Assignment-4: ES215 (https://github.com/pps-19012/COA/tree/main/Assignment-4)

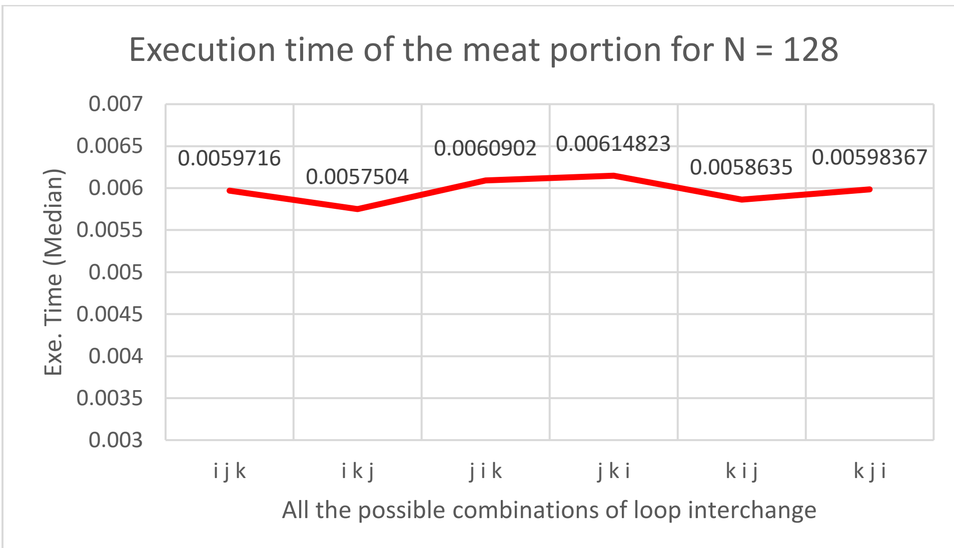
Pushpendra Pratap Singh – 20110151

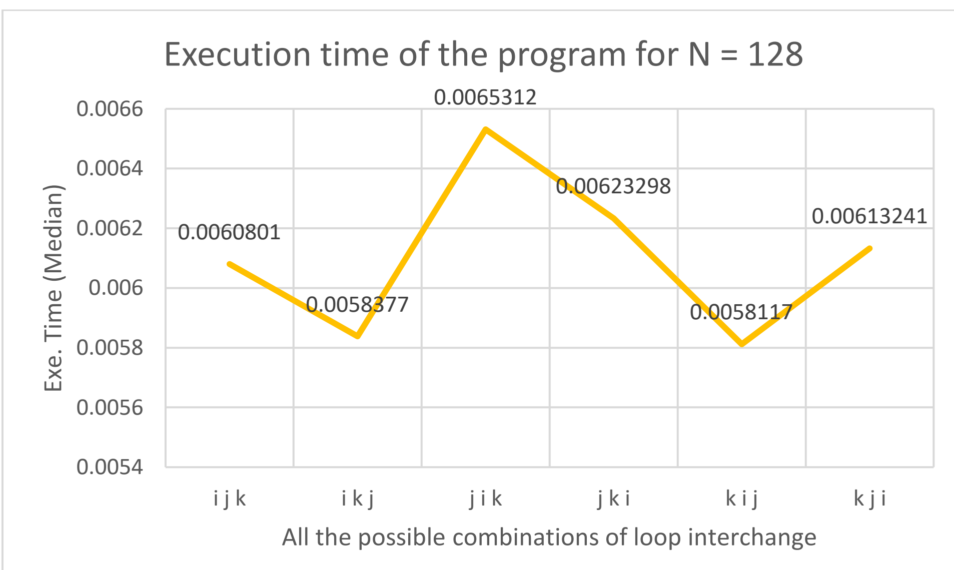
Here, I have shown the plot of execution time of both the meat portion and of the program. At first, there might seem discrepancy associated with times from the following plots. However, note that this due to the fact that I have taken median value from 3 observations only. If I would have repeated it for multiple times, this discrepancy is removed. I have added data at the end.

Also, note that as the function is not giving any output, both the times are approximately the same (not considering the unnecessary discrepancy).

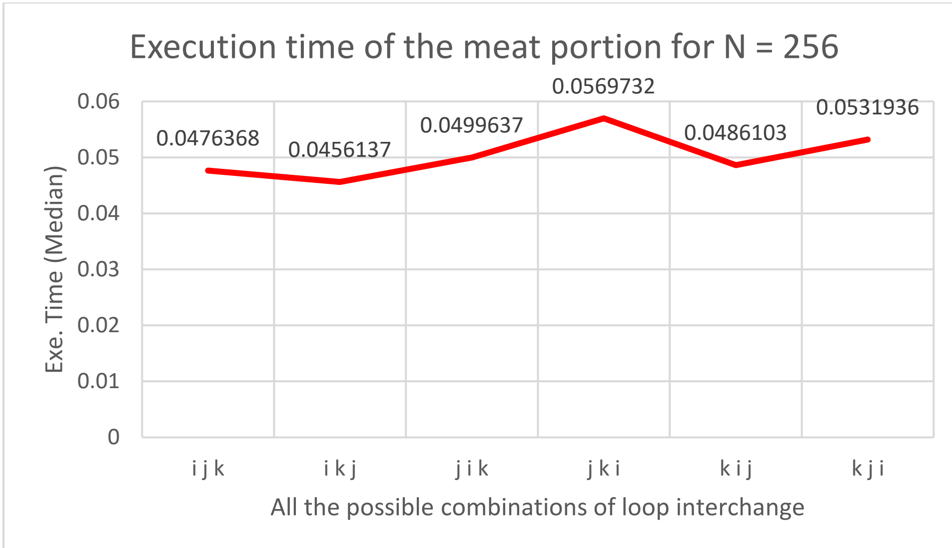
Q3. C++

(i) N = 128

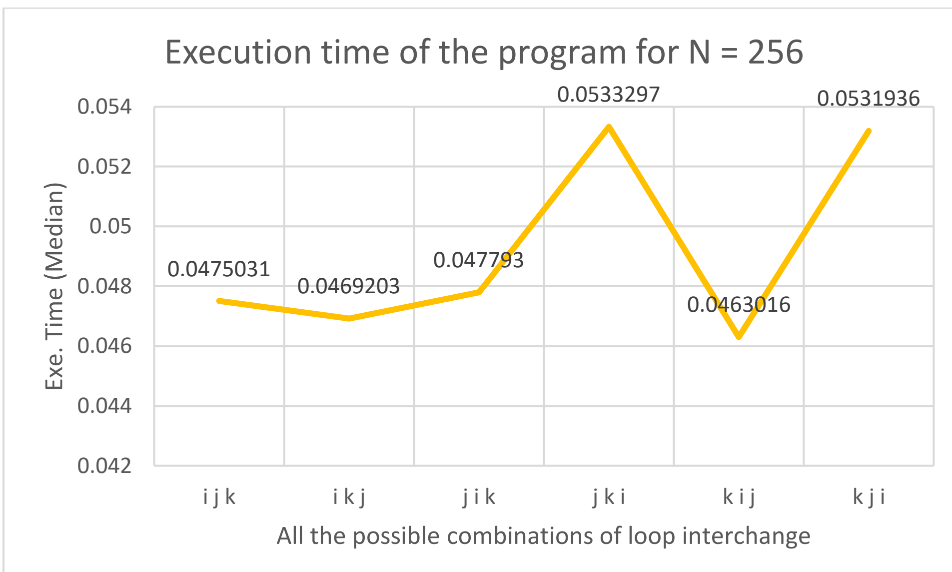




(ii) N = 256

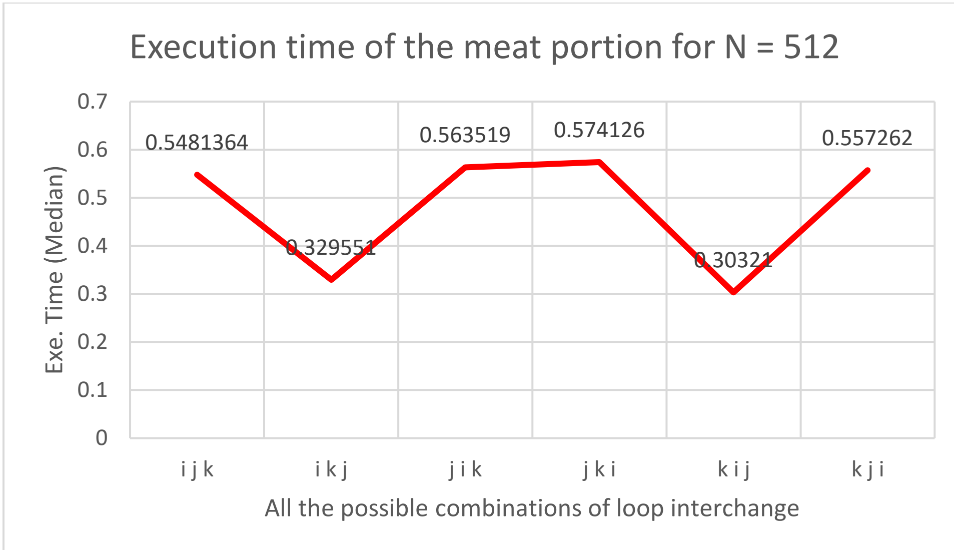


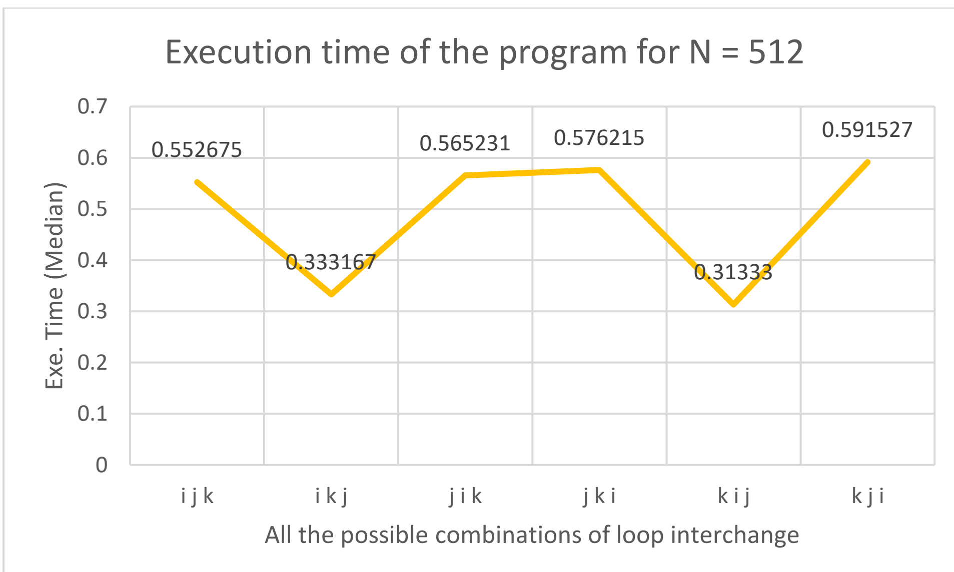
Execution time of the meat portion for N = 256
Exe. Time (Median)
0.06
0.05
0.04
0.03
0.02
0.01
0
0.0476368
0.0456137
0.0499637
0.0569732
0.0486103
0.0531936
i j k
i k j
j i k
j k i
k i j
k j i
All the possible combinations of loop interchange



Execution time of the program for N = 256
Exe. Time (Median)
0.054
0.052
0.05
0.048
0.046
0.044
0.042
0.0475031
0.0469203
0.047793
0.0533297
0.0463016
0.0531936
i j k
i k j
j i k
j k i
k i j
k j i
All the possible combinations of loop interchange

(ii) N = 512





Observations:

It is clear that as we change the loop combination, the execution times differ, i.e., it directly affects the performance. As the value of N increases, we observe a decrease in the combination *ikj* and *kij*. The execution time of the meat portion of the combination *ikj* and *kij* are significantly less as compared to the other possible combinations of loop.

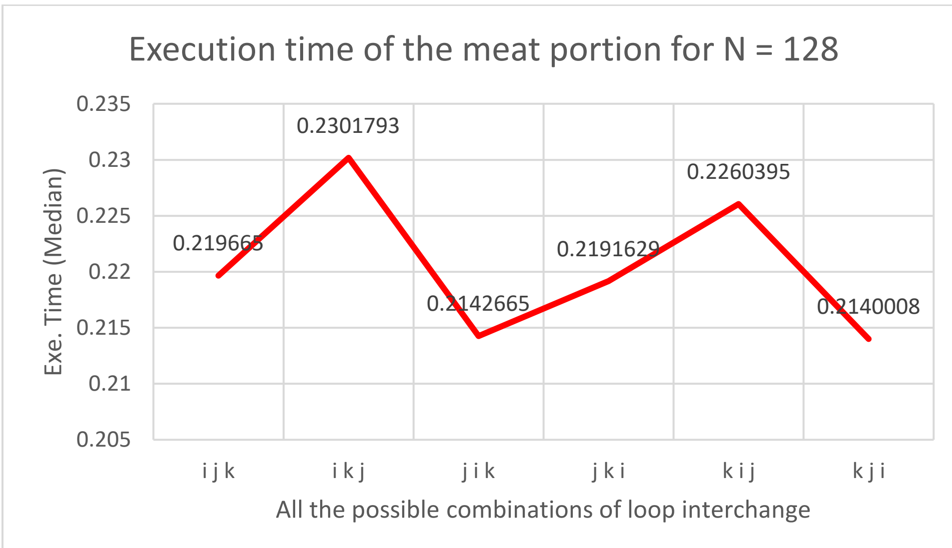
Reasons:

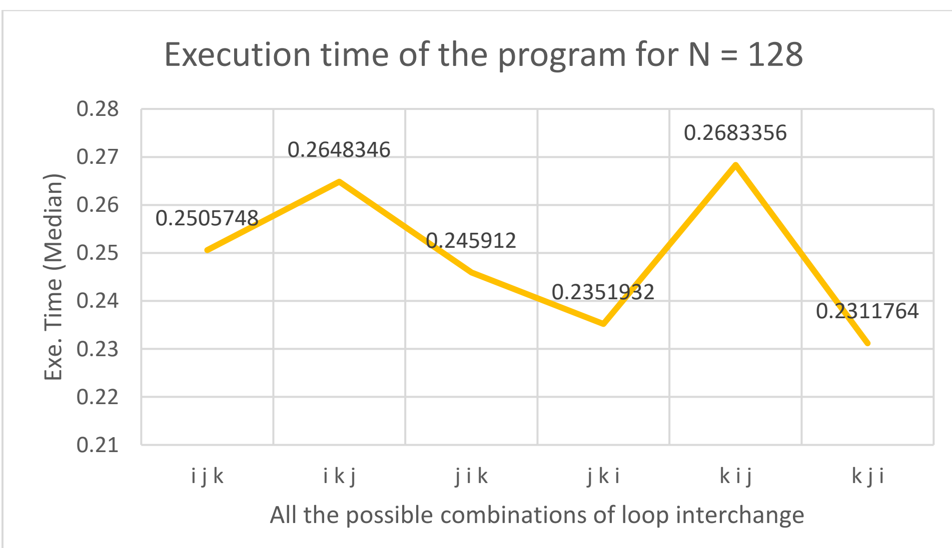
To understand the underlying concept, I searched on the net and found a stackoverflow article which addresses this topic. (https://stackoverflow.com/questions/26583536/why-is-there-a-significant-difference-in-this-c-for-loops-execution-time)

The difference arises due to cache misses. The matrix is stored in row-major order. Thus, while accessing the elements, it is easier to load a continuous segment of memory in a row. Due to this, if we access the consecutive row elements, we will not have cache miss. For example, for the *ikj* combination, we are accessing [*i*,*k*] and [*k*,*j*] which takes less time due to row-major order. Similar follows for *kij* combination.

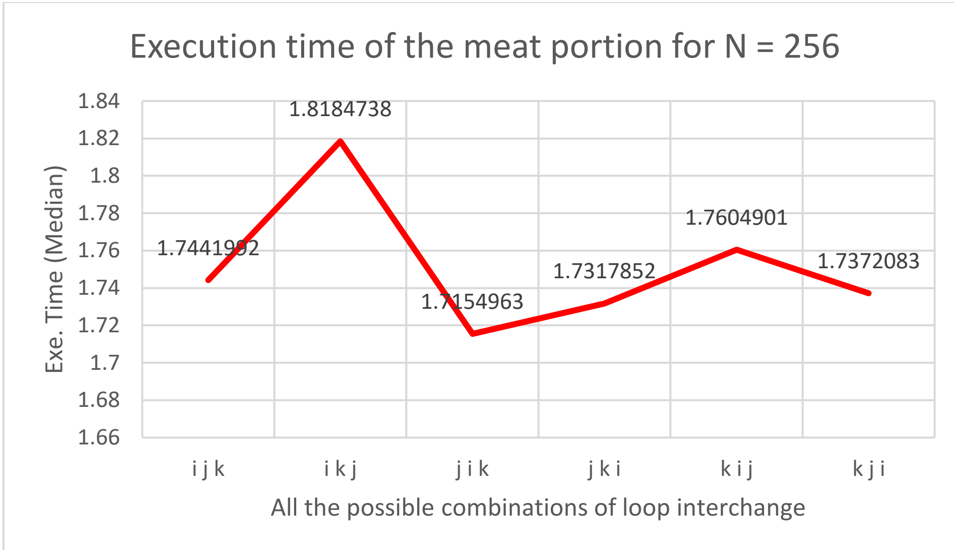
Q4. Python

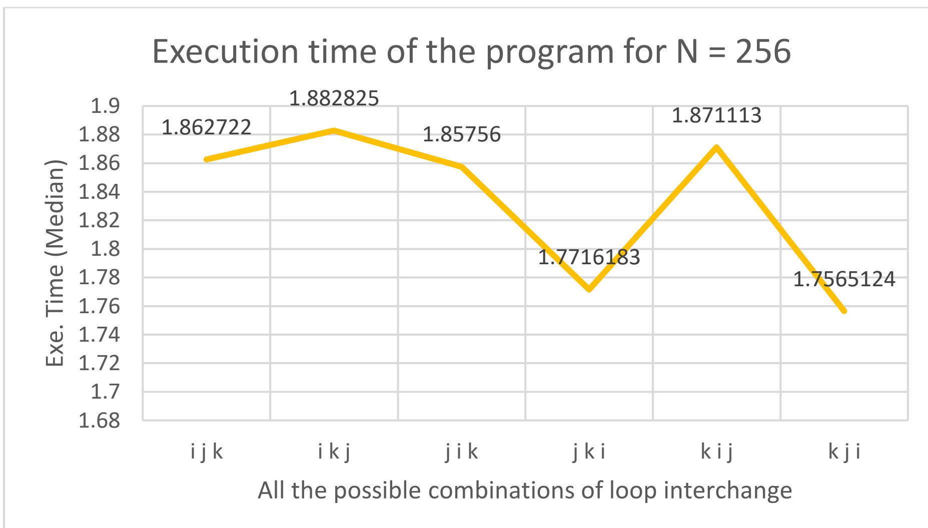
(i) N = 128



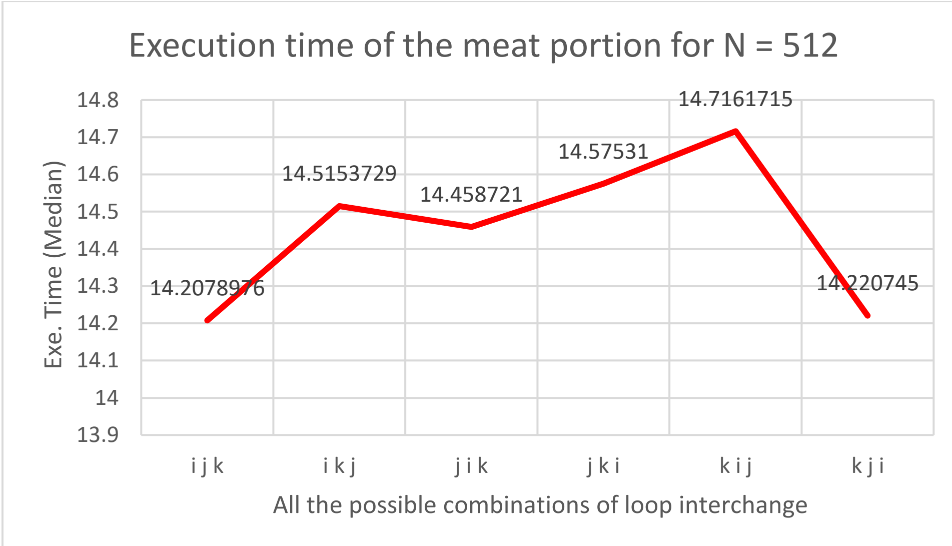


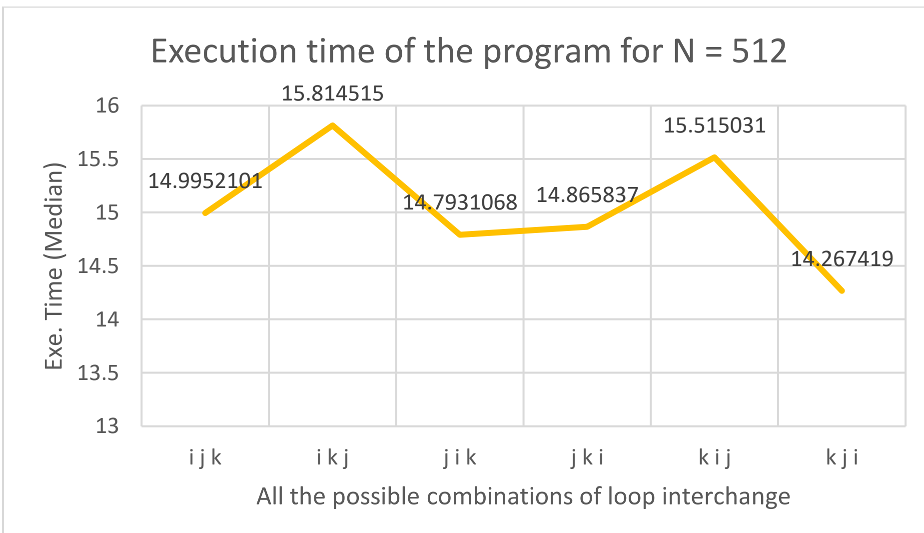
(ii) N = 256


Execution time of the meat portion for N = 256
1.84
1.82
1.8
1.78
1.76
1.74
1.72
1.7
1.68
1.66
Exe. Time (Median)
1.7441992
1.8184738
1.7154963
1.7317852
1.7604901
1.7372083
i j k
i k j
j i k
j k i
k i j
k j i
All the possible combinations of loop interchange


Execution time of the program for N = 256
1.9
1.88
1.86
1.84
1.82
1.8
1.78
1.76
1.74
1.72
1.7
1.68
Exe. Time (Median)
1.862722
1.882825
1.85756
1.7716183
1.871113
1.7565124
i j k
i k j
j i k
j k i
k i j
k j i
All the possible combinations of loop interchange

(iii) N = 512





Observations:

It is clear that as we change the loop combination, the execution times differ, i.e., it directly affects the performance. As the value of N increases, we observe an increase in the combination *ikj* and *kij*. The execution time of the meat portion of the combination *ikj* and *kij* are significantly greater as compared to the other possible combinations of loop. This is completely opposite of the trend observed in C++.

Reasons:

The difference in execution time arises due to cache misses. Here, unlike C++, the matrix is stored in column-major order. Thus, while accessing the elements, it is easier to load a continuous segment of memory in a column. Due to this, if we access the consecutive column elements, we will not have cache miss. For example, for the *jik* combination, we are accessing [*j*,*i*] and [*i*,*k*] which takes less time due to colum-major order. Similar follows for *jki* combination. Also, due to the same reason, *ikj* and *kij* (which is in row-major order) takes greater time.

Execution time values:

- C++ (Execution time of the meat portion)

N = 128	First	Second	Third	Median
i j k	0.00599314	0.00586231	0.0059716	0.0059716
i k j	0.0059639	0.0055892	0.0057504	0.0057504
j i k	0.0057839	0.0060902	0.0061046	0.0060902
j k i	0.0062628	0.00560319	0.00614823	0.00614823
k i j	0.0058405	0.0063084	0.0058635	0.0058635
k j i	0.0061902	0.00595416	0.00598367	0.00598367
N = 256	First	Second	Third	Median
i j k	0.0464956	0.0476368	0.0487289	0.0476368
i k j	0.048618	0.0456137	0.0443029	0.0456137
j i k	0.0507181	0.0499637	0.047793	0.0499637
j k i	0.0585628	0.0569732	0.0535723	0.0569732
k i j	0.0469702	0.0500291	0.0486103	0.0486103
k j i	0.0534527	0.0520448	0.0531936	0.0531936
N = 512	First	Second	Third	Median
i j k	0.5438171	0.5489625	0.5481364	0.5481364
i k j	0.316523	0.329551	0.333162	0.329551
j i k	0.563519	0.550124	0.581625	0.563519
j k i	0.54919	0.594007	0.574126	0.574126
k i j	0.30321	0.298562	0.3381	0.30321
k j i	0.521849	0.557262	0.603354	0.557262

- C++ (Execution time of the program)

N = 128	First	Second	Third	Median
i j k	0.0059196	0.006166	0.0060801	0.0060801
i k j	0.0058377	0.0058297	0.0058381	0.0058377
j i k	0.0060042	0.0065312	0.0069922	0.0065312
j k i	0.00623187	0.00652319	0.00623298	0.00623298
k i j	0.0058117	0.0061084	0.0051359	0.0058117
k j i	0.00613241	0.00595416	0.0067638	0.00613241
N = 256	First	Second	Third	Median
i j k	0.0475031	0.0435968	0.0506189	0.0475031
i k j	0.049613	0.0455367	0.0469203	0.0469203
j i k	0.0507181	0.0431399	0.047793	0.047793
j k i	0.05955628	0.0533297	0.0532754	0.0533297
k i j	0.0420796	0.0500291	0.0463016	0.0463016
k j i	0.0525472	0.0538441	0.0531936	0.0531936
N = 512	First	Second	Third	Median
i j k	0.5338171	0.552675	0.5646318	0.552675
i k j	0.333167	0.328451	0.334162	0.333167
j i k	0.572519	0.565231	0.550818	0.565231
j k i	0.589193	0.574459	0.576215	0.576215
k i j	0.31333	0.295562	0.3182	0.31333
k j i	0.583855	0.591527	0.62353	0.591527

- Python (Execution time of the meat portion)

N = 128	First	Second	Third	Median
i j k	0.219665	0.2160399	0.2356891	0.219665
i k j	0.2301793	0.2300405	0.2365	0.2301793
j i k	0.2099235	0.2142665	0.2378125	0.2142665
j k i	0.2497756	0.2191629	0.2099986	0.2191629
k i j	0.2298164	0.2260395	0.2250027	0.2260395
k j i	0.2140008	0.2199983	0.2125601	0.2140008
N = 256	First	Second	Third	Median
i j k	1.7441992	1.8091807	1.7012665	1.7441992
i k j	1.8184738	1.8276201	1.7598927	1.8184738
j i k	1.7154963	1.703942	1.7511601	1.7154963
j k i	1.6498889	1.7374791	1.7317852	1.7317852
k i j	1.7393816	1.7682619	1.7604901	1.7604901
k j i	1.7372083	1.8174798	1.7083609	1.7372083
N = 512	First	Second	Third	Median
i j k	14.2078976	14.101633	14.6100323	14.2078976
i k j	14.5153729	14.7496926	14.4610257	14.5153729
j i k	14.0503129	14.6849272	14.458721	14.458721
j k i	15.2900519	14.57531	14.1302084	14.57531
k i j	14.7161715	14.5175578	14.719008	14.7161715
k j i	14.220745	14.1756689	14.3161492	14.220745

- Python (Execution time of the program)

N = 128	First	Second	Third	Median
i j k	0.2353169	0.2508575	0.2505748	0.2505748
i k j	0.2648346	0.2821211	0.2509101	0.2648346
j i k	0.2328257	0.255128	0.245912	0.245912
j k i	0.2381091	0.2351932	0.2323865	0.2351932
k i j	0.2683356	0.2508344	0.2824935	0.2683356
k j i	0.2357811	0.2311764	0.225083	0.2311764
N = 256	First	Second	Third	Median
i j k	1.862722	1.9942464	1.8079857	1.862722
i k j	1.9843485	1.882825	1.839683	1.882825
j i k	1.8674907	1.85756	1.772143	1.85756
j k i	1.7716183	1.727516	1.797096	1.7716183
k i j	1.876876	1.871113	1.867069	1.871113
k j i	1.6950764	1.7565124	1.7580783	1.7565124
N = 512	First	Second	Third	Median
i j k	15.5391721	14.281212	14.9952101	14.9952101
i k j	15.1458547	15.814515	16.2348439	15.814515
j i k	15.112717	14.7931068	14.4974734	14.7931068
j k i	14.978901	14.267244	14.865837	14.865837
k i j	15.515031	14.7758066	15.5207751	15.515031
k j i	14.1012411	14.267419	15.5256967	14.267419