University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithms Theory

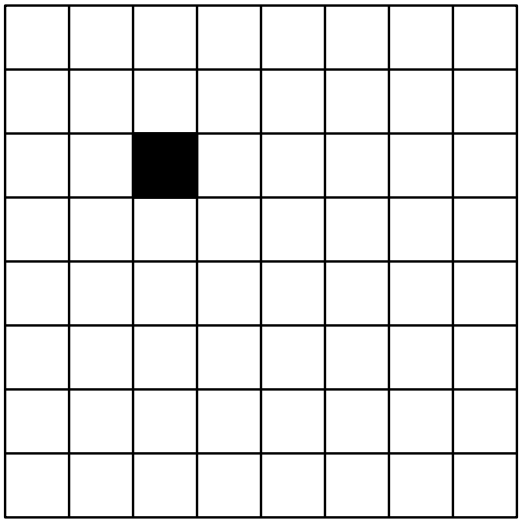
Sample Solution Exercise Sheet 2

Due: Tuesday, 17th of November 2020, 4 pm

Exercise 1: Divide and Conquer *(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. *(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 $O(1)$. *(6 Points)*

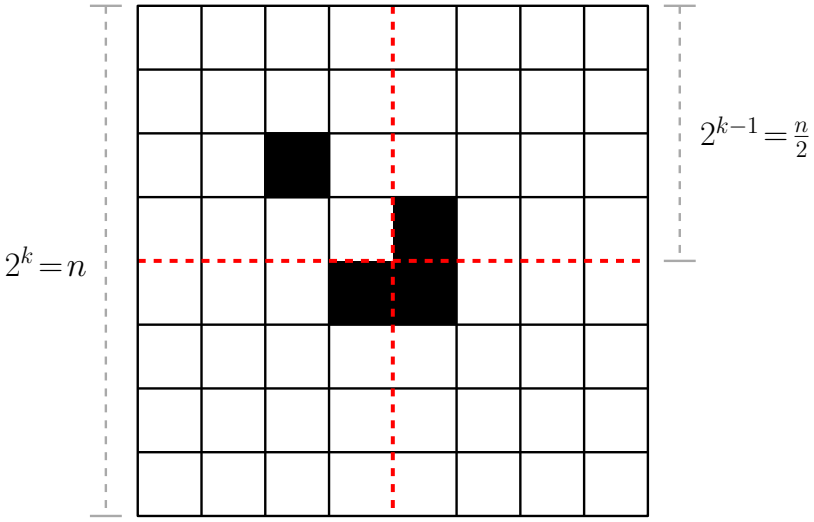
(c) Show the running time of $O(n^2)$. *(2 Points)*

Sample Solution

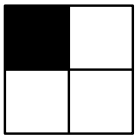
(a) No there is not. Assume there were such a valid tiling using t copies of the above tile. Then $3t = 2^{2k}$, i.e., 3 divides a power of 2, a contradiction since 2 and 3 are both prime.

(b) Our strategy is to divide a $2^k \times 2^k$ board into four $2^{k-1} \times 2^{k-1}$ smaller boards and maintain the invariant that one cell will always be tiled (using the knowledge that a valid tiling is impossible on a completely empty board). This invariant immediately gives us a valid tiling for the base case of a 2×2 board.

Divide: We have a $2^k \times 2^k$ board with $k > 1$ and one cell already tiled. We divide it into four $2^{k-1} \times 2^{k-1}$ smaller boards. We make sure that each small board has one cell that is already tiled. We do this by placing the tile such that it covers exactly one cell of each small board that does not have a tiled cell yet, as exemplified by the following figure



Base Case: We have a 2×2 board with one cell already tiled which looks like the example below (except for the orientation of the tiled cell). This can obviously be covered using one of our specifically shaped tiles and we obtain a valid tiling.



Conquer: By producing valid tilings for all 2×2 boards we obtain a valid tiling for the whole board.

(c) The run time is $T(n) = 4T(n/2) + \mathrm{O}(1)$ which solves to $\mathrm{O}(n^2)$ using the Master Theorem.

Remark: Assuming that finding the occupied cell is not for free (in the exercise we make that assumption) does actually not increase the runtime asymptotically. Assume the board is given by an $n \times n$ array filled with boolean values (where true means the cell is tiled and vice versa). Then we find the tiled cell in $O(n^2)$ once before the divide and conquer starts. In the recursive calls of the algorithm we always pass the coordinates of the cell that is already occupied as a parameter. The coordinate of the filled cell in each smaller board obtained from the recursion is clearly always known after placing the L-tile and can be passed as a parameter to further recursive calls.

Exercise 2: Fast Fourier Transformation (FFT) *(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 *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 $p_{00} := p$). *(3 Points)*

(b) Illustrate the *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 $\deg(p_{ij}) < N$). *(7 Points)*

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 ω, from which $DFT_N(p_{ij})$ can be derived. Use $\sqrt{\cdot}$ instead of floating point numbers if possible.

Sample Solution

(a)

$$
\begin{aligned}
p_{00}(x) &= 7x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x \\
p_{10}(x) &= 6x^3 + 4x^2 + 2x \\
p_{11}(x) &= 7x^3 + 5x^2 + 3x + 1 \\
p_{20}(x) &= 4x \\
p_{21}(x) &= 6x + 2 \\
p_{22}(x) &= 5x + 1 \\
p_{23}(x) &= 7x + 3 \\
p_{30}(x) &= 0 \\
p_{31}(x) &= 4 \\
p_{32}(x) &= 2 \\
p_{33}(x) &= 6 \\
p_{34}(x) &= 1 \\
p_{35}(x) &= 5 \\
p_{36}(x) &= 3 \\
p_{37}(x) &= 7
\end{aligned}
$$

(b) Base cases of the FFT algorithm (for any $x \in \mathbb{C}$):

$$
\begin{aligned}
p_{30}(x) &= DFT_1\big(p_{30}\big) = (0) \\
p_{31}(x) &= DFT_1\big(p_{31}\big) = (4) \\
p_{32}(x) &= DFT_1\big(p_{32}\big) = (2) \\
p_{33}(x) &= DFT_1\big(p_{33}\big) = (6) \\
p_{34}(x) &= DFT_1\big(p_{34}\big) = (1) \\
p_{35}(x) &= DFT_1\big(p_{35}\big) = (5) \\
p_{36}(x) &= DFT_1\big(p_{36}\big) = (3) \\
p_{37}(x) &= DFT_1\big(p_{37}\big) = (7)
\end{aligned}
$$

Bottom computation with the recursive formula:

$$
\begin{aligned}
p_{20}(\omega_2^0) &= p_{30}(\omega_1^0) + \omega_2^0 \cdot p_{31}(\omega_1^0) = 0 + 1 \cdot 4 = 4 \\
p_{20}(\omega_2^1) &= p_{30}(\omega_1^0) - \omega_2^0 \cdot p_{31}(\omega_1^0) = 0 - 1 \cdot 4 = -4 \\
\\
p_{21}(\omega_2^0) &= p_{32}(\omega_1^0) + \omega_2^0 \cdot p_{33}(\omega_1^0) = 2 + 1 \cdot 6 = 8 \\
p_{21}(\omega_2^1) &= p_{32}(\omega_1^0) - \omega_2^0 \cdot p_{33}(\omega_1^0) = 2 - 1 \cdot 6 = -4 \\
\\
p_{22}(\omega_2^0) &= p_{34}(\omega_1^0) + \omega_2^0 \cdot p_{35}(\omega_1^0) = 1 + 1 \cdot 5 = 6 \\
p_{22}(\omega_2^1) &= p_{34}(\omega_1^0) - \omega_2^0 \cdot p_{35}(\omega_1^0) = 1 - 1 \cdot 5 = -4 \\
\\
p_{23}(\omega_2^0) &= p_{36}(\omega_1^0) + \omega_2^0 \cdot p_{37}(\omega_1^0) = 3 + 1 \cdot 7 = 10 \\
p_{23}(\omega_2^1) &= p_{36}(\omega_1^0) - \omega_2^0 \cdot p_{37}(\omega_1^0) = 3 - 1 \cdot 7 = -4
\end{aligned}
$$

$$
\begin{aligned}
p_{10}(\omega_4^0) &= p_{20}(\omega_2^0) + \omega_4^0 \cdot p_{21}(\omega_2^0) = 4 + 1 \cdot 8 = 12 \\
p_{10}(\omega_4^1) &= p_{20}(\omega_2^1) + \omega_4^1 \cdot p_{21}(\omega_2^1) = -4 + i \cdot (-4) = -4 - 4i \\
p_{10}(\omega_4^2) &= p_{20}(\omega_2^0) - \omega_4^0 \cdot p_{21}(\omega_2^0) = 4 - 1 \cdot 8 = -4 \\
p_{10}(\omega_4^3) &= p_{20}(\omega_2^1) - \omega_4^1 \cdot p_{21}(\omega_2^1) = -4 - i \cdot (-4) = -4 + 4i \\
\\
p_{11}(\omega_4^0) &= p_{22}(\omega_2^0) + \omega_4^0 \cdot p_{23}(\omega_2^0) = 6 + 1 \cdot 10 = 16 \\
p_{11}(\omega_4^1) &= p_{22}(\omega_2^1) + \omega_4^1 \cdot p_{23}(\omega_2^1) = -4 + i \cdot (-4) = -4 - 4i \\
p_{11}(\omega_4^2) &= p_{22}(\omega_2^0) - \omega_4^0 \cdot p_{23}(\omega_2^0) = 6 - 1 \cdot 10 = -4 \\
p_{11}(\omega_4^3) &= p_{22}(\omega_2^1) - \omega_4^1 \cdot p_{23}(\omega_2^1) = -4 - i \cdot (-4) = -4 + 4i
\end{aligned}
$$

$$
\begin{aligned}
p_{00}(\omega_8^0) &= p_{10}(\omega_4^0) + \omega_8^0 \cdot p_{11}(\omega_4^0) = 12 + 1 \cdot 16 = 28 \\
p_{00}(\omega_8^1) &= p_{10}(\omega_4^1) + \omega_8^1 \cdot p_{11}(\omega_4^1) = -4 - 4i + \tfrac{i+1}{\sqrt{2}} \cdot (-4 - 4i) = -4 - 4i \cdot (\sqrt{2}+1) \\
p_{00}(\omega_8^2) &= p_{10}(\omega_4^2) + \omega_8^2 \cdot p_{11}(\omega_4^2) = -4 + i \cdot (-4) = -4 - 4i \\
p_{00}(\omega_8^3) &= p_{10}(\omega_4^3) + \omega_8^3 \cdot p_{11}(\omega_4^3) = -4 + 4i + \tfrac{i-1}{\sqrt{2}} \cdot (-4 + 4i) = -4 - 4i \cdot (\sqrt{2}-1) \\
p_{00}(\omega_8^4) &= p_{10}(\omega_4^0) - \omega_8^0 \cdot p_{11}(\omega_4^0) = 12 - 1 \cdot 16 = -4 \\
p_{00}(\omega_8^5) &= p_{10}(\omega_4^1) - \omega_8^1 \cdot p_{11}(\omega_4^1) = -4 - 4i - \tfrac{i+1}{\sqrt{2}} \cdot (-4 - 4i) = -4 + 4i \cdot (\sqrt{2}-1) \\
p_{00}(\omega_8^6) &= p_{10}(\omega_4^2) - \omega_8^2 \cdot p_{11}(\omega_4^2) = -4 - i \cdot (-4) = -4 - 4i \\
p_{00}(\omega_8^7) &= p_{10}(\omega_4^3) - \omega_8^3 \cdot p_{11}(\omega_4^3) = -4 + 4i - \tfrac{i-1}{\sqrt{2}} \cdot (-4 + 4i) = -4 + 4i \cdot (\sqrt{2}+1)
\end{aligned}
$$

Rewriting the discrete fourier transforms as vectors (not strictly necessary, though):

$$
\begin{aligned}
DFT_2(p_{20}) &= (4, -4) \\
DFT_2(p_{21}) &= (8, -4) \\
DFT_2(p_{22}) &= (6, -4) \\
DFT_2(p_{23}) &= (10, -4)
\end{aligned}
$$

$$
\begin{aligned}
DFT_4(p_{10}) &= (12, -4 - 4i, -4, -4 + 4i) \\
DFT_4(p_{11}) &= (16, -4 - 4i, -4, -4 + 4i)
\end{aligned}
$$

$$
\begin{aligned}
DFT_8(p_{00}) = \big(&28, -4 - 4i \cdot (\sqrt{2}+1), -4 - 4i, -4 - 4i \cdot (\sqrt{2}-1), \\
&-4, -4 + 4i \cdot (\sqrt{2}-1), -4 - 4i, -4 + 4i \cdot (\sqrt{2}+1)\big)
\end{aligned}
$$