Challenge 1: Matrix Multiplication

Consider the following pseudocode algorithm, which is naive implementation of matrix multiplication. For simplicity we assume that the matrices are square.

```
int[][] matrixMultiply(int[][] A, int[][] B) {
     int n = A.length;

     // initialise C
     int[][] C = new int[n][n];

     for (int i = 0; i < n; i++) {
          for (int k = 0; k < n; k++) {
                    for (int j = 0; j < n; j++) {
                         C[i][j] += A[i][k] * B[k][j];
                    }
          }
     }
     return C;
}
```

Tasks.

1. Provide a specification to describe the behaviour of this algorithm, and prove that it correctly implements its specification.
2. Show that matrix multiplication is associative, i.e., the order in which matrices are multiplied can be disregarded: A(BC) = (AB)C. To show this, you should write a program that performs the two different computations, and then prove that the result of the two computations is always the same.
3. [Optional, if time permits] In the literature, there exist many proposals for more efficient matrix multiplication algorithms. Strassen's algorithm was one of the first. The key idea of the algorithm is to use a recurisive algorithm that reduces the number of multiplications on submatrices (from 8 to 7), see https://en.wikipedia.org/wiki/Strassen_algorithm for an explanation. A relatively clean Java implementation (and Python and C++) can be found here: https://martin-thoma.com/strassen-algorithm-in-python-java-cpp/. Prove that the naive algorithm above has the same behaviour as Strassen's algorithm. Proving it for a

restricted case, like a 2x2 matrix should be straightforward, the challenge is to prove it for arbitrary matrices with size 2^n.