Reminder: \( \log n \) denotes the binary logarithm, i.e., \( \log n = \log_2 n \).

1. **Recurrences: Substitution Method**

1. Consider the following recurrence:

\[
T(1) = 1 \quad \text{and} \quad T(n) = T(n-1) + n
\]

Show that \( T(n) \in O(n^2) \) using the substitution method.

**Solution.** We need to show that \( T(n) \leq C \cdot n^2 \), for some suitable constant \( C \). To this end, we first plug our guess into the recurrence:

\[
T(n) = T(n-1) + n \leq C(n-1)^2 + n.
\]

It is required that \( n^2 - 2n + 1 \leq 0 \) for every \( C \geq 1 \).

Observe that \( n^2 - 2n + 1 \) holds for every \( n \geq 1 \). Our guess thus holds for every \( C \geq 1 \).

It remains to verify the base case. We have \( T(1) = 1 \) and \( C1^2 = C \). Hence, \( C1^2 \leq T(1) \) holds for every \( C \geq 1 \). We thus choose \( C = 1 \).

We have shown that \( T(n) \leq Cn^2 = n^2 \) holds for every \( n \geq 1 \). This implies that \( T(n) = O(n^2) \).

2. Consider the following recurrence:

\[
T(1) = 1 \quad \text{and} \quad T(n) = T(\lfloor n/2 \rfloor) + 1
\]

Show that \( T(n) \in O(\log n) \) using the substitution method.

**Hint:** Use the inequality \( \lfloor n/2 \rfloor \leq \frac{n}{\sqrt{2}} = \frac{n}{2^{1/2}} \), which holds for all \( n \geq 2 \). Use \( n = 2 \) as your base case.
Solution. We need to show that $T(n) \leq C \cdot \log n$, for a suitable constant $C$. To this end, we plug our guess into the recurrence:

$$T(n) = T(\lceil n/2 \rceil) + 1 \leq C \cdot \log \left( \frac{n}{\sqrt{2}} \right) + 1 = C \log(n) - C \cdot \frac{1}{2} \log(2) + 1 = C \log(n) - \frac{1}{2} C + 1,$$

where we used the inequality $\lceil n/2 \rceil \leq \frac{n}{\sqrt{2}}$. It is required that $C \log(n) - \frac{1}{2} C + 1 \leq C \log(n)$:

$$C \log(n) - \frac{1}{2} C + 1 \leq C \log(n)$$

$$1 \leq \frac{1}{2} C$$

$$2 \leq C.$$

The “induction step” part of the proof thus works for any $C \geq 2$. Regarding the base case, we will consider $n = 2$. We have:

$$T(2) = T(1) + 1 = 2.$$

We thus need to show that $2 \leq C \log 2$. This holds for every $C \geq 2$. We can thus pick the value $C = 2$. This proves that $T(n) \in O(\log n)$.

2 Search in a Sorted Matrix

We are given an $n$-by-$n$ integer matrix $A$ that is sorted both row- and column-wise, i.e., every row is sorted in non-decreasing order from left to right, and every column is sorted in non-decreasing order from top to bottom. Give a divide-and-conquer algorithm that answers the question:

“Given an integer $x$, does $A$ contain $x$?”

What is the runtime of your algorithm?

Solution. For simplicity, we assume that $n$ is a power of two in this solution. We define the following submatrices of matrix $A$:

$$A_{11} = A[0 \ldots \frac{n}{2} - 1, 0 \ldots \frac{n}{2} - 1]$$

$$A_{21} = A[\frac{n}{2} \ldots n, 0 \ldots \frac{n}{2} - 1]$$

$$A_{12} = A[0 \ldots \frac{n}{2} - 1, \frac{n}{2} \ldots n - 1]$$

$$A_{22} = A[\frac{n}{2} \ldots n - 1, \frac{n}{2} \ldots n - 1]$$

Observe that the dimensions of all submatrices are $n/2 \times n/2$.

We first check whether $A[\frac{n}{2} - 1, \frac{n}{2} - 1] = x$. If this is the case then we have found $x$ and we are done. Otherwise, we distinguish the following two cases:
1. Suppose that \( A_{\frac{n}{2}-1, \frac{n}{2}-1} < x \) holds. Then, since \( A \) is sorted in both column and row order, it is not hard to see that \( x \) is not contained in \( A_{11} \). We then invoke our algorithm recursively and search for \( x \) in the three submatrices \( A_{12}, A_{21}, A_{22} \).

2. Suppose that \( A_{\frac{n}{2}-1, \frac{n}{2}-1} > x \) holds. Then, similar as before, it is not hard to see that \( x \) is not contained in \( A_{22} \). We then invoke our algorithm recursively and search for \( x \) in the three submatrices \( A_{11}, A_{12}, A_{21} \).

Observe that the proposed algorithm is a recursive algorithm. We thus need to decide what to do if the input to a recursive call is a \( 1 \times 1 \) matrix. In this case we simply check whether the single element in the matrix equals \( x \) in \( O(1) \) time.

Let \( T(n) \) be the runtime of the algorithm when executed on an input array of dimension \( n \times n \). We thus obtain the following recurrence:

\[
T(n) = \begin{cases} 
O(1) , & \text{if } n = 1 , \\
3T(n/2) + O(1) , & \text{otherwise}.
\end{cases}
\]

It remains to solve the recurrence \( T(n) \). First, we eliminate the \( O(1) \) terms and replace them with a large enough constant \( C \):

\[
T(n) = \begin{cases} 
C , & \text{if } n = 1 , \\
3T(n/2) + C , & \text{otherwise}.
\end{cases}
\]

Our recursion is simple enough to obtain a solution via the recursion tree method. In the lecture, we used the recursion tree method in order to obtain a guess the we then verified using the substitution method. The recursion here is however simple enough to conduct a complete analysis using the recursion tree.

From the recursion tree, we see that the tree has \( \log(n) + 1 \) levels. Denoting the root of the tree as level 0, we see that level \( i \) has \( 3^i \) nodes. Furthermore, every node is labeled by \( C \). The total work therefore is:

\[
\sum_{i=0}^{\log n} 3^i C = C \cdot \sum_{i=0}^{\log n} 3^i = C \cdot \frac{3^{\log(n)+1} - 1}{3 - 1} = \frac{C}{2} \cdot \left( 2^{\log(3) \log(n) + \log(3)} - 1 \right) \leq \frac{C}{2} \cdot \left( 2^{\log(3) \log(n) + \log(3)} \right) = \frac{C}{2} \cdot \left( n^{\log 3} \cdot 3 \right) = O(n^{\log 3}) \approx O(n^{1.5849...}) .
\]

We used the formula \( \sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1} \) in this calculation.

Last, I would like to mention that there exists a solution to this problem that runs in time \( O(n) \). Can you think of such a solution?