EXERCISE FOR CSE202 – WEEK 5

Exercise 1. *Consider a punctured grid of size $n \in \mathbf{N}_{\geq 1}$ as defined in TD04. Determine the asymptotic run time complexity of the algorithm from Question 3 from that TD via the master theorem with respect to the number of squares of the grid. Briefly discuss whether this complexity is asymptotically optimal for our setting of tiling with L-shapes.*

Solution : Let $N = 4^n$, that is, the number of squares of the grid, and let $C\colon \{4^i \mid i \in \mathbf{N}_{\geq 1}\} \to \mathbf{N}$ denote the complexity of the algorithm for a grid of a given number of squares.

If $N = 4$, then $C(N)$ is constant. If $N \geq 16$, then the algorithm performs operations with cost $d \in \Theta(1)$ in order to place the middle. Afterward, it recurs four times, each time on a quarter of the input size. This results in

$$T(N) = 4 \cdot T\left(\frac{N}{4}\right) + d\ .$$

In order to apply the master theorem, we compare $N^{\log_4(4)} = N$ to d. We see that there exists a constant $a \in [0, 1)$, that is $a < \log_4(4)$, such that $d \in \mathrm{O}(N^a)$. Thus, via the master theorem, we get that $C \in \Theta(N)$. In other words, the complexity is linear in the number of squares.

This complexity is asymptotically optimal because each algorithm that solves this problem needs to return a set of the L-shapes that has a size of $\lfloor N/3 \rfloor \in \Omega(N)$. $\square$

Exercise 2. *For $d \in \mathbf{N}_{\geq 1}$, we consider* unimodal *d-dimensional (square) arrays whose entries are* unique! *Let $[d] = [1, d] \cap \mathbf{N}$. For $n \in \mathbf{N}_{\geq 1}$, let a d-dimensional array A of* size n *be defined as $A\colon [n]^d \to \mathbf{R}$. Further, a d-dimensional array A of size n has a* local maximum *at position $\boldsymbol{i} \in [n]^d$ if and only if it holds that the at most $2d$ direct neighbors of $A[\boldsymbol{i}]$ are not greater than $A[\boldsymbol{i}]$. Formally, that is the case if and only if, for all $\boldsymbol{j} \in \bigcup_{b \in [d]} \{\boldsymbol{k} \in [n]^d \mid (\forall c \in [d] \smallsetminus \{b\}\colon \boldsymbol{k}_c = \boldsymbol{i}_c) \wedge \boldsymbol{k}_b = \boldsymbol{i}_b \pm 1\}$, it holds that $A[\boldsymbol{j}] \leq A[\boldsymbol{i}]$. We say that A is unimodal if and only if A has* exactly *one local maximum (which is also its global maximum).*

(1) *Let A be a unimodal one-dimensional array of size $n \in \mathbf{N}_{\geq 1}$ with unique entries. Describe a recursive algorithm that finds the local maximum of A in time $\mathrm{O}(\log n)$. Prove the correctness of your algorithm, and conduct a run time analysis using the master theorem.*
(2) *Let A be a unimodal two-dimensional array of size $n \in \mathbf{N}_{\geq 1}$ with unique entries. Describe a recursive algorithm that finds the local maximum of A in time $\mathrm{O}(n)$. Prove the correctness of your algorithm, and conduct a run time analysis using the master theorem.*
(3) *Let $d \in \mathbf{N}_{\geq 3}$. Briefly describe how the method from the first two parts generalizes to unimodal d-dimensional arrays of size $n \in \mathbf{N}_{\geq 1}$ with unique entries. What run time does the master theorem yield?*

Solution : Let $n \in \mathbf{N}_{\geq 1}$. For $i, j \in [n]$, let $[i..j] = [i, j] \cap \mathbf{N}$. When we use such sets as indices for an array A of size n, we refer to A whose domain at the respective dimension is restricted to the set $[i..j]$ (instead of $[n]$).

(1) Before we describe our algorithm, we note that, for each index $i \in [n]$ such that $A[i]$ is not not the local maximum of A, the entry $A[i]$ has at most one neighbor that is larger than $A[i]$. We prove this claim via contradiction, thus assume that $A[i]$ has at least two neighbors that are larger than $A[i]$.

Since A is one-dimensional, $A[i]$ has exactly two neighbors that are larger than it. Let m_1 and m_2 denote the (global) maximum of $A[1..i-1]$ and of $A[i+1..n]$, respectively. Since both of these maxima are larger than each remaining value in their respective subarrays as well as larger than $A[i]$, by assumption, the array A has at least two local maxima, m_1 and m_2, which contradicts the assumption that A is unimodal.

Algorithm. We use this property for our algorithm, which proceeds as follows on an array ranging from $i \in [n]$ to $j \in [n]$: If $A[\lfloor (i+j)/2 \rfloor] =: v$ is a local maximum, then we return it. Otherwise, by our property above, v has exactly one larger neighbor. The algorithm recurs into this subarray, that is, either $A[1..\lfloor (i+j)/2 \rfloor - 1]$ or $A[\lfloor (i+j)/2 \rfloor + 1..n]$.

Correctness. If the currently investigated entry v is a local optimum, the algorithm correctly returns it. If not, it recurs into the subarray B that contains the larger neighbor w of v. Let $M \geq w$ be the (global) maximum of B. Assume that M is not the local maximum of A. By our property, M has then at most one larger neighbor (in A). This cannot be any remaining element in B, because M is the maximum of B. Since v is the only element directly connected to B, it must be the larger neighbor. However, this contradicts $M > v$. Thus, M is the local maximum of A.

Run time. Let $C \colon \mathbf{N}_{\geq 1} \to \mathbf{N}$ denote the run time of the algorithm on an array of a given size $n \in \mathbf{N}_{\geq 1}$. If $n = 1$, note that $C(n) = C(1)$ is constant. If $n \geq 2$, during each recursion, the algorithm performs operations with cost at most $b \in \Theta(1)$. Afterward, it recurs on at most half of its input size. Thus, it follows that

$$C(n) \leq C\left(\left\lceil \frac{n}{2} \right\rceil\right) + b \ .$$

We see that $b \in \Theta(n^{\log_2(1)}) = \Theta(1)$. Thus, it follows via the master theorem that $C \in \mathrm{O}(\log n)$.

(2) Before we describe our algorithm, we show that a certain type of entry does not exist in A. For each $i \in [n]$, let $m_i = \max_{j \in [n]} A[i][j]$ be the maximum of row i. We show, for each $i \in [n]$, that if m_i is not a local maximum, then it has at most one larger neighbor in the neighboring rows. In other words, if m_i is not a local maximum, it is not a saddle point. We prove this claim via contradiction, thus assume that m_i has larger neighbors in rows $i-1$ and $i+1$.

Let M_1 be the (global) maximum of the subarray $A[i-1][[n]]$, and let M_2 be the (global) maximum of the subarray $A[i+1][[n]]$. Since both of these maxima are larger than each remaining value in their respective subarrays as well as all values in row i, since they are both larger than m_i, it follows that A has at least two local maxima, M_1 and M_2. This contradicts the assumption that A is unimodal.

Algorithm. We use this property for our algorithm, which proceeds as follows on an array whose first index ranges from $i \in [n]$ to $j \in [n]$: Let $k = \lfloor (i+j)/2 \rfloor$. If m_k is a local maximum, return it. Otherwise, by our property, m_k has exactly one larger neighbor. The algorithm recurs into this subarray, that is, either $A[1..k-1][[n]]$ or $A[k+1..n][[n]]$.

Correctness. If the currently investigated entry v is a local maximum, the algorithm correctly returns it. If not, it recurs into the subarray B that contains the larger neighbor w of v. Let $M \geq w$ be the (global) maximum of B. Assume that M is not the local maximum of A. By our property, M has at most one larger neighbor in one of its neighboring rows (in A). This cannot be any remaining element in B, because M is the maximum of B. Thus, this larger neighbor must be in row k. However, each element in row k is at most m_k, which is strictly smaller than M. This contradicts that M has at most one larger neighbor in A. Thus, M is the local maximum of A.

Run time. Let $C\colon \mathbf{N}_{\geq 1} \to \mathbf{N}$ denote the run time of the algorithm on a two-dimensional array with a given number of elements. If $n = 1$, note that $C(n^2) = C(1)$ is constant. If $n \geq 2$, during each recursion, the algorithm performs operations with cost at most $b \leq 2n$, as it operates on a single row (and at most two other elements). Afterward, it recurs on at most half of its input size. Thus, it follows that

$$C(n^2) \leq C\left(\left\lceil \frac{n^2}{2} \right\rceil\right) + b\ .$$

We see that there exists a constant $a > 0$ such that $b \in \Omega(n^{2\log_2(1)+a}) = \Omega(n^a)$. Further, there exists a constant $d \in (0, 1)$ such that $n/2 \leq dn$. Thus, it follows via the master theorem that $C \in \mathrm{O}(b) = \mathrm{O}(n)$.

(3) Similarly to before, for each $i \in [n]$, if the maximum of $A[i]$ (that is, the subarray where the first entry is fixed to i) is not a local maximum of A, it has at most one larger neighbor in a neighboring row. Thus, the algorithm operating on an array whose first index ranges from $i \in [n]$ to $j \in [n]$ first checks whether the maximum of $A[\lfloor (i+j)/2 \rfloor]$ is a local maximum. If so, it returns it. If not, then the algorithm recurs into the subarray with the larger neighbor. The correctness follows from the property analogously as it did in the previous cases.

Run time. Let $C\colon \mathbf{N}_{\geq 1} \to \mathbf{N}$ denote the run time of the algorithm on a d-dimensional array with a given number of elements. If $n = 1$, note that $C(n^d) = C(1)$ is constant. If $n \geq 2$, during each recursion, the algorithm performs operations with cost at most $b \leq 2n^{d-1}$, as it operates on a $(d-1)$-dimensional array (and at most two other elements). Afterward, it recurs on at most half of its input size. Thus, it follows that

$$C(n^2) \leq C\left(\left\lceil \frac{n^2}{2} \right\rceil\right) + b\ .$$

We see that there exists a constant $a > 0$ such that $b \in \Omega(n^{2\log_2(1)+a}) = \Omega(n^a)$. Further, there exists a constant $c \in (0, 1)$ such that $n^{d-1}/2 \leq cn^{d-1}$. Thus, it follows via the master theorem that $C \in \mathrm{O}(b) = \mathrm{O}(n^{d-1})$.

Note that the algorithm is basically the same for all dimensions $d \in \mathbf{N}_{\geq 1}$. However, the *run time* is different if $d = 1$, where it is $\mathrm{O}(\log n)$ instead of $\mathrm{O}(1)$. The reason for this difference is that the recursion always is logarithmic, which is of a lower order of magnitude if $d \geq 2$ but not if $d = 1$. □