Fundamental Algorithms 4

Exercise 1

Try the Recursion Tree Method (compare lecture) for the following recurrence:

\[ T(n) = T(n/3) + T(2n/3) + O(n) \]

Show that the height of the recursion tree is in \( O(\log(n)) \).

- We assume that all occurring \( n \) are multiples of 3. Further, let \( c \) be the constant in the \( O(n) \) term. We then obtain the recursion tree

\[
\begin{array}{c}
\text{cn} \\
\text{c(n/3)} & \text{c(2n/3)} \\
\text{c(n/9)} & \text{c(2n/9)} & \text{c(2n/9)} & \text{c(4n/9)} \\
\end{array}
\]

On each level, we obviously obtain \( cn \) operations, independent of the level.

- The longest path in the recursion tree is the rightmost path with problem size \( n \to 2/3n \to (2/3)^2n \to \cdots \to 1 \) until we stop at problem size 1. The height \( h \) of the tree can be determined via the equation \( (2/3)^h n = 1 \), leading to \( h = \log_{3/2} n \).

We could expect the total cost to be \( O(cn \log_{3/2} n) = O(n \log n) \).

What could be a flaw using the recursion tree method for such unbalanced trees? Show that \( T(n) \in O(n \log(n)) \), anyway, by using the substitution method.

- Problem: If the tree was a complete binary tree, we would have \( 2^{\log_{3/2} n} = n^{\log_{3/2} 2} \) leaves (as \( \log_{3/2} n = \log_{2} n / \log_{2} 3/2 = \log_{2} n / \log_{2} 2 \), using the formula \( \log_{a} b = 1 / \log_{b} a \)). As \( \log_{3/2} 2 > 1 \), the number of terms would be \( \omega(n \log n) \) on the last level. Hence, the simple approach of assuming constant effort \( c \) for \( T(1) \) on the final level does no longer work: in that case, the costs would sum up to \( \Theta(cn^{\log_{3/2} 2}) \) on the last level – and not \( cn! \)
Hence, we’d have to explicitly consider that the tree starts to thin out much earlier (starting at level $1 + \log_3 n$), and we would have to examine the exact cost on all subsequent levels, which is more tedious than our tree diagram suggests.

- We simplify and assume that the total cost are $O(n \log n)$ and use the substitution method to verify this:

  Assuming that $T(n) \leq an \log n$ for a suitable constant $a$, we obtain

  \[
  T(n) \leq T(n/3) + T(2n/3) + cn \\
  \leq a(n/3) \log(n/3) + a(2n/3) \log(2n/3) + cn \\
  = a3n/3 \log n - a ((n/3) \log 3 + (2n/3) \log(3/2)) + cn \\
  = an \log n - a ((n/3) \log 3 + (2n/3) \log 3 - (2n/3) \log 2) + cn \\
  = an \log n - an (\log 3 - 2/3 \log 2) + cn \\
  \leq an \log n
  \]

  for $d \geq c / (\log 3 - 2/3 \log 2)$.

**Exercise 2**

For the so-called BFPRT Algorithm, an algorithm to determine the median element of an array, we obtain the following (slightly simplified) recurrence equation for its running time $T(n)$ (depending on the number $n$ of elements):

\[
T(n) = s(n,k) + T\left(\frac{n}{k}\right) + T\left(\frac{l}{2k}n\right).
\]

$k$ and $l$ are parameters ($k$ usually small, for example $k = 3$ or $k = 5$) where $k = 2l + 1$. For the function $s$, we can assume $s(n,k) \in \Theta(n \log k)$.

a) Show that $T(n) \in O(n)$.

b) Does it make sense to use large values for $k$ (and $l$, resp.)?

**Solution:**

We try to prove the claim by inserting the assumed solution $T(n) \leq cn$ into the recurrence equation:

\[
\begin{align*}
\frac{cn}{k} & \geq s(n,k) + \frac{c}{k} \cdot n \\
\Leftrightarrow c(n - \frac{n}{k} - \frac{l}{2k}n) & \geq s(n,k)
\end{align*}
\]

As $s(n,k) \in \Theta(n \log k)$, there is a constant $C_s$ such that $s(n,k) \leq C_s n \log k$ for large enough $n$. Therefore, $c$ has to be large enough to satisfy

\[
\begin{align*}
c(n - \frac{n}{k} - \frac{l}{2k}n) & \geq C_s n \log k \geq s(n,k) \\
\Leftrightarrow c & \geq \frac{C_s \log k}{1 - \frac{1}{k} - \frac{l}{2k}} \in O(\log k)
\end{align*}
\]

Hence, we can choose a suitable, large enough $c$ that is independent of $n$, and thus prove $T(n) \in O(n)$, but the involved constant has to slightly grow with $k$, as $c \in O(\log k)$. As a consequence, $k$ should be of limited size.