ES 215 : Assignment 4

Name : Prey Patel
Roll No : 20110132

Q1.

In RAW dependency, the ID stage of the next instruction can be executed after the completion of the WB stage of the previous stage. Therefore, it requires three stalls to prevent the RAW hazards.

Therefore, CPI(raw) = Execution instruction + no. of stalls = 1 + 3 = 4

In branch dependency, the IF stage will be executed after the completion of the last instruction. If the branch predictor makes a wrong guess then it will require two more cycles.

Therefore, CPI(bd) = 1 + 2 = 3
Normal CPI = 1

a. New CPI = RAW dependency + Branch dependency + no. hazards
= 0.3 x 4 + 0.2 x 3 + 0.5 x 1
= 2.3
Speedup = 1/ 2.3 = 0.43478

b. New CPI = Branch dependency + no. hazards
= 0.4 x 3 + 0.6 x 1
= 1.8
Speedup = 1/ 1.8 = 0.55556

With a branch predictor with 80% accuracy –
For right branch prediction, CPI will be one.

a. New CPI = RAW dependency + Branch dependency (right + wrong prediction) + no. hazards
= 0.3 x 4 + 0.2 (0.8 x 1 + 0.2 x 3) + 0.5 x 1
= 1.98

Speedup = 1/ 1.98 = 0.50505

b. New CPI = Branch dependency (right + wrong prediction) + no. hazards
= 0.4 (0.8 x 1 + 0.2 x 3) + 0.6 x 1
= 1.16
Speedup = 1/ 1.16 = 0.86207

Here, we can clearly see that the branch predictor increases the speedup of the system in both cases

Q2.

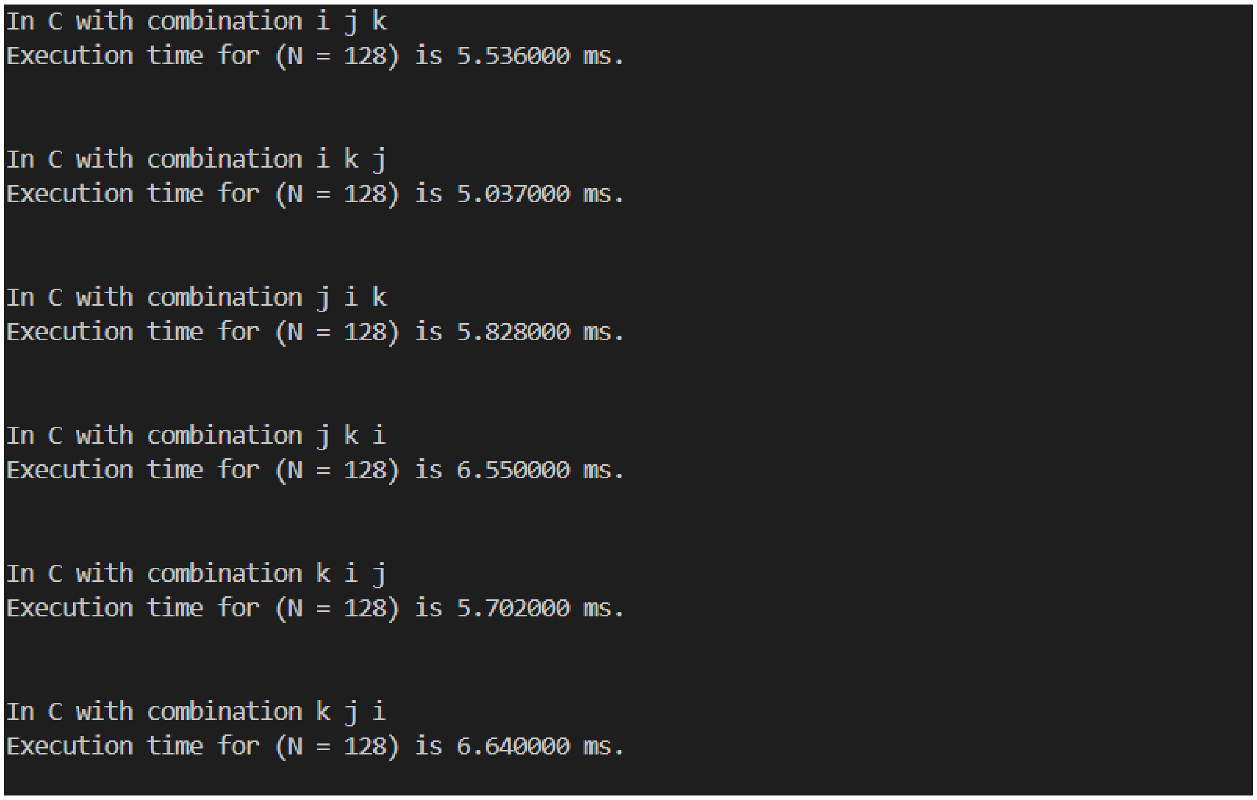
Given CPI = 1.5
Assuming that there is only branch hazard, let the total number of cycles required for branch instruction and stalls be n. Then,
CPI = branch instruction with hazard + non branch instructions
1.5 = 0.8 x 1 + 0.2 x n
n = (1.5 - 0.8) / 0.2
n = 3.5

Therefore, branch instructions takes average delay slots of 2.5
With the compiler, which can fill 85% of the delay slots (making their CPI one) -
New CPI = Branch instructions + non-branch instructions + non filled stalls
= 0.2 x 1 + 0.8 x 1 + 0.2 x (1 - 0.85) x 2.5
= 1.075

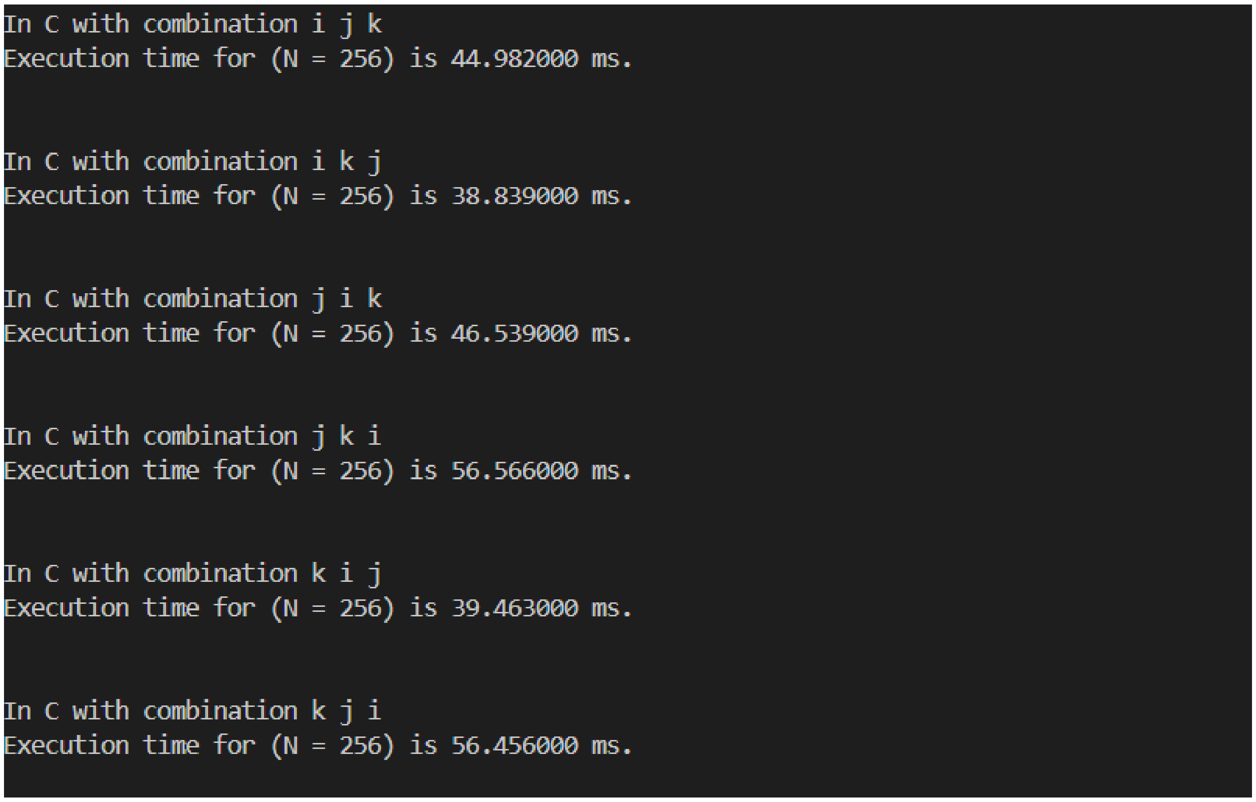
The compiler increases the performance of the program by reducing the CPI by filling the delay slots.

Q3.

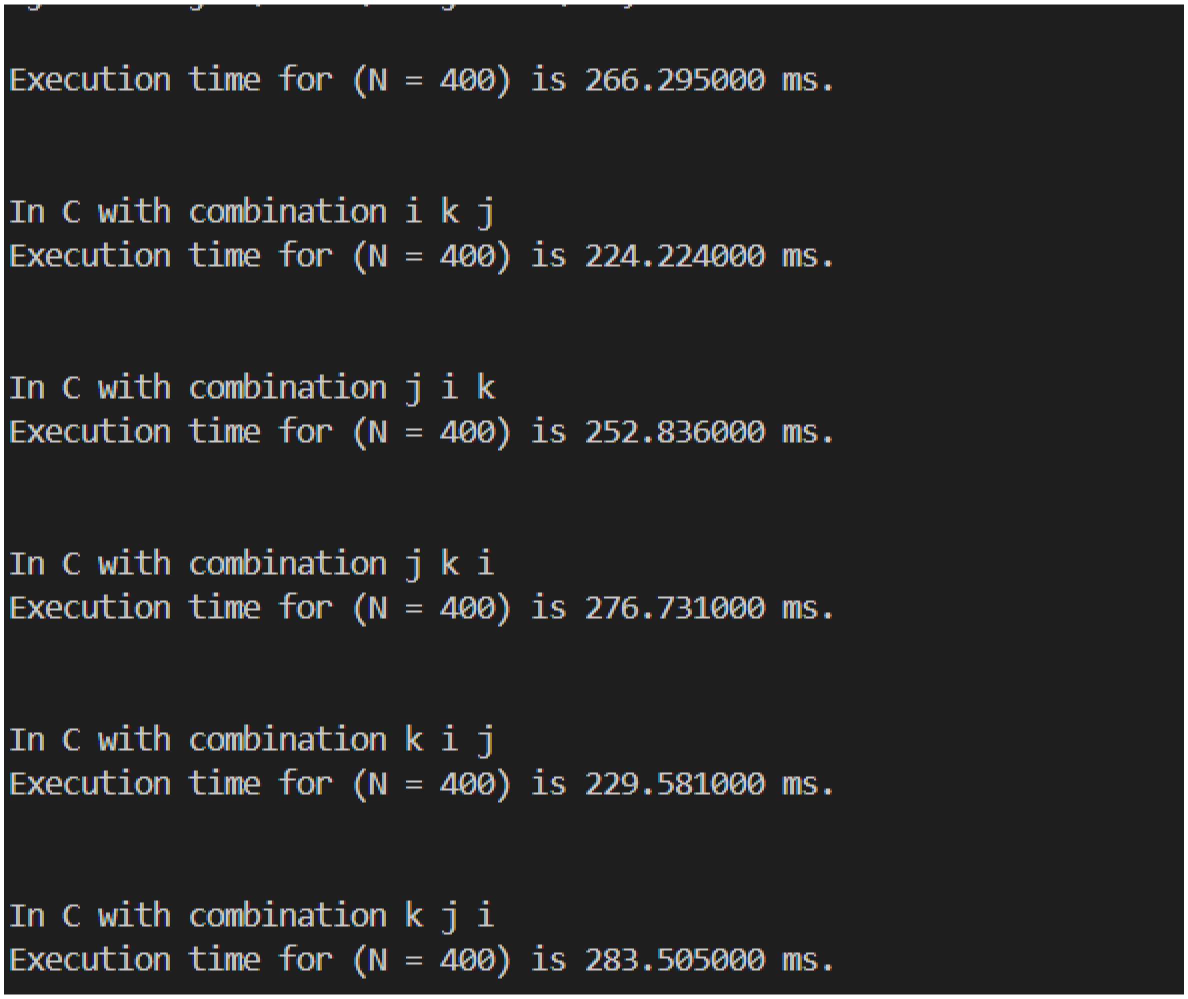
a.



```
In C with combination i j k
Execution time for (N = 128) is 5.536000 ms.


In C with combination i k j
Execution time for (N = 128) is 5.037000 ms.


In C with combination j i k
Execution time for (N = 128) is 5.828000 ms.


In C with combination j k i
Execution time for (N = 128) is 6.550000 ms.


In C with combination k i j
Execution time for (N = 128) is 5.702000 ms.


In C with combination k j i
Execution time for (N = 128) is 6.640000 ms.
```



```
In C with combination i j k
Execution time for (N = 256) is 44.982000 ms.


In C with combination i k j
Execution time for (N = 256) is 38.839000 ms.


In C with combination j i k
Execution time for (N = 256) is 46.539000 ms.


In C with combination j k i
Execution time for (N = 256) is 56.566000 ms.


In C with combination k i j
Execution time for (N = 256) is 39.463000 ms.


In C with combination k j i
Execution time for (N = 256) is 56.456000 ms.
```

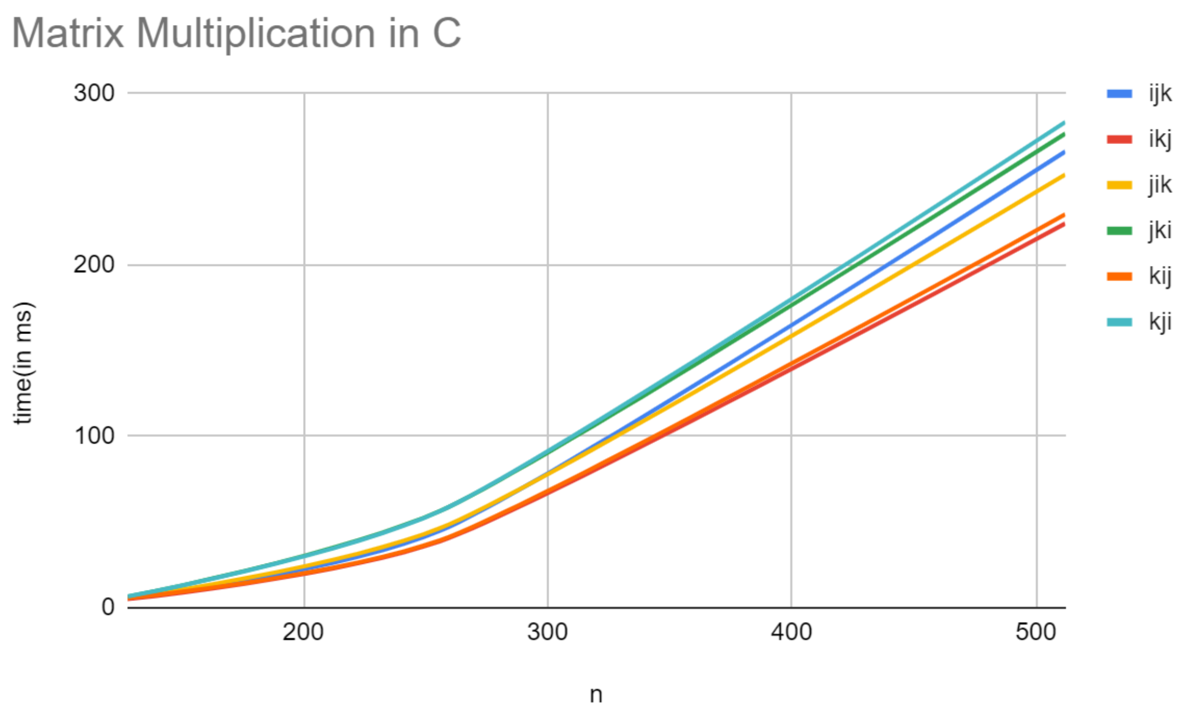


```
Execution time for (N = 400) is 266.295000 ms.


In C with combination i k j
Execution time for (N = 400) is 224.224000 ms.


In C with combination j i k
Execution time for (N = 400) is 252.836000 ms.


In C with combination j k i
Execution time for (N = 400) is 276.731000 ms.


In C with combination k i j
Execution time for (N = 400) is 229.581000 ms.


In C with combination k j i
Execution time for (N = 400) is 283.505000 ms.
```

The Matrix Multiplication programme is of order (n^3). We can see that on changing the combinations of loop interchange i, j and k the execution time varies as the address of the data accessed by the CPU changes. In the Matrix Multiplication programme, the ikj combination has the lowest execution time because the data in consecutive instructions are likely to have the same address. Hence, for this programme, ikj is the best combination as it gives the best performance.

b.

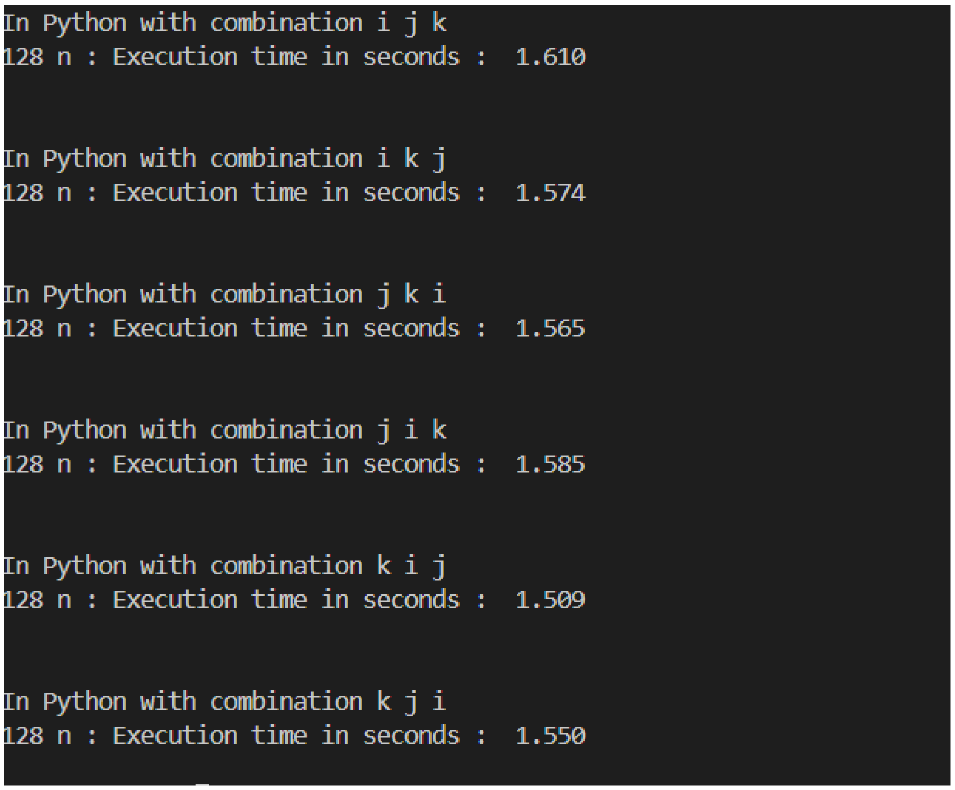
In C						
n	ijk	ikj	jik	jki	kij	kji
128	5.536	5.037	5.828	6.55	5.702	6.64
256	44.982	38.839	46.539	56.566	39.463	56.456
512	266.295	224.224	252.836	276.731	229.581	283.505



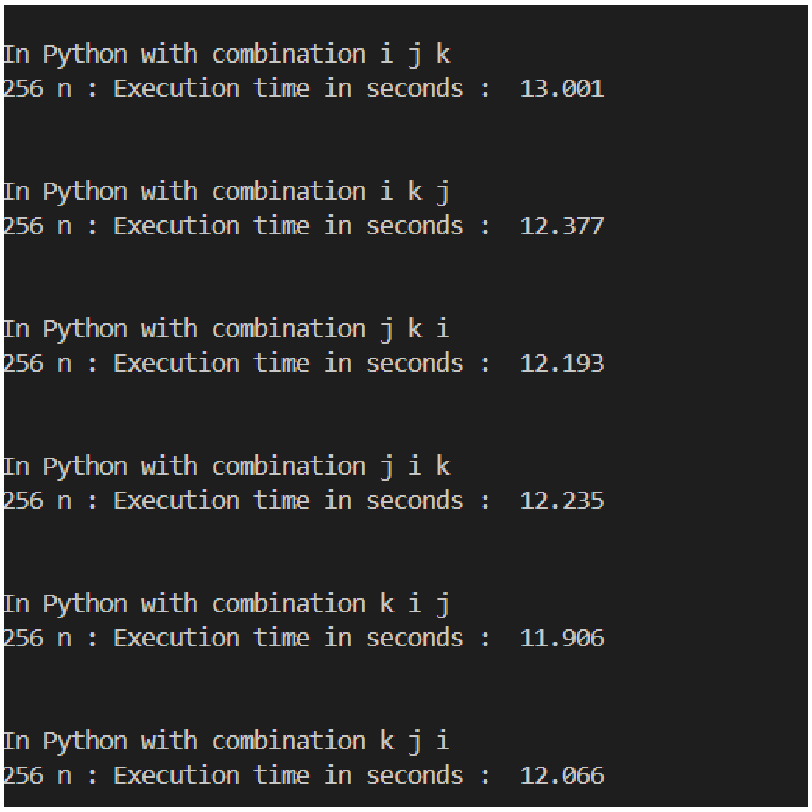
We can clearly observe that as the value of N doubles the execution time approximately becomes 8 times this shows that the Matrix Multiplication programme is of order n^3. Also for the kij combination, the execution time for N = 128 is higher than other combinations but for N = 256, and 400, the execution time is lesser than most of the other combinations. This concludes that the combination that suits best at a particular value of N is not necessary that it will also hold true for other values of N.

Q4.

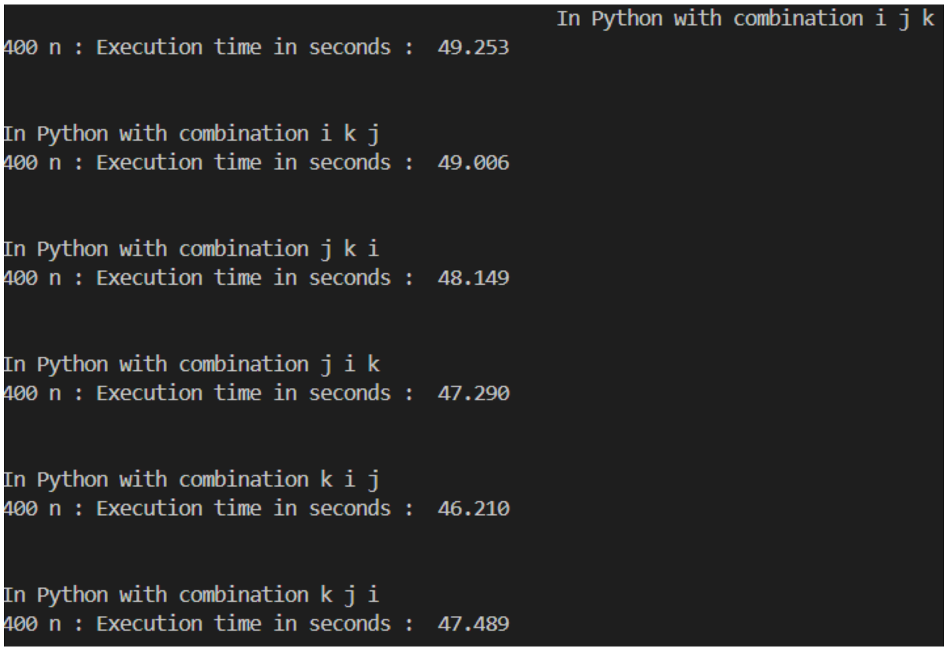
a.



```
In Python with combination i j k
128 n : Execution time in seconds :  1.610


In Python with combination i k j
128 n : Execution time in seconds :  1.574


In Python with combination j k i
128 n : Execution time in seconds :  1.565


In Python with combination j i k
128 n : Execution time in seconds :  1.585


In Python with combination k i j
128 n : Execution time in seconds :  1.509


In Python with combination k j i
128 n : Execution time in seconds :  1.550
```



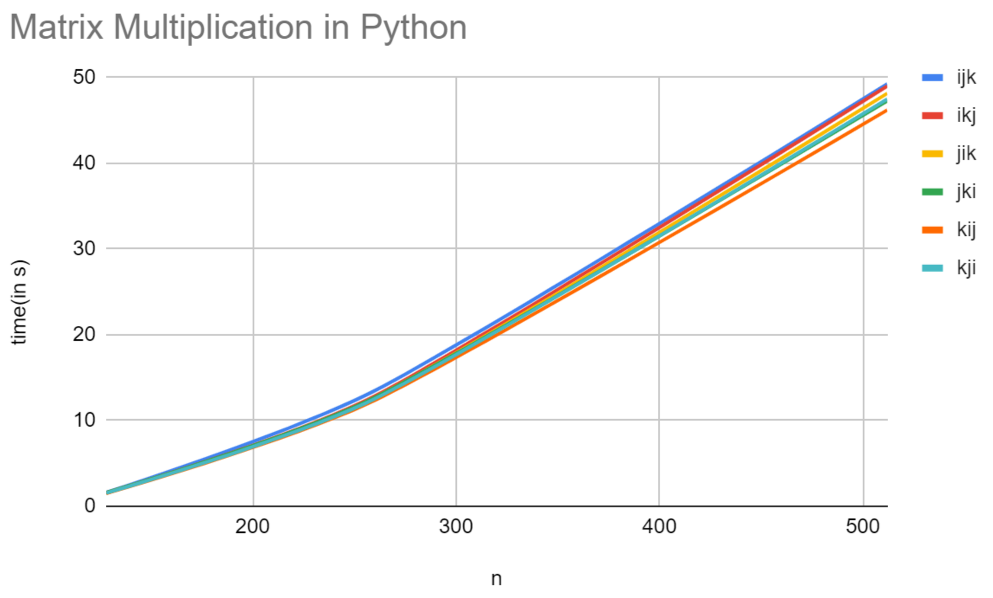
```
In Python with combination i j k
256 n : Execution time in seconds :  13.001


In Python with combination i k j
256 n : Execution time in seconds :  12.377


In Python with combination j k i
256 n : Execution time in seconds :  12.193


In Python with combination j i k
256 n : Execution time in seconds :  12.235


In Python with combination k i j
256 n : Execution time in seconds :  11.906


In Python with combination k j i
256 n : Execution time in seconds :  12.066
```



```
                                               In Python with combination i j k
400 n : Execution time in seconds :  49.253


In Python with combination i k j
400 n : Execution time in seconds :  49.006


In Python with combination j k i
400 n : Execution time in seconds :  48.149


In Python with combination j i k
400 n : Execution time in seconds :  47.290


In Python with combination k i j
400 n : Execution time in seconds :  46.210


In Python with combination k j i
400 n : Execution time in seconds :  47.489
```

b.

In Python						
n	ijk	ikj	jik	jki	kij	kji
128	1.61	1.574	1.565	1.585	1.509	1.55
256	13.001	12.377	12.193	12.235	11.906	12.066
512	49.253	49.006	48.149	47.29	46.21	47.489



In python, it takes much more time in running loops, hence we can see a great difference in execution time as compared to C programming language. We can also see that relatively there is very little difference in execution time of the various combinations. This is because the majority of execution time is taken in running the loop instruction, not in accessing the data.