- **Problem 1**

Continuity table for the inner loop

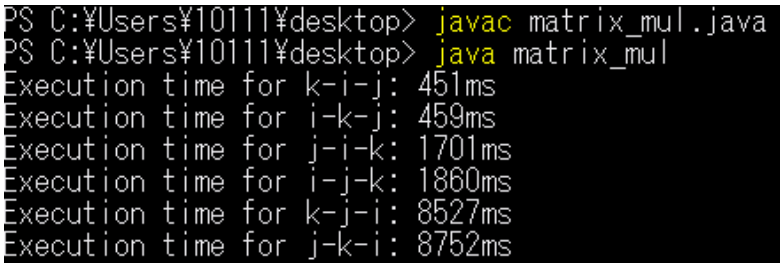
Order	c[i][j]	a[i][k]	b[k][j]
i-j-k	Not change	Continuous	Discontinuous
i-k-j	Continuous	Not change	Continuous
j-i-k	Not change	Continuous	Discontinuous
j-k-i	Discontinuous	Discontinuous	Not change
k-i-j	Continuous	Not change	Continuous
k-j-i	Discontinuous	Discontinuous	Not change

According to the above continuity table, since we know that continuous retrieving will be much faster compared to discontinuous retrieving because it's more likely to find the element we want to retrieve in the cache (i.e. cache hit), my guess is as follows:

<------slowest----------fastest------->

j-k-i ≈ k-j-i i-j-k ≈ j-i-k i-k-j ≈ k-i-j

After running the "matrix_mul.java" codes, the execution results are as follows:



- **Problem 2**

For C/C++/Java, the elements in an array are stored as sequential and contiguous blocks in the memory row by row. So, when we do matrix multiplication, large chunks of those blocks surrounding the current position will be loaded from the memory to cache when they are firstly accessed. If we access the elements of this array in a sequential manner, they will be retrieved very quickly (i.e. more cache hits) since the next element that we want to retrieve is very likely to be in the cache already.

However, for Python, the list is implemented as an array of pointers. The elements in the list are actually scattered in the memory which are not contiguous. So, when we do matrix multiplication, though the blocks surrounding the current positions are loaded into cache, it is very likely that we will have a lot of cache misses because the elements we want to retrieve are not sequentially stored in the memory, which means they might be at a very far position from the current element we retrieved (i.e. not loaded in the cache). So even when we change the order to retrieve elements sequentially, the cache does not contribute much to the final performance.