Ruchit Jagodara
22110102

Github repo

Assignment 1

COA

- Question 1

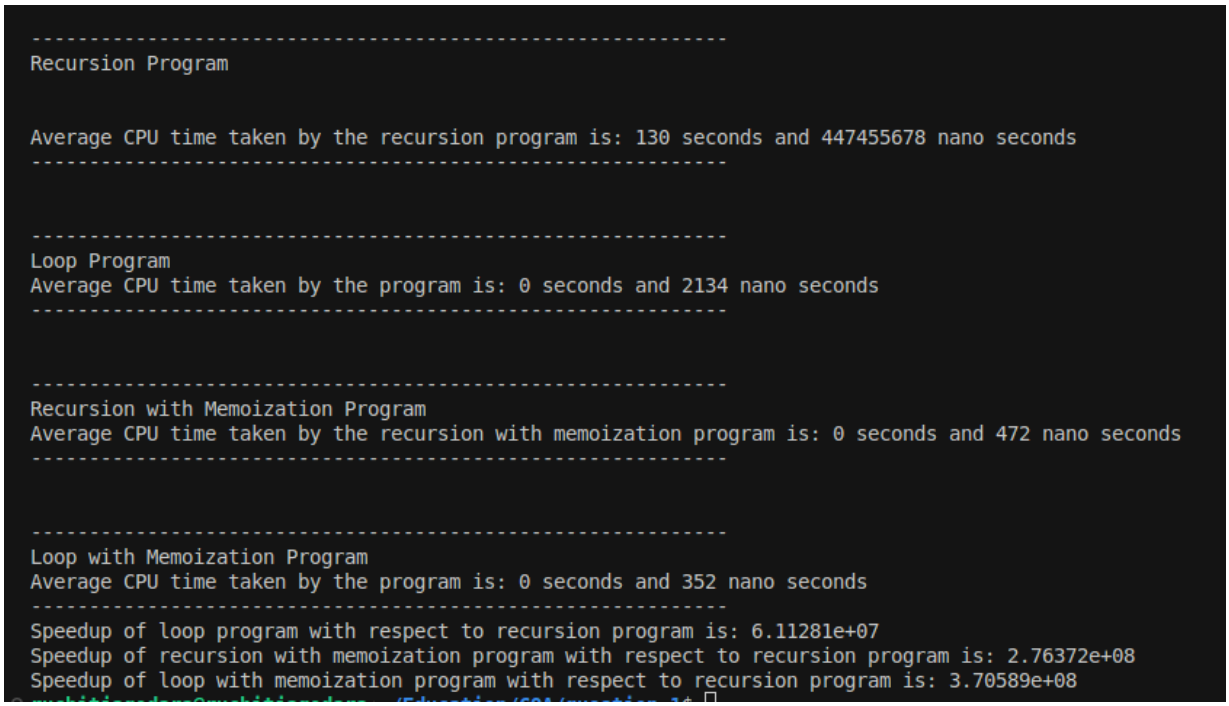
Assumption:-

Assuming the computer performance remains the same throughout the process, and only one thread of the CPU is working throughout the process so that the calculated time remains consistent throughout the process.

Assuming the time we get from the “clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);” functionality is the actual CPU time at that instant.

On the basis of the reference, the first Fibonacci number is 0, and the second one is 1.

Result:-



Here, I have run all the programs 5 times and then calculated the average time for all the programs and then calculating the speedups of all programs, the results that I get is like below,

1. Speedup of loop program with respect to recursion program is :- 6.11281e+07
2. Speedup of recursion with memoization program with respect to recursion program is :- 2.76372e+08
3. Speedup of loop with memoization program with respect to recursion program is:- 3.70589e+08

Here, we can notice that the speedup for the loop with memoization program is highest and which also makes sense as the total number of operations in it is lesser than other programs.

- Question 2

Assumption:-

Here, I have made different files for all different values of n as that was not mentioned in the assignment about how exactly we have to find the output of time and as in part (c), it is required to calculate the value of execution time for each iteration so I assumed that the professor wants us to write the program in this way only.

Here, instead of initializing the matrix to a random number or any specific value, instead as mentioned in the question I have hardcoded it to be the symmetric matrix containing all values = 2 in case of python and in case of cpp I am just initializing the matrices to maintain the accuracy of calculating the time.

(a):-

For CPP:-

1. For 64 int:-

output :-
real 0m0.005s
user 0m0.001s
sys 0m0.004s

2. For 128 int:-

output :-
real 0m0.016s
user 0m0.013s

sys 0m0.003s

3. For 256 int:-

output:-

real 0m0.049s
user 0m0.045s
sys 0m0.004s

4. For 512 int:-

output:-

real 0m0.333s
user 0m0.327s
sys 0m0.005s

5. For 1024 int:-

output:-

real 0m3.169s
user 0m3.161s
sys 0m0.009s

6. For 64 double:-

output:-

real 0m0.009s
user 0m0.005s
sys 0m0.004s

7. For 128 double:-

output:-

real 0m0.017s
user 0m0.016s
sys 0m0.001s

8. For 256 double:-

output:-

real 0m0.057s
user 0m0.052s
sys 0m0.005s

9. For 512 double:-

output:-

real 0m0.380s
user 0m0.374s
sys 0m0.006s

10. For 1024 double:-

output:-

real 0m3.266s
user 0m3.250s
sys 0m0.016s

For Python3:-

1. For 64 int:-

output :-

real 0m0.043s
user 0m0.037s
sys 0m0.006s

2. For 128 int:-

output :-

real 0m0.160s
user 0m0.154s

sys 0m0.006s

3. For 256 int:-

output:-

real 0m1.193s
user 0m1.184s
sys 0m0.009s

4. For 512 int:-

output:-

real 0m10.052s
user 0m10.034s
sys 0m0.018s

5. For 1024 int:-

output:-

real 0m3.169s
user 0m3.161s
sys 0m0.009s

6. For 64 double:-

output:-

real 0m0.009s
user 0m0.005s
sys 0m0.004s

7. For 128 double:-

output:-

real 0m0.017s
user 0m0.016s
sys 0m0.001s

8. For 256 double:-

output:-

```
real    0m0.057s
user    0m0.052s
sys     0m0.005s
```

9. For 512 double:-

output:-

```
real    0m0.380s
user    0m0.374s
sys     0m0.006s
```

10. For 1024 double:-

output:-

```
real    1m30.403s
user    1m30.353s
sys     0m0.038s
```

(b):-

Using the reference, I am calculating the execution time accordingly.

Below table is for int values.

Value of N	For CPP			For Python		
	Execution time for Meat portion (in ns)	Execution time (in ns)	Total portion of meat portion in total execution time (in %)	Execution time for Meat portion (in ns)	Execution time (in ns)	Total portion of meat portion in total execution time (in %)
64	990570	1039356	95.3061319	20467704	20526358	99.7142503
128	7283251	7366428	98.8708639	191006529	191396991	99.7959937
256	41950114	42176060	99.464279	1474530523	1475348335	99.9445682
512	524817928	526502979	99.6799541	11281678716	11287672998	99.9468953
1024	5591663459	5597542476	99.8949715	104329516694	104355293010	99.9752995

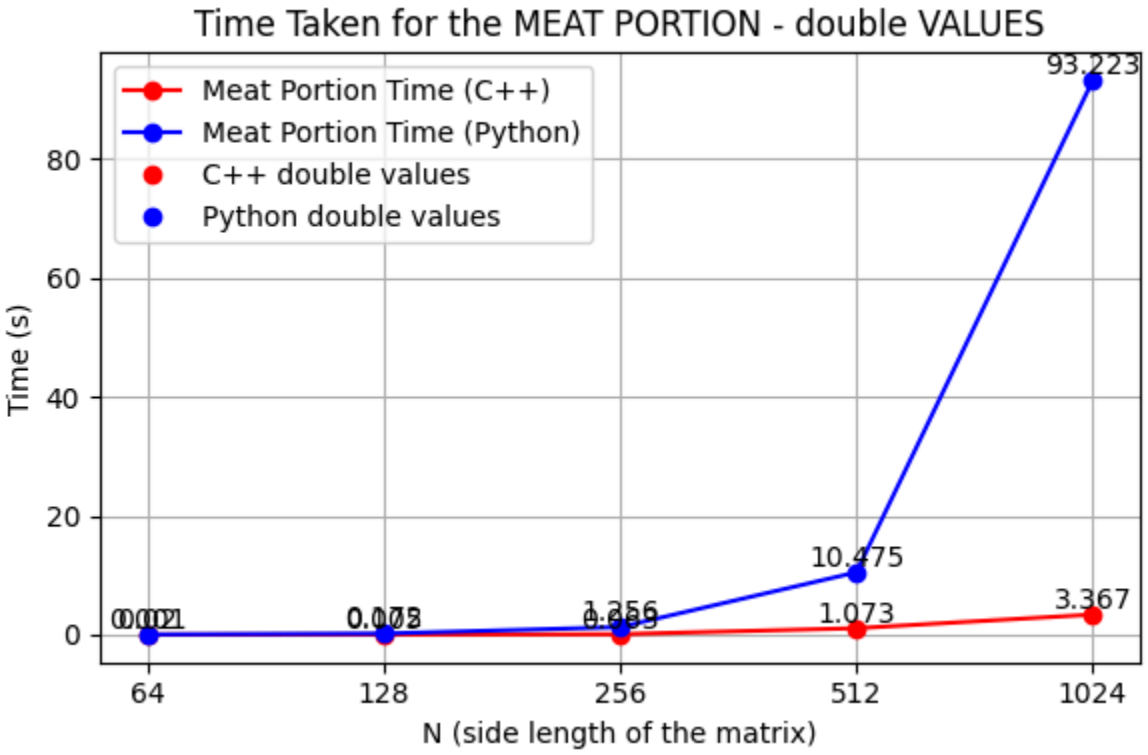
The below table is for double values.

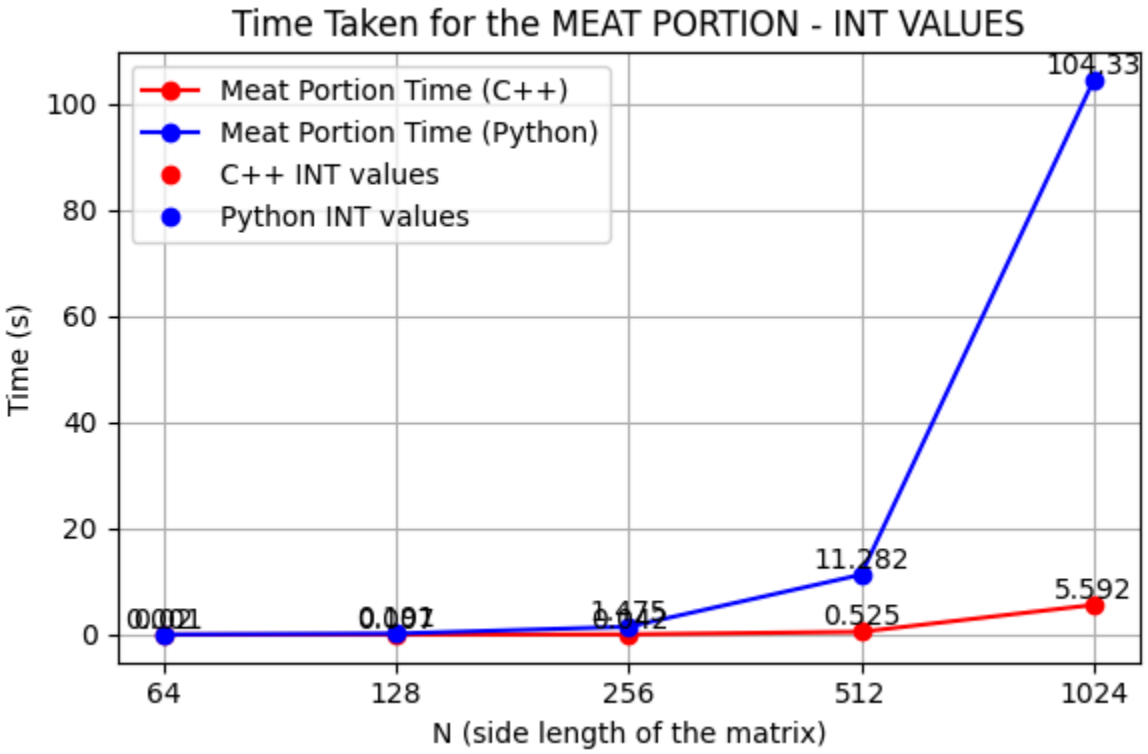
Value of N	For CPP			For Python		
	Execution time for Meat portion (in ns)	Execution time (in ns)	Total portion of meat portion in total execution time (in %)	Execution time for Meat portion (in ns)	Execution time (in ns)	Total portion of meat portion in total execution time (in %)
64	1015804	1077789	94.2488743	20321555	20403360	99.5990611
128	5315160	5464279	97.2710215	171813635	172535603	99.5815542
256	62623395	63612581	98.4449837	1256342557	1256950309	99.9516487
512	1073431535	1075100475	99.8447643	10475069672	10482405602	99.9300167
1024	3366783716	3373826888	99.7912409	93223005177	93253612960	99.9671779

Here, all values specified above are also displayed in the output of every code, so if you run the code of mat_mul_int_64.cpp it will display you the total execution time and the execution time of the meat portion for cpp language.

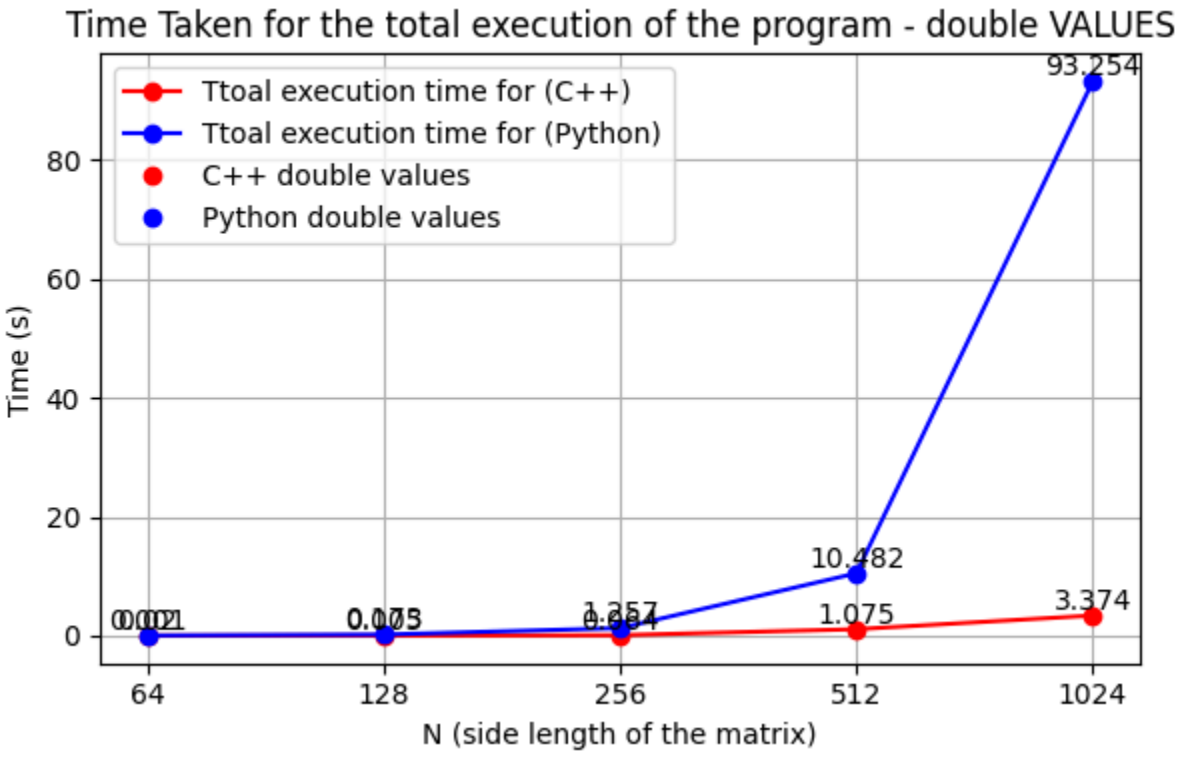
(c):-

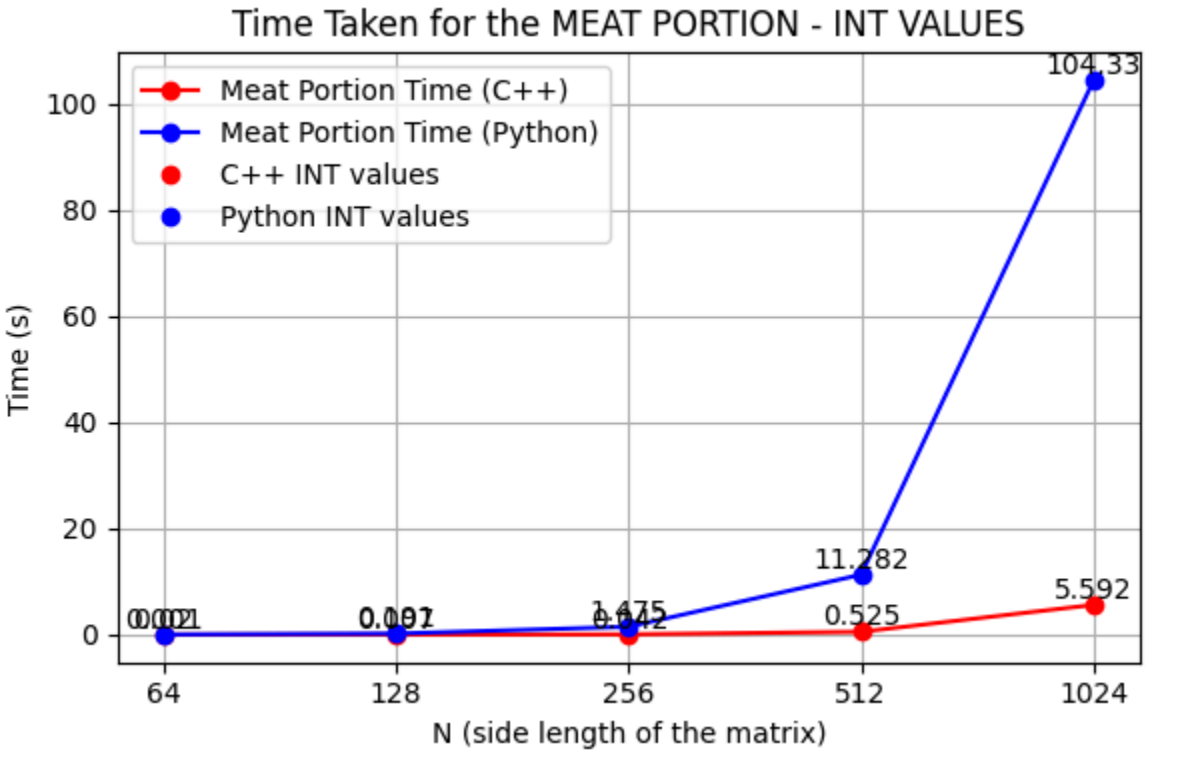
Below are plots of execution time of meat parts:-



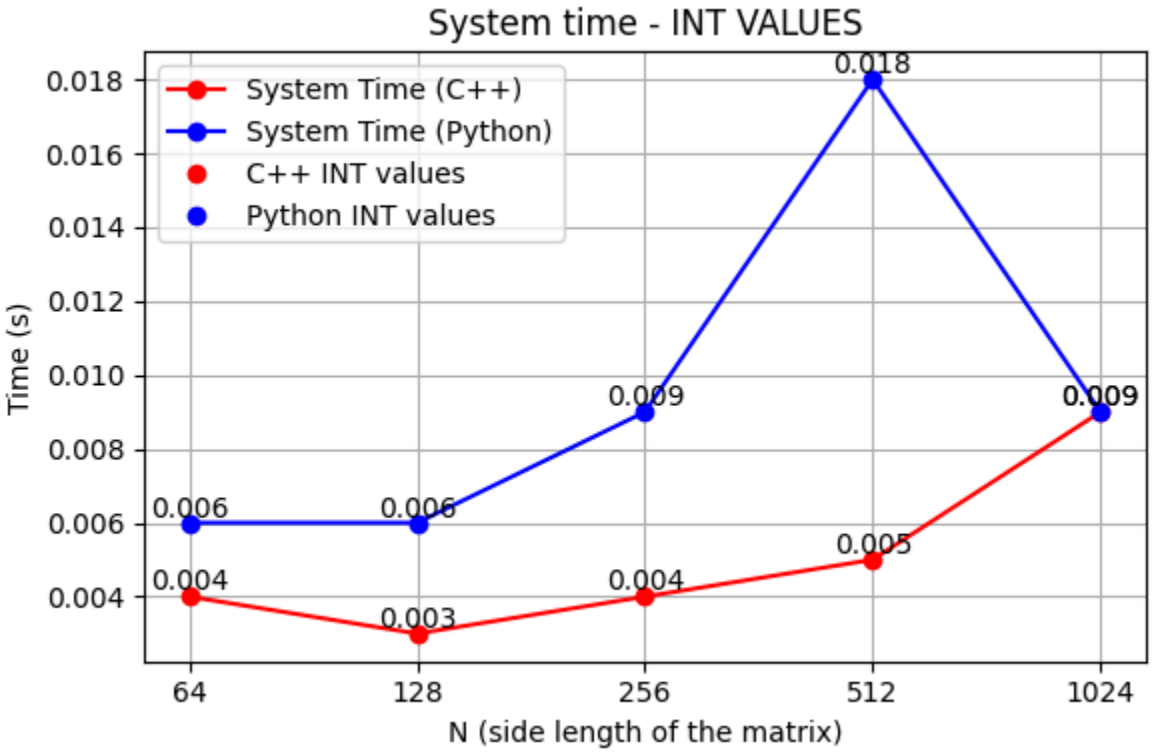


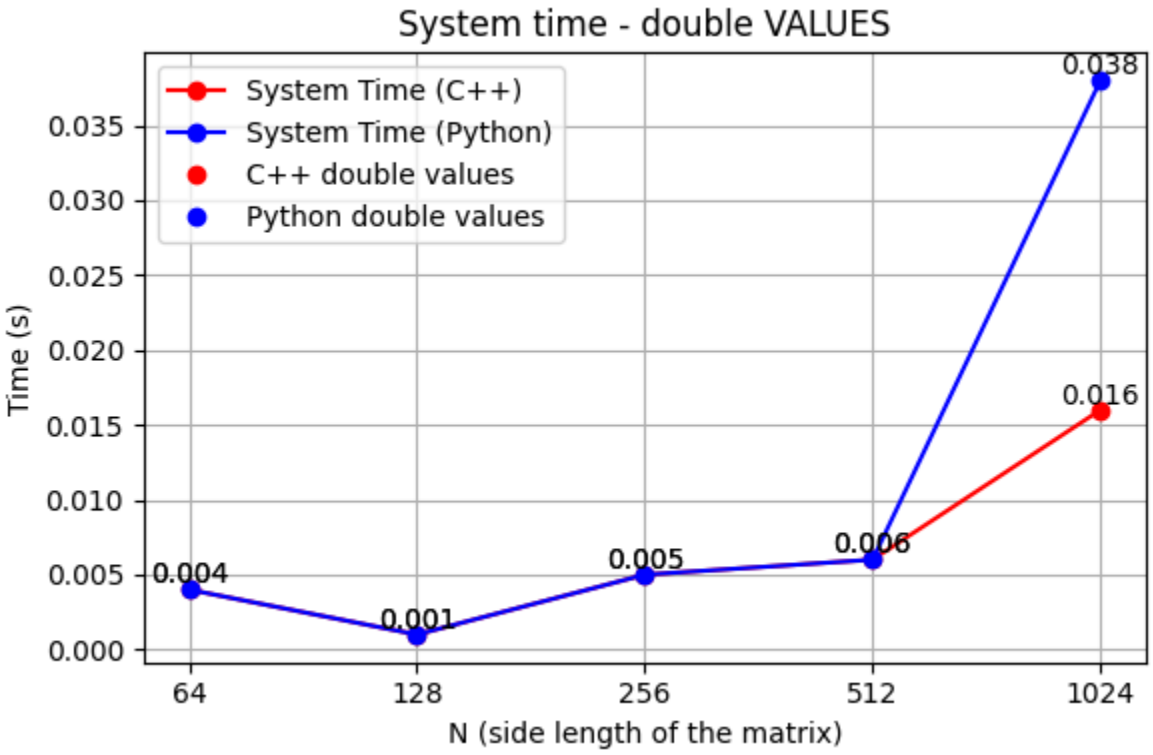
Below are the plots of the total execution time:-





Below are graphs of the system times:-





Observations:-

Here, we can notice that the average system time of Python for any value of N is greater than or equal to the value of system time of cpp for that particular N.

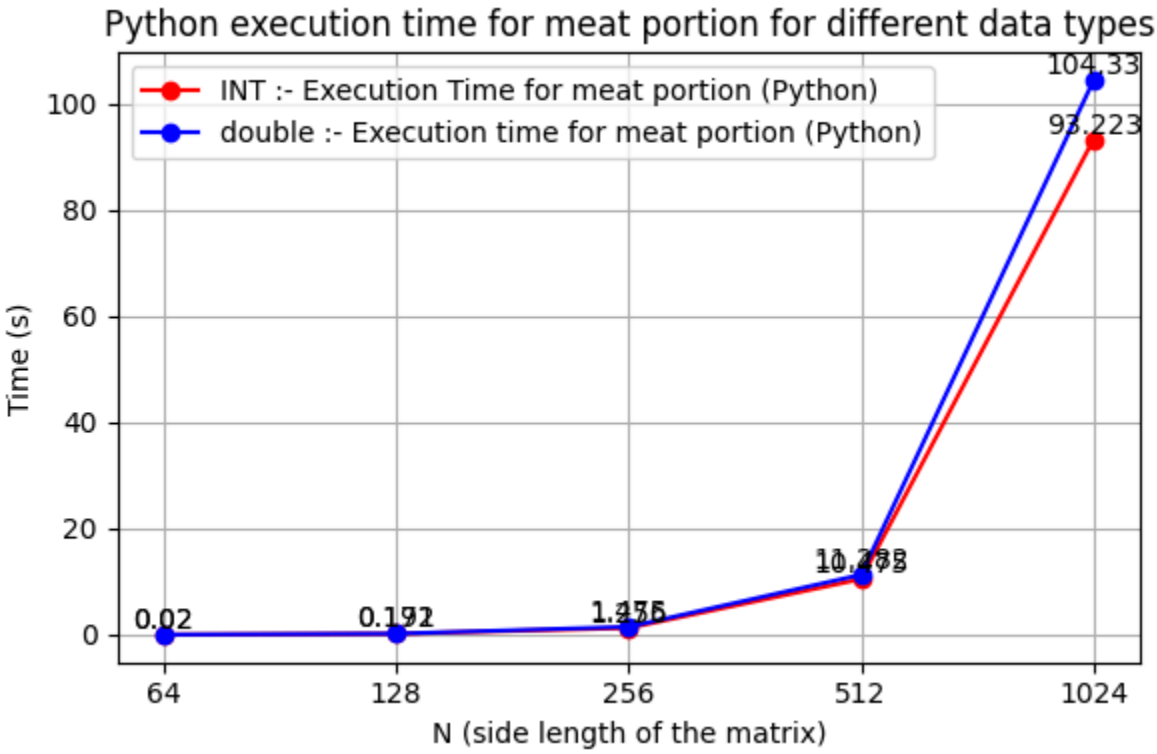
This is happening because python uses automatic memory management, including garbage collection, while cpp allows more control over memory management, including manual allocation and deallocation, enabling more efficient use of resources.

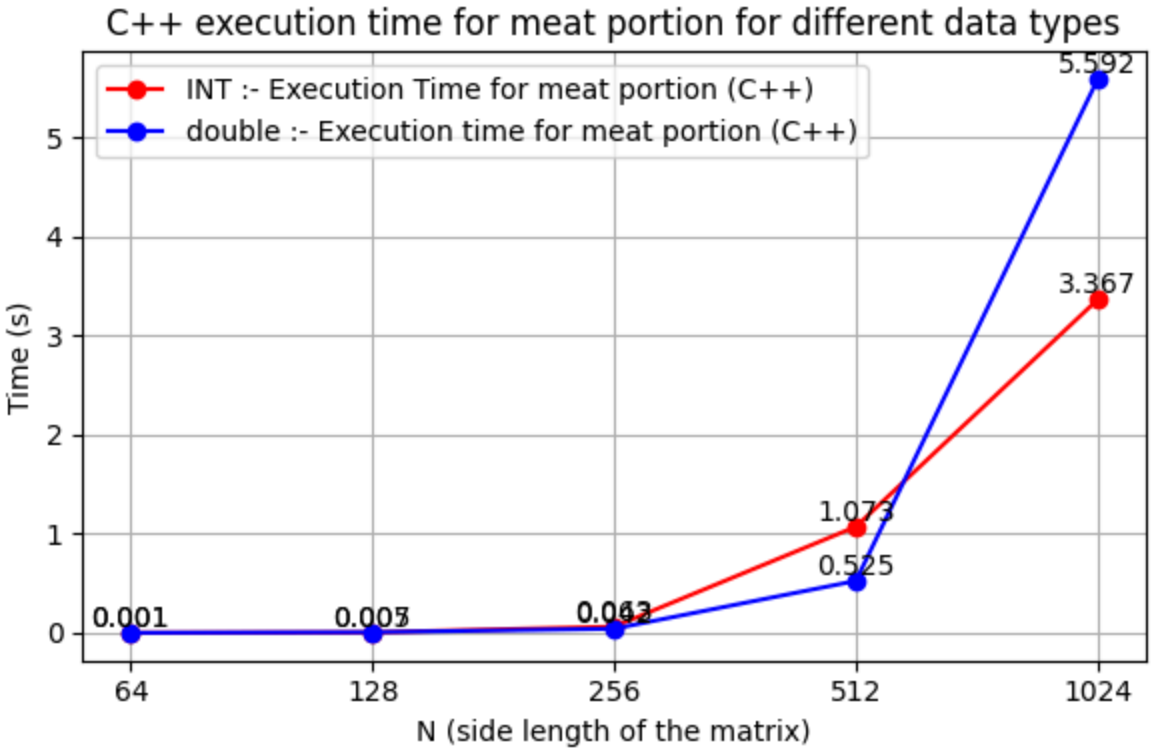
Also, because python is a interpreted language while cpp is a compiled language and python's code is executed line by line by the python interpreter at the runtime. Because of this it increases the system time.

Here, in the graphs related to meat portion execution time and total execution time for both int and double values we can see that python takes more time than cpp in all the cases.

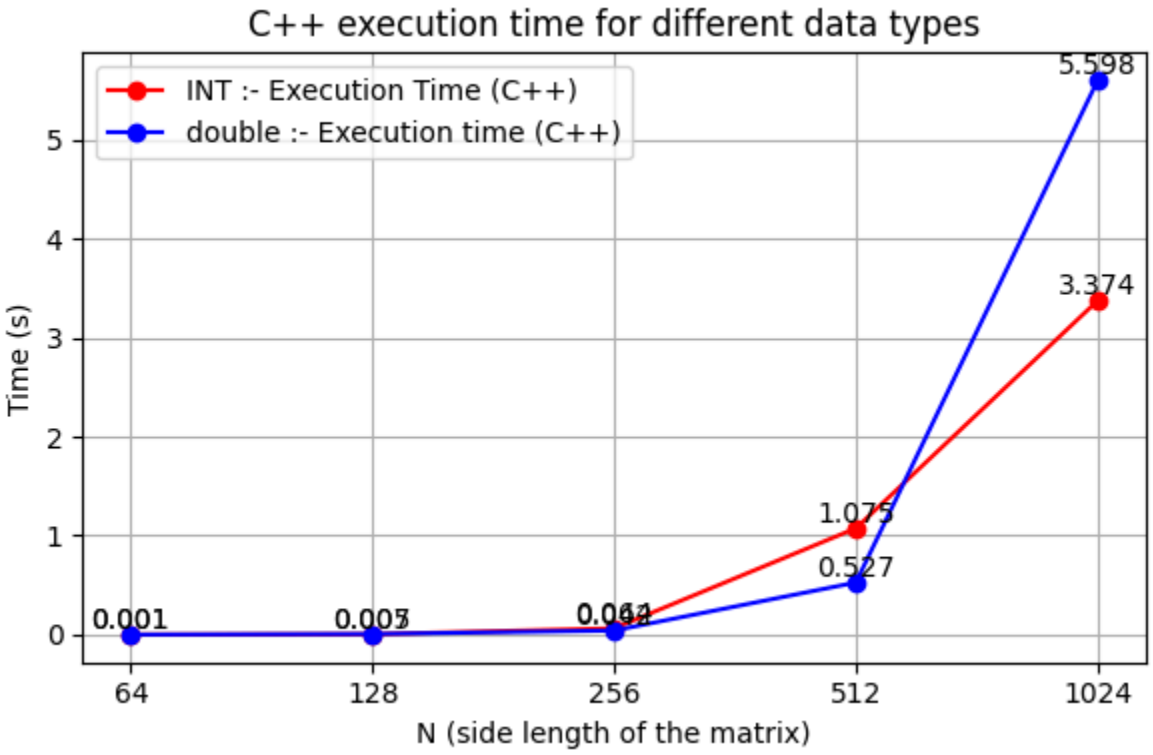
Below are the plots giving comparisons between different data types of the same language.

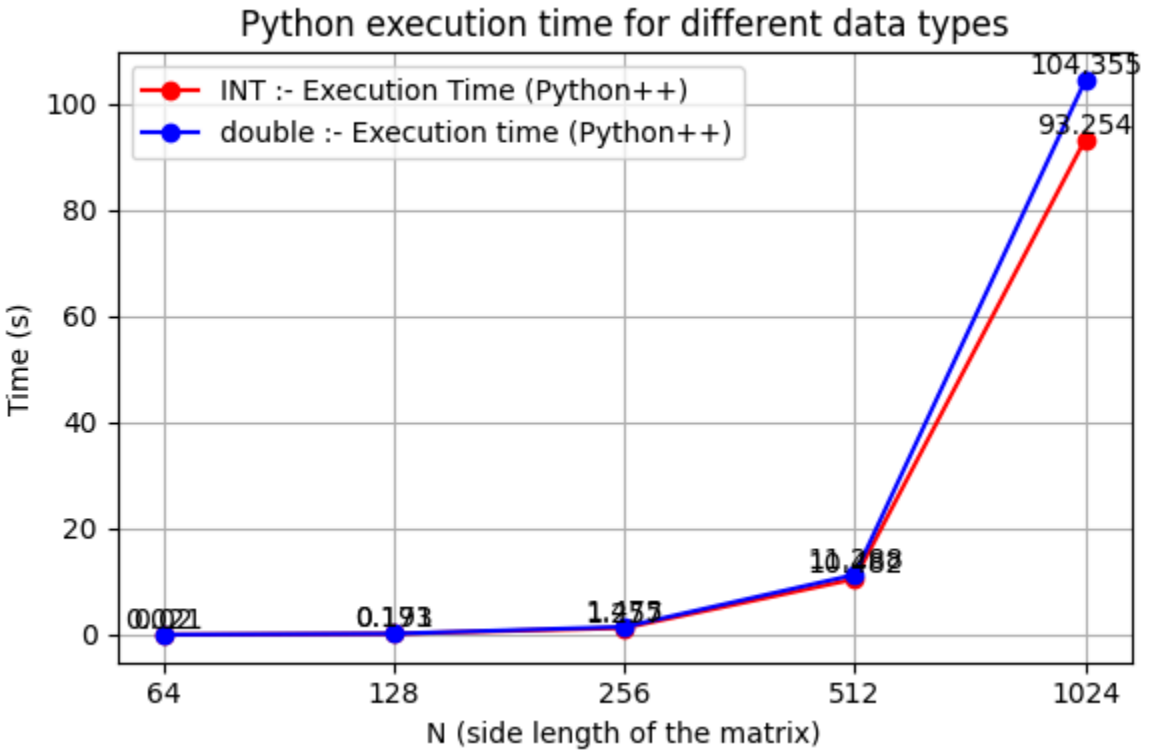
Below are the plots for different languages about the meat portion of execution time, giving a comparison between INT and double data types.



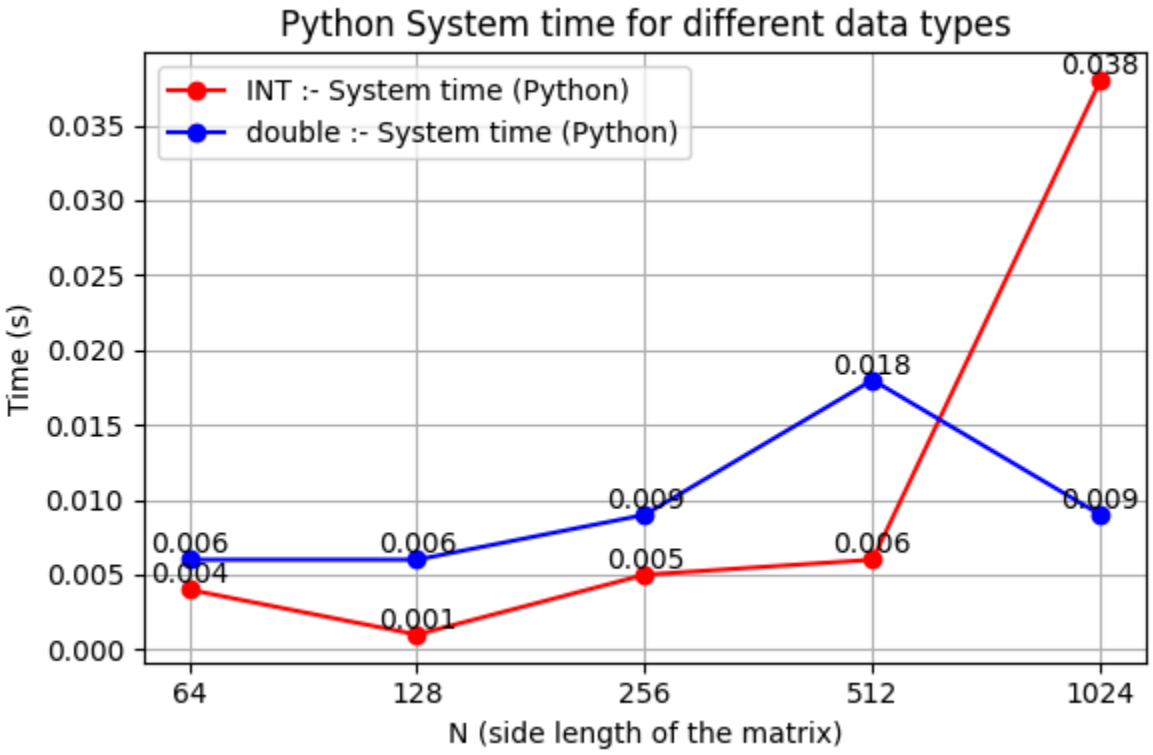

C++ execution time for meat portion for different data types
INT :- Execution Time for meat portion (C++)
double :- Execution time for meat portion (C++)
Time (s)
N (side length of the matrix)
64
128
256
512
1024
0
1
2
3
4
5
0.001
1.073
0.525
3.367
5.592

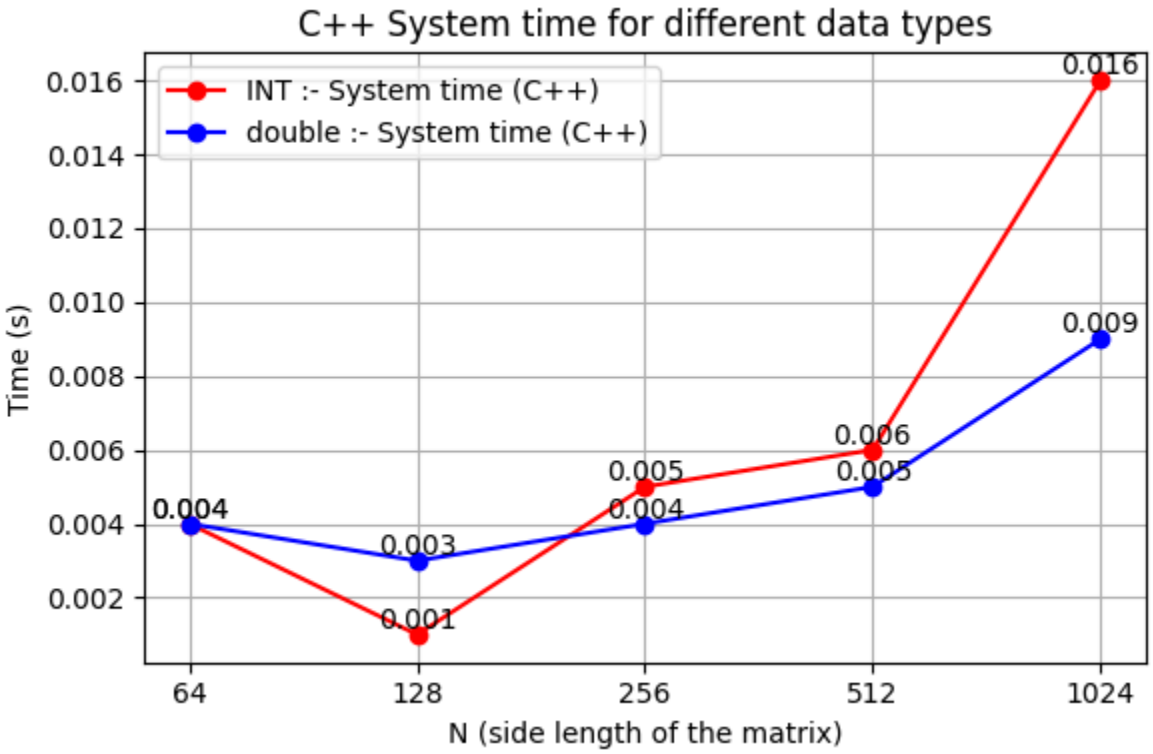
Below are plots of different languages about total execution time, giving a comparion between INT and double data types.





Below are plots of different languages about system time, giving a comparison between INT and double data types.





Observations:-

Here, generally we can see that INT data type has lower values of time as compared to double.

In graphs, we can see that in some cases the int data type is taking more time than double data type as in the case of system time for c++ and this is because of the uncertainty in reading of the system time through time command. But in general, int data type takes lesser time than double.

The int type is simpler in terms of storage and arithmetic operations, as it doesn't involve the additional overhead of handling floating-point precision, which is necessary for double types. This results in faster processing times for int operations in most cases.