COA Assignment 1

Roll Number: 21310060

Question 2:

Matrix Multiplication

Bucket 2: Python

```
import time

import numpy as np

start = time.time( )

n=int(input("Enter the matrix size: "))

array1 = np.random.randint(1,10,size=(n,n))   // Random NxN Matrix 1

print(array1)

array2 = np.random.randint(1,10,size=(n,n))    // Random NxN Matrix 2

print(array2)

result =np.zeros([n, n], dtype = int)

# iterating by row of array1

for i in range(len(array1)):

    # iterating by column by array2

    for j in range(len(array2[0])):

        # iterating by rows of array2

        for k in range(len(array2)):

            result[i][j] += array1[i][k] * array2[k][j]

for r in result:

    print(r)

end = time.time( )
```

```
print("The complete time taken for execution of above program is :", end-start
```

2(a)

```
[gandhi@workstation ~]$ time python mat.py
Enter the matrix size: 32
[[3 9 9 ... 3 6 8]
 [7 9 1 ... 2 4 2]
 [5 6 3 ... 2 2 9]
 ...
 [1 4 2 ... 6 5 6]
 [4 3 7 ... 7 9 1]
 [8 4 5 ... 2 2 7]]
[[4 7 2 ... 6 2 7]
 [5 4 4 ... 6 6 2]
 [6 7 8 ... 2 8 7]
 ...
 [7 3 1 ... 6 3 1]
 [8 8 2 ... 8 6 9]
 [2 4 7 ... 7 5 8]]
[763 788 810 902 819 781 799 857 822 971 855 886 801 791 707 848 748 694
 830 813 696 771 700 846 749 802 830 754 754 629 765 927]
[765 783 837 857 779 749 817 892 807 999 929 837 821 855 715 859 858 802
 784 822 699 751 792 817 797 767 786 815 759 657 774 956]
[ 815  795  870  885  882  764  813  950  910 1037  921  892  797  768
  700  867  820  764  773  816  776  835  797  880  758  730  827  798
  734  637  768 1022]
[789 776 796 915 833 756 768 976 834 997 944 813 773 689 705 873 817 744
 729 765 685 822 754 763 763 729 725 722 764 609 693 963]
[ 855  956  936  967  983  878  863 1050  903 1104  994  993  894  871
  785  870  920  871  901  915  740  872  870  912  888  863  940  855
  826  671  877 1122]
[ 807  839  792  914  873  829  910  993  820 1011  926  909  856  835
  701  887  844  720  827  831  705  844  778  898  868  776  947  714
  741  704  833 1007]
[753 777 771 819 715 734 714 786 715 885 885 788 738 766 608 787 724 699
 757 762 545 697 761 800 746 790 749 744 673 577 698 851]
[784 801 862 870 825 852 791 948 872 985 958 827 780 789 709 880 831 762
 741 815 701 852 846 793 804 801 780 798 778 649 693 980]
[ 947  916  899 1081  956  884  934 1087  930 1168 1074  994  874  856
```

```
The time of execution of above program is : 2.261556625366211

real    0m2.377s
user    0m0.150s
sys     0m0.031s
```

```
[gandhi@workstation ~]$ time python mat.py
Enter the matrix size: 64
[[8 6 8 ... 6 9 4]
 [6 4 8 ... 4 9 3]
 [8 3 5 ... 7 4 7]
 ...
 [3 3 1 ... 6 9 3]
 [1 1 5 ... 3 3 1]
 [9 6 1 ... 1 6 1]]
[[3 1 3 ... 2 9 3]
 [2 6 3 ... 7 6 5]
 [6 5 8 ... 2 2 5]
 ...
 [2 2 1 ... 7 8 4]
 [4 2 8 ... 8 7 3]
 [7 1 9 ... 8 8 1]]
[1576 1348 1461 1496 1405 1536 1480 1613 1679 1649 1741 1733 1512 1476
 1769 1314 1873 1437 1489 1576 1725 1424 1378 1671 1428 1546 1837 1455
 1476 1583 1683 1547 1543 1688 1540 1832 1518 1506 1505 1485 1583 1563
 1723 1531 1513 1766 1397 1703 1494 1394 1334 1703 1686 1636 1308 1575
 1736 1650 1530 1797 1471 1602 1441 1430]
[1784 1489 1624 1663 1545 1655 1677 1726 1881 1699 1907 1995 1659 1623
 1994 1468 2008 1719 1591 1847 1988 1565 1705 1758 1556 1699 1873 1592
 1754 1818 1756 1808 1833 1962 1590 1856 1766 1703 1819 1769 1690 1706
 1820 1726 1642 1835 1562 1667 1588 1531 1375 1694 1791 1850 1666 1762
 1938 1765 1752 1884 1677 1785 1693 1749]
[1829 1446 1586 1593 1659 1749 1598 1842 1885 1765 1996 2032 1652 1661
 1984 1476 2040 1789 1686 1875 1891 1563 1608 1828 1533 1763 1874 1703
 1679 1753 1868 1790 1864 1872 1553 1938 1678 1723 1780 1660 1765 1690
 1997 1718 1546 1839 1528 1719 1575 1592 1516 1731 1868 1942 1524 1744
 1842 1780 1662 1888 1574 1726 1695 1590]
```

```
The time of execution of above program is : 3.753375291824341

real    0m3.832s
user    0m0.221s
sys     0m0.026s
```

CPU time taken is 0.221 seconds.

System time taken is **0.026** seconds.

```
[gandhi@workstation ~]$ time python mat.py
Enter the matrix size: 128
[[4 3 9 ... 8 1 1]
 [7 7 4 ... 3 1 7]
 [3 7 7 ... 9 9 5]
 ...
 [8 7 4 ... 1 9 8]
 [9 9 9 ... 8 3 5]
 [2 9 5 ... 9 1 1]]
[[4 5 3 ... 7 9 9]
 [9 3 5 ... 9 5 4]
 [6 1 8 ... 7 3 5]
 ...
 [5 5 8 ... 1 4 2]
 [8 8 7 ... 7 9 4]
 [1 1 9 ... 2 2 6]]
[3207 3277 3336 2979 3233 3073 3309 3201 3202 3255 3164 3152 3186 3370
 3064 3302 3235 3219 3191 3412 3288 3286 3369 3205 3085 3352 3142 3345
 3130 3154 3042 3109 3183 3117 3104 3252 3019 2865 3084 3131 3082 3162
 3290 3357 3196 2974 2931 3046 2797 3121 3340 3086 3354 3189 3119 3187
 3438 3394 3128 3066 3288 3002 3200 2868 2918 2980 3202 3031 3070 3295
 3125 2821 2808 3324 3310 3209 3217 3200 3061 3053 3095 3434 3074 3256
 3064 3350 3256 3216 3093 3299 3226 3315 3243 2651 2999 3015 3308 3409
 3393 3051 3115 3352 3198 3133 3439 3106 3459 3295 3192 3090 3300 2978
 3420 3101 3219 3190 3049 3131 3386 3119 2922 2935 3264 3213 3233 3435
 3159 3373]
[3130 3099 3151 2908 3240 2992 3177 3302 3226 3273 3158 3321 3051 3132
 3128 3256 2967 3039 3229 3459 3078 3117 3211 3328 3135 3254 3115 3184
 3099 3136 2821 3120 3036 3083 2991 3240 2792 2954 3138 3025 3010 2811
 3381 3311 3385 3041 3090 3246 2991 3142 3315 3032 3313 3248 3214 3108
 3215 3302 3237 3173 3478 3189 3202 2778 3004 3101 3353 3112 2811 3340
 3319 3094 2883 3212 3144 3304 3067 3115 3177 2733 3072 3323 2933 3308
 2950 3214 3339 3291 3088 3240 3203 3262 3131 2724 3014 2984 3251 3184
 3146 2952 2909 3437 3364 3054 3340 2875 3311 3267 3142 3000 3377 2969
 3276 3068 3121 3198 3193 2884 3219 3030 2903 2912 3188 3307 3019 3322
 3206 3260]

The time of execution of above program is : 3.3299622535705566

real    0m3.409s
user    0m1.275s
sys     0m0.018s
```

```
Enter the matrix size: 256
[[6 8 1 ... 7 6 2]
 [2 7 1 ... 5 4 6]
 [7 4 7 ... 2 6 1]
 ...
 [1 6 3 ... 9 5 6]
 [7 6 4 ... 6 8 5]
 [7 9 9 ... 2 5 2]]
[[3 4 2 ... 8 2 9]
 [2 4 9 ... 5 5 9]
 [4 4 3 ... 4 2 5]
 ...
 [6 9 5 ... 3 1 4]
 [9 8 8 ... 7 6 6]
 [7 4 5 ... 9 1 4]]
[6990 6870 6781 6665 6932 6401 6312 7006 6948 6806 6659 6785 6985 6674
 6516 6557 6650 6767 6375 6508 6873 6506 6570 6364 6869 6983 6527 6575
 6665 6582 6518 6756 6918 6890 6699 6429 7179 6538 6729 6542 6405 7211
 6821 7294 6639 6743 7098 6376 6587 6744 6451 6725 6737 6613 6681 6465
 6979 6441 6605 6549 7029 6624 6704 6749 6402 6573 6588 6877 6574 6767
 6215 6340 6887 6146 6474 7120 6627 6394 6759 6845 6475 6469 6915 6937
 6349 6768 6782 6952 6506 6552 6596 6351 6970 6579 6518 6762 6585 6851
 6103 6500 7112 7240 6735 6425 6513 5974 6704 6623 6449 6724 6591 6811
 7142 6821 6680 6913 6574 6639 6613 6744 6671 6336 6693 6927 6289 6992
 6640 6533 6591 6459 6928 6563 6894 6655 6570 6647 6741 6097 6823 6089
 6926 6570 6515 6795 6735 6477 6718 6362 6464 6771 7015 6943 6484 6204
 6484 6900 6731 6683 6382 6431 6389 6598 6542 7236 6373 6670 6369 6585
 6645 6496 6931 7191 6825 6356 6849 7017 6994 6370 6412 6596 6621 6330
 6837 6621 6776 6802 6742 6366 6743 6968 6557 6135 6894 6490 6490 6417
 6579 6749 6715 6066 6700 6425 6534 6766 6718 6807 6095 6253 7038 6666
 6392 6744 6823 6429 6492 6490 6534 6867 6300 6731 6826 6495 6690 6584
 6864 6437 6340 6545 6933 6736 6679 7038 6209 6602 6367 6655 6603 6638
 6403 6254 6720 6528 6331 6250 6571 6678 6742 6903 6503 6503 6722 6847
 6155 6625 7018 7166]
```

```
The time of execution of above program is : 12.75303053855896

real    0m12.872s
user    0m9.921s
sys     0m0.034s
```

```
[gandhi@workstation ~]$ time python mat.py
Enter the matrix size: 512
[[8 6 4 ... 2 8 2]
 [7 4 9 ... 8 6 9]
 [5 7 4 ... 8 7 5]
 ...
 [6 6 7 ... 7 6 4]
 [4 4 1 ... 6 6 9]
 [5 2 8 ... 5 6 5]]
[[8 5 9 ... 5 5 2]
 [4 6 2 ... 8 5 5]
 [5 2 7 ... 2 8 5]
 ...
 [9 8 4 ... 7 9 6]
 [9 5 9 ... 4 8 1]
 [7 3 5 ... 8 3 5]]

The time of execution of above program is : 87.52591490745544

real    1m27.600s
user    1m25.271s
sys     0m0.061s
```

Summary:

Matrix Size	System Time (Sec)	CPU time(Sec)
32	0.031	0.15
64	0.026	0.221
128	0.018	1.275
256	0.034	9.921
512	0.061	1m 25.271

2(b) Language Hooks

Results : With Integer Values

Enter the matrix size: **32**

The time taken for execution of only multiplication is :

0.06555700302124023

The complete time taken for execution of above program is :
1.6687030792236328

Enter the matrix size: **64**
The time taken for execution of only multiplication is :
0.25228381156921387
The complete time taken for execution of above program is :
2.1250040531158447

Enter the matrix size: **128**
The time taken for execution of only multiplication is :
1.6973381042480469
The complete time taken for execution of above program is :
4.109076976776123

Enter the matrix size: **256**
The time taken for execution of only multiplication is :
13.124876022338867
The complete time taken for execution of above program is :
21.24715304374695

Enter the matrix size: **512**
The time taken for execution of only multiplication is :
107.68445014953613
The complete time taken for execution of above program is :
111.74628901481628

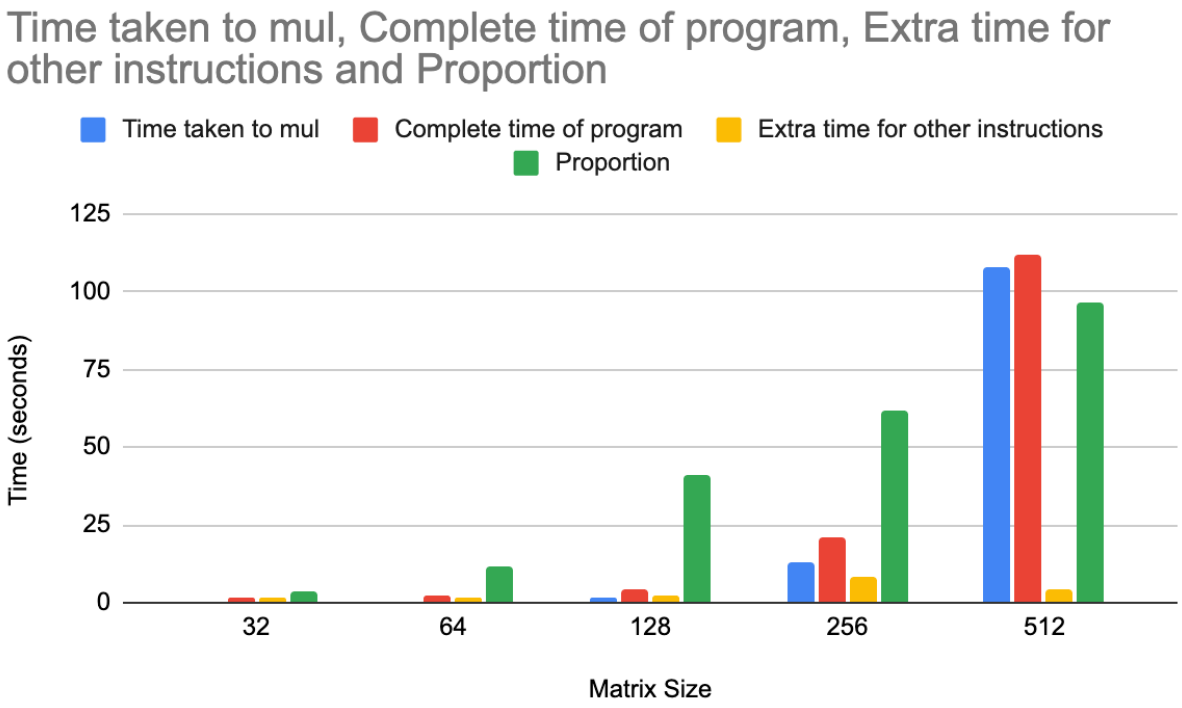
2.(C)

Matrix Size	Time taken to mul (Seconds)	Complete time of program (Seconds)	Extra time for other instructions (Seconds)	Proportion (Seconds)
32	0.06555700302	1.668703079	1.603146076	3.928620007
64	0.2522838116	2.125004053	1.872720242	11.87215672
128	1.697338104	4.109076977	2.411738873	41.30704082

256	13.12487602	21.24715304	8.122277021	61.77239838
512	107.6844501	111.746289	4.061838865	96.36512416

Proportion = Time taken to mul/Complete time of program

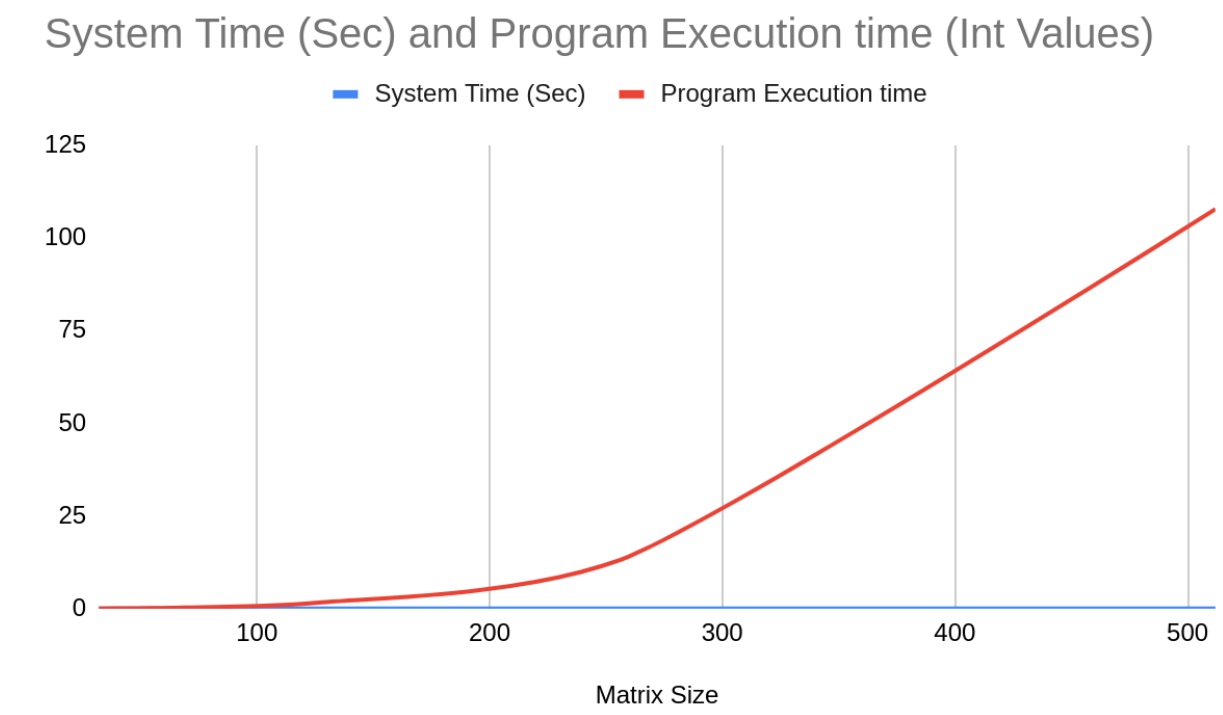
Extra time for other instructions = (Complete time of program) -(Time taken to mul)



Comparison of System time and Program execution time

Matrix Size	System Time (Sec)	Program Execution time
32	0.031	0.06555700302
64	0.026	0.2522838116
128	0.018	1.697338104
256	0.034	13.12487602

512	0.061	107.6844501



Results : With Double Type

A) System Time and CPU Time

```
[gandhi@workstation ~]$ time python mat_double.py
Enter the matrix size: 32
[[5.88075941 1.76308927 8.5087149  ... 5.49649294 1.47860081 7.2899872 ]
 [3.99656757 6.04114978 7.196698   ... 8.96466604 4.12733805 8.59187913]
 [3.02354193 4.89981667 3.06597284 ... 9.82907709 5.79257268 7.54497478]
 ...
 [2.88400458 8.29724608 1.31143493 ... 9.12305625 1.98350552 7.02629196]
 [7.03639861 1.0748162  5.42625229 ... 2.79461034 5.33423008 4.79178444]
 [1.27510053 4.72059812 7.3644124  ... 9.36526498 7.47324465 3.89010372]]
[[4.77648724 1.36928445 4.4716588  ... 3.23664352 2.57338326 2.21028662]
 [1.17766106 2.80464798 4.3105152  ... 7.96238366 8.17743697 6.55234338]
```

```
The time of execution of above program is : 2.020040273666382

real    0m2.093s
user    0m0.128s
sys     0m0.015s
```

NxN= 64

```
The time of execution of above program is : 2.770994186401367

real    0m2.849s
user    0m0.580s
sys     0m0.018s
```

NxN= 128

```
The time of execution of above program is : 7.08700156211853

real    0m7.165s
user    0m4.448s
sys     0m0.029s
```

NxN=256

```
The time of execution of above program is : 39.567676305770874

real    0m39.645s
user    0m35.554s
sys     0m0.021s
```

NxN=512

```
The time of execution of above program is : 273.74853348731995

real    4m33.833s
user    4m30.728s
sys     0m0.611s
```

Summary:

Matrix Size	System Time (Sec)	CPU time(Sec)
32	0.015	0.128
64	0.018	0.58
128	0.029	4.448

256	0.021	35.554
512	0.611	4m30.728

B) Language Hooks

Enter the matrix size: **32**

The time taken for execution of only multiplication is : **0.0672607421875**
The complete time taken for execution of above program is : **1.7463581562042236**

Enter the matrix size: **64**

The time taken for execution of only multiplication is : **0.2518939971923828**
The complete time taken for execution of above program is : **2.1299591064453125**

Enter the matrix size: **128**
The time taken for execution of only multiplication is : **1.6830329895019531**
The complete time taken for execution of above program is : **5.50803804397583**

Enter the matrix size: **256**

The time taken for execution of only multiplication is : **13.303164958953857**
The complete time taken for execution of above program is : **15.361217021942139**

Enter the matrix size: **512**

The time taken for execution of only multiplication is : **108.16893005371094**
The complete time taken for execution of above program is : **Jupiter Error**

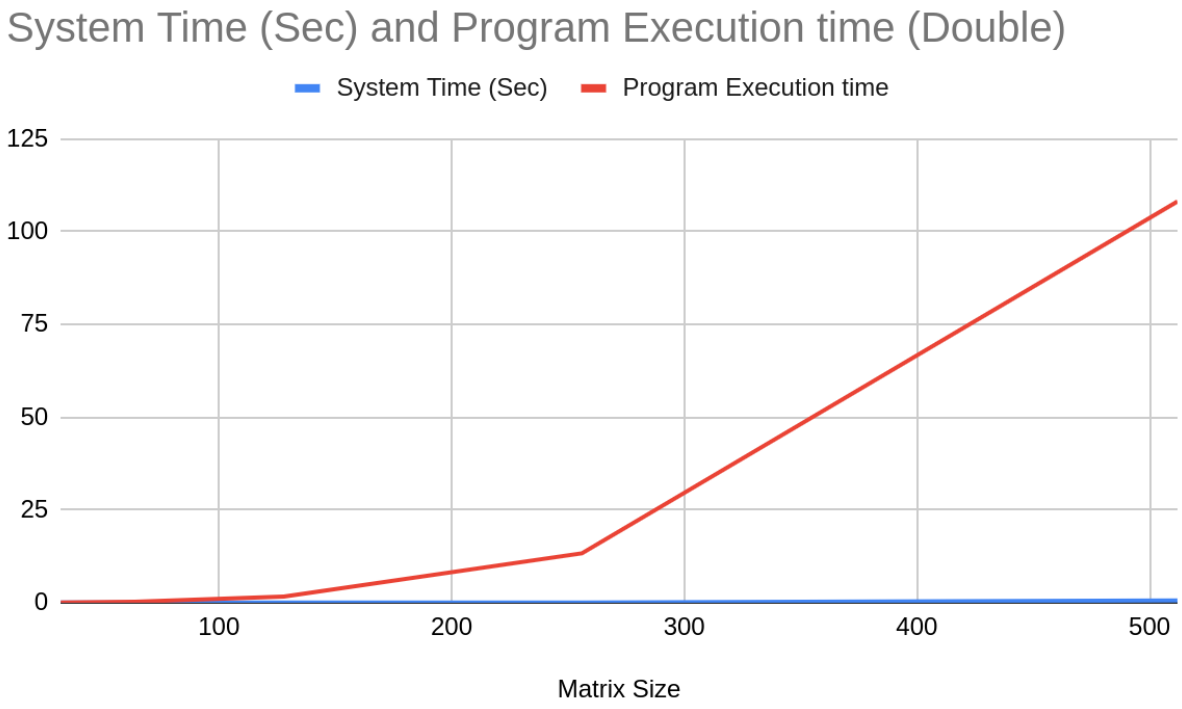
```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

2.(C)

With FLoat Numbers				
Matrix Size	**Time taken to mul**	**Complete time of program**	**Extra time for other instructions**	**Proportion**
32	0.06726074219	1.746358156	1.679097414	3.851486131
64	0.2518939972	2.129959106	1.878065109	11.82623631
128	1.68303299	5.508038044	3.825005054	30.55594344
256	13.30316496	15.36121702	2.058052063	86.60228509
512	108.1689301	Jupiter limit exceeded	-	-

Comparison of System time and Program execution time

Matrix Size	System Time (Sec)	Program Execution time
32	0.015	0.06726074219
64	0.018	0.2518939972
128	0.029	1.68303299
256	0.021	13.30316496
512	0.611	108.1689301



Bucket 1: Matrix Multiplication using C++

Double:

```
#include<bits/stdc++.h>

using namespace std;

void mulMat(vector <vector<double> > &mat1,vector <vector<double> >
&mat2) {
    int R1 =
mat1.size(),C1=mat1[0].size(),R2=mat2.size(),C2=mat2[0].size();
    vector <vector<double> > rslt(R1,vector <double>(C2,0));
     cout << "Multiplication of given two matrices is:\n" << endl;

     for (int i = 0; i < R1; i++) {
          for (int j = 0; j < C2; j++) {
               rslt[i][j] = 0;

               for (int k = 0; k < R2; k++) {
                    rslt[i][j] += mat1[i][k] * mat2[k][j];
               }

               cout << rslt[i][j] << "\t";
```

```
            }

            cout << endl;
      }
}

int main(void) {
    int N;
    cout<<"enter size: ";
    cin>>N;
    clock_t t;
    t = clock();
    int R1,C1,R2,C2;
    // cout<<"enter data R1";
     // cin>>R1;
    // cout<<"enter data C1";
    // cin>>C1;
    // cout<<"enter data R2";
    // cin>>R2;
    // cout<<"enter data C2";
    // cin>>C2;
    R1=N;
    C1=N;
    R2=N;
    C2=N;


     vector <vector<double> > mat1(R1,vector<double>(C1));
     vector <vector<double> > mat2(R2,vector<double>(C2));

     for(int i=0;i<R1;i++)
         for(int j=0;j<C1;j++)
        {
            // cout<<"enter data for M1";
            // cin>>mat1[i][j];
            mat1[i][j]=rand ();
        }


     for(int i=0;i<R2;i++)
         for(int j=0;j<C2;j++)
        {
        //      cout<<"enter data for M2";
        //    cin>>mat2[i][j];
             mat2[i][j]=rand ();
```

```
        }


     if (C1 != R2) {
          cout << "The number of columns in Matrix-1 must be equal
to the number of rows in "
                  "Matrix-2" << endl;
          cout << "Please update MACROs according to your array
dimension in #define section"
                  << endl;

         return 0;
     }
    t = clock();

     mulMat(mat1, mat2);
     t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

     printf("\nMatrix_multiplication took %f seconds to execute \n",
time_taken);



     return 0;
}
```

A) System and CPU Time

N=32

```
Matrix_multiplication took 0.000000 seconds to execute

real    0m4.590s
user    0m0.003s
sys     0m0.007s
```

N=64

```
Matrix_multiplication took 0.000000 seconds to execute

real    0m1.944s
user    0m0.008s
sys     0m0.005s
```

N=128

```
Matrix_multiplication took 0.040000 seconds to execute

real    0m2.391s
user    0m0.029s
sys     0m0.029s
```

N=256

```
Matrix_multiplication took 0.270000 seconds to execute

real    0m4.085s
user    0m0.198s
sys     0m0.087s
```

N=512

```
Matrix_multiplication took 1.940000 seconds to execute

real    0m5.560s
user    0m1.618s
sys     0m0.354s
```

Summary

Matrix Size	System Time (Sec)	CPU time(Sec)
32	0.007	0.003
64	0.005	0.008
128	0.029	0.029
256	0.087	0.198
512	0.354	1.618

b) Using Language hooks

N=32
Matrix_multiplication took 0.000000 seconds to execute
N=64
Matrix_multiplication took 0.010000 seconds to execute
N=128
Matrix_multiplication took 0.040000 seconds to execute
N=256
Matrix_multiplication took 0.270000 seconds to execute

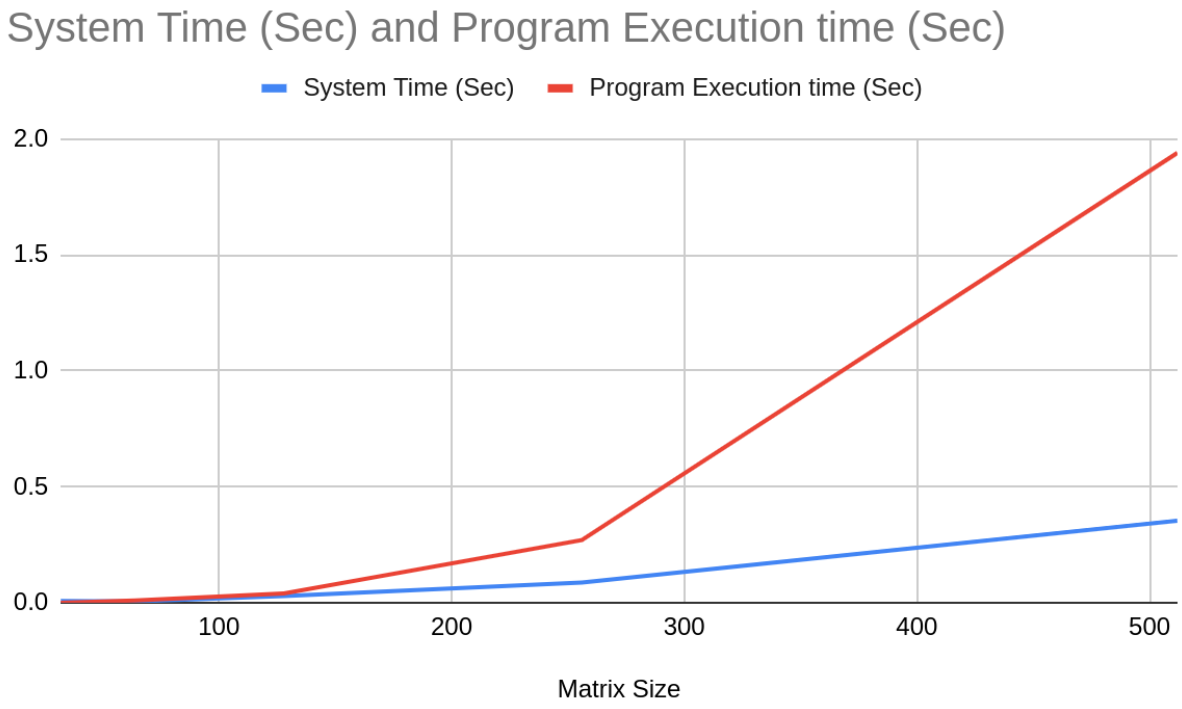
N=512
Matrix_multiplication took 1.940000 seconds to execute

Summary:

Matrix Size	Program Execution time (Sec)
32	0
64	0.01
128	0.04
256	0.27
512	1.94

C) System Time V/s Program execution time

Matrix Size	System Time (Sec)	Program Execution time (Sec)
32	0.007	0
64	0.005	0.01
128	0.029	0.04
256	0.087	0.27
512	0.354	1.94



2.Matrix Multiplication using C++

Integer:

N=32

```
Mat mul 0.000000 seconds to execute

real    0m2.582s
user    0m0.005s
sys     0m0.006s
```

N=64

```
Mat mul 0.010000 seconds to execute

real    0m2.527s
user    0m0.013s
sys     0m0.005s
[gandhi@workstation ~]$
```

N=128

```
Mat mul 0.040000 seconds to execute

real    0m3.838s
user    0m0.020s
sys     0m0.025s
```

N=256

```
Mat mul 0.250000 seconds to execute

real    0m3.129s
user    0m0.151s
sys     0m0.101s
```

N=512

```
Mat mul 1.770000 seconds to execute

real    0m4.325s
user    0m1.425s
sys     0m0.366s
[gandhi@workstation ~]$
```

Summary:

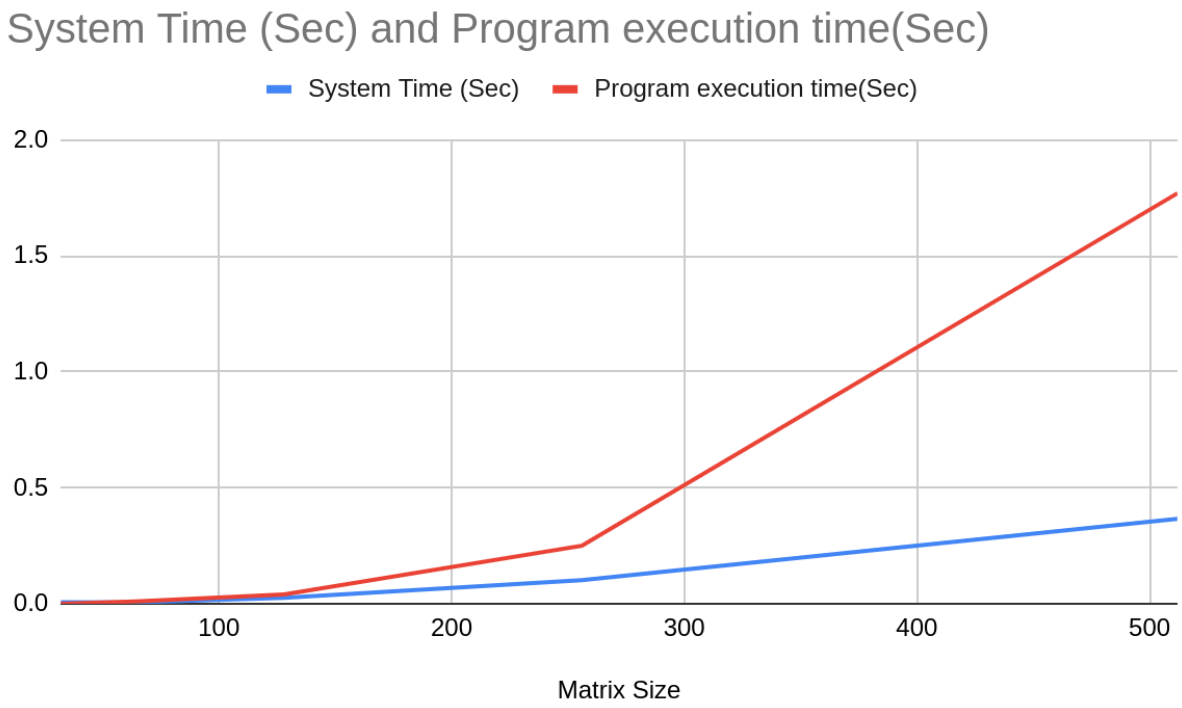
Matrix Size	System Time (Sec)	CPU time(Sec)
32	0.006	0.005
64	0.005	0.013
128	0.025	0.02
256	0.101	0.151
512	0.366	1.425

B) Program Execution time

Matrix Size	Program execution time(Sec)
32	0
64	0.01
128	0.04
256	0.25

512	1.77

C) Comparision of both Program and System execution time



Conclusion:

Compared to Bucket1 using Python, Bucket2 using Cpp ran faster.

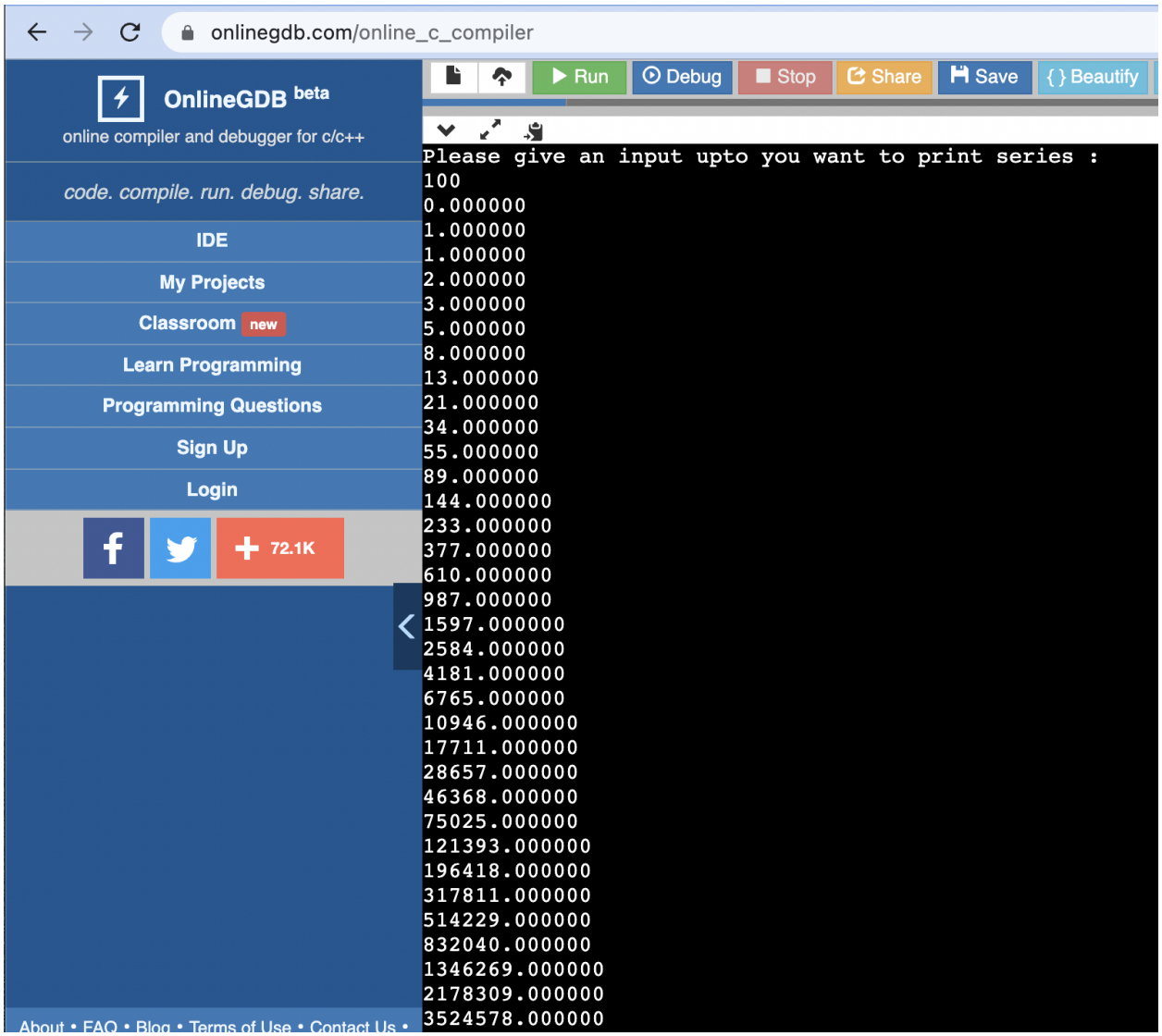
Question 1.

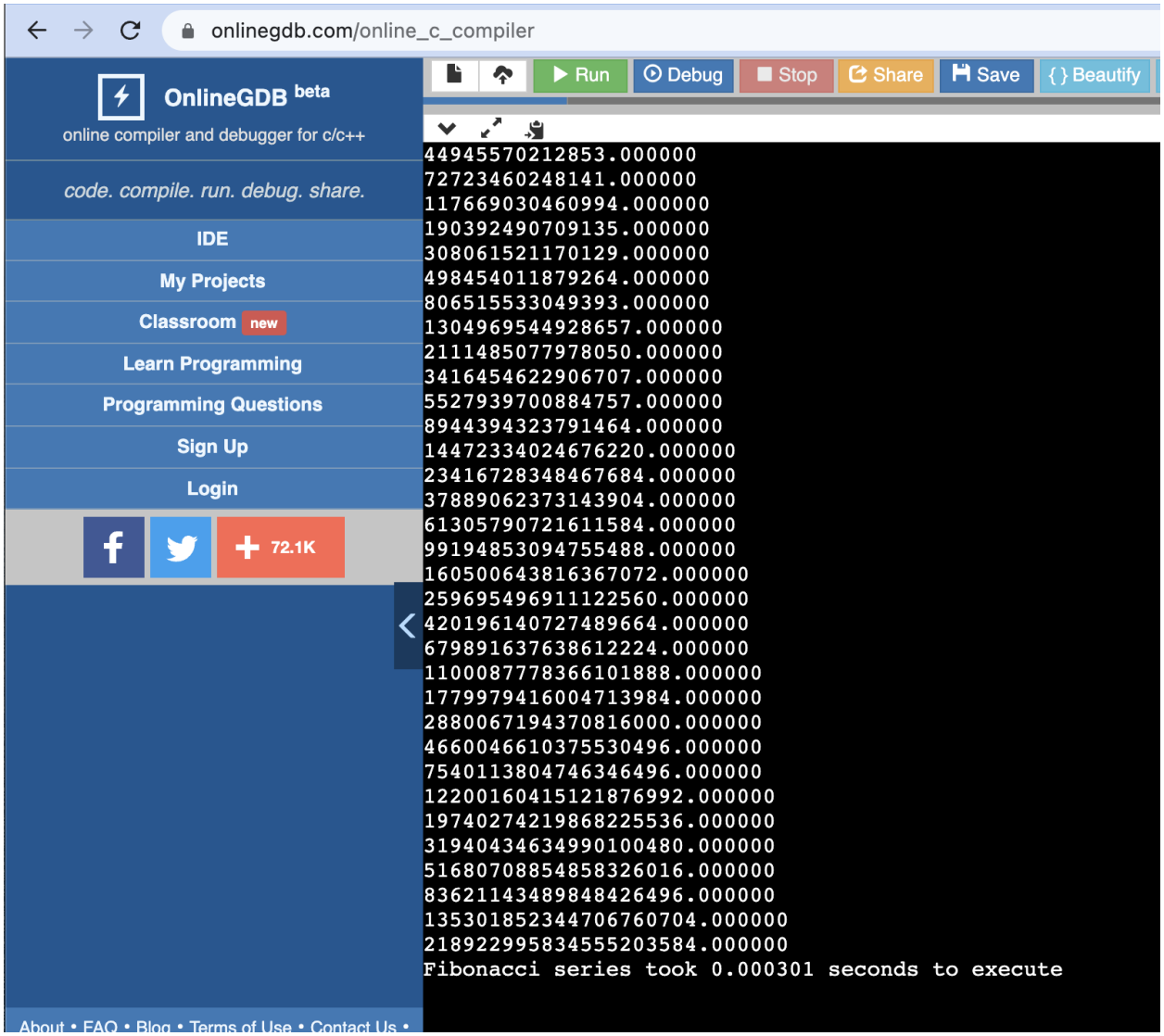
b) Recursion using Loop:

I have used clock function from time.h for calculating program execution time

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
int fib(int n)
{
  double num1 = 0, num2 = 1, result;
  int i;

  for (i = 0; i < n; i++)
  {
    if (i <= 1)
      result = i;
    else
    {
      result = num1 + num2;
      num1 = num2;
      num2 = result;
    }
    printf("%f\n", result);
  }
  return 0;
}

int main()
{ int n;
    printf("Please give an input upto you want to print series :
\n");
   scanf("%d", &n);
   //printf("Fibonacci Series is:\n");
    clock_t t;
    t = clock();
    fib(n);
    t = clock() - t;
```

```
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in
seconds

    printf("Fibonacci series took %f seconds to execute \n",
time_taken);
    return 0;
}
```





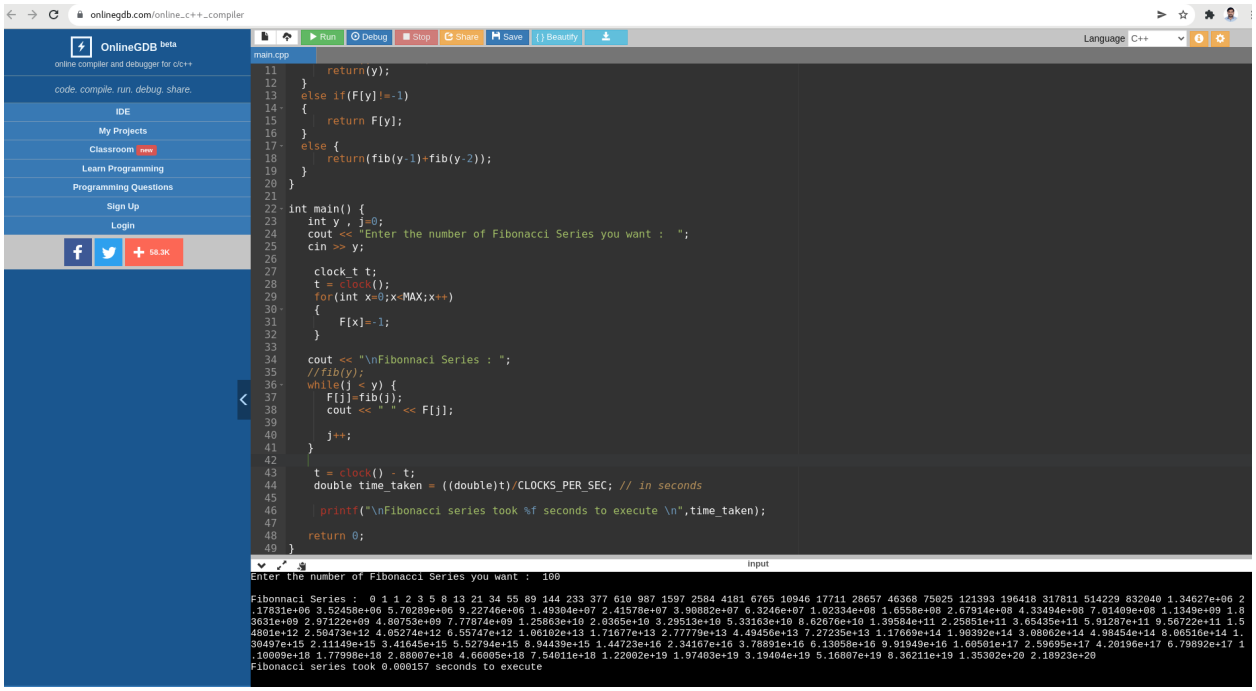
The time taken for overall execution of the program is 0.000301 seconds.

C. Fib_Memorization

```
#include <iostream>
#include <time.h>
#include <cstdio>
#define MAX 100
long double F[MAX];

//#define TIME_UTC
using namespace std;
long double fib(int y) {
```

```
  if((y==1)||(y==0)) {
      return(y);
  }
  else if(F[y]!=-1)
  {
      return F[y];
  }
  else {
      return(fib(y-1)+fib(y-2));
  }
}

int main() {
   int y , j=0;
   cout << "Enter the number of Fibonacci Series you want :
";
   cin >> y;

    clock_t t;
    t = clock();
    for(int x=0;x<MAX;x++)
    {
        F[x]=-1;
    }

   cout << "\nFibonnaci Series : ";
   //fib(y);
   while(j < y) {
      F[j]=fib(j);
      cout << " " << F[j];

      j++;
   }

    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in
seconds

     printf("\nFibonacci series took %f seconds to execute
\n",time_taken);
```

```
    return 0;
}
```



Program time execution is

Fibonacci series took 0.000157 seconds to execute