Exercise 1: Divide and Conquer \hspace{1cm} (10 Points)

Consider a board with \( n \times n \) cells with \( n = 2^k \) for a \( k \in \mathbb{N}_{\geq 1} \) (see below for an example). We have an unlimited supply of a specifically shaped tile, which covers exactly 3 cells of the board as follows:

The goal is to cover the board with tiles (which can be turned by 90, 180 and 270 degrees). We call an arrangement of tiles on the board a valid tiling, if all cells can be covered with the tile above without any overlaps and without going over the edges of the board. Assume that the input board has an arbitrary single cell that is initially covered (before the start of the algorithm). E.g. for \( n = 8 \) the board may look like this:

(a) Is there a valid tiling for every \( 2^k \times 2^k \) board \( (k \in \mathbb{N}_{\geq 1}) \) that is initially completely empty? Prove or disprove. \hspace{1cm} (2 Points)

(b) Describe a divide and conquer algorithm that computes a valid tiling on a \( n \times n \) board in \( O(n^2) \) (with \( n = 2^k, k \in \mathbb{N}_{\geq 1} \)) that has one tiled cell. Assume that placing a tile is in \( O(1) \). \hspace{1cm} (6 Points)

(c) Show the running time of \( O(n^2) \). \hspace{1cm} (2 Points)
Exercise 2: Fast Fourier Transformation (FFT) \hspace{1cm} (10 Points)

Let \( p(x) = 7x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x \). We want to compute the discrete fourier transform \( DFT_8(p) \) (where we define \( DFT_8(p) := DFT_8(a) \) given that \( a \) is the vector of coefficients of \( p \)). More specifically, we want you to visualize the steps which the FFT-algorithm performs as follows.

(a) Illustrate the \textit{divide} procedure of the algorithm. More precisely, for the \( i \)-th divide step, write down all the polynomials \( p_{ij} \) for \( j \in \{0, \ldots, 2^i - 1\} \) that you obtain from further dividing the polynomials from the previous divide step \( i-1 \) (we define \( p_{00} := p \)). \hspace{1cm} (3 Points)

(b) Illustrate the \textit{combine} procedure of the algorithm. That is, starting with the polynomials of smallest degree as base cases, compute the \( DFT_N(p_{ij}) \) bottom up with the recursive formula given in the lecture (where \( N \) is the smallest power of 2 such that \( \deg(p_{ij}) < N \)). \hspace{1cm} (7 Points)

\textbf{Hints:} The base case for a polynomial \( p = a \) of degree 0 is \( DFT_1(p) = DFT_1(a) = a \). In general, it also suffices to give the \( p_{ij}(\omega) \) for the appropriate roots of unity \( \omega \), from which \( DFT_N(p_{ij}) \) can be derived. Use \( \sqrt{\cdot} \) instead of floating point numbers if possible.