Problem 1

Language: C language

Size: n = 1024

Result:

- Execution time:

  i-j-k: 24.77s

  i-k-j: 3.05s

  j-i-k: 12.41s

  j-k-i: 25.56s

  k-i-j: 3.09s

  k-j-i: 23.34s

- Execution speed:

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

Explanation:

Considering the inner loop of c[i][j] += a[i][k] * b[k][j]:

- Under order i-k-j: c[i][j] and b[k][j] are continuous, a[i][k] doesn't change.
- Under order k-i-j: c[i][j] and b[k][j] are continuous, a[i][k] doesn't change.
- Under order i-j-k: b[k][j] is not continuous
- Under order j-i-k: b[k][j] is not continuous
- Under order j-k-i: c[i][j] and a[i][k] are not continuous
- Under order k-j-i: c[i][j] and a[i][k] are not continuous

Using order i-k-j and k-i-j is the fastest, since it is more likely to find the continuous elements in cache, which is much faster than finding them in memory.

Problem 2

In C/C++/Java, a matrix/array is stored inside the memory as a whole block. During execution, a smaller block around current position will be stored in cache. Therefore, if we access the elements in a sequential and continuous way, they can be retrieved quickly, since they are stored in cache.

In Python, a list is actually a group of pointers, which means elements of a list are NOT continuously stored in the memory as a whole block. During execution, although a smaller block around current position is still stored in cache, it probably does not contain the elements we want to retrieve. We still need to access the elements we want in the memory, which is much slower. No matter which order we use, the cache does not make sense in the performance.