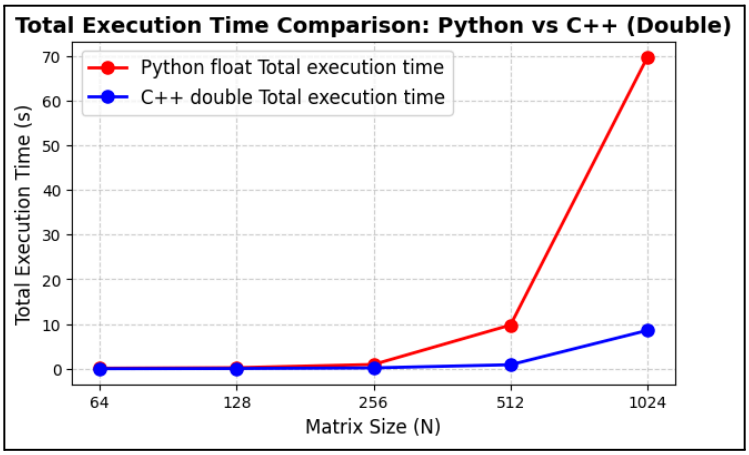
User CPU Time Comparision between both languages





Total Execution Time Comparision between both languages





Key Observations:

- The execution time for C++ is much lesser than Python therefore we can conclude that C++ is better than python on the basis of performance as we now that performance is inversely proportional to execution time.