Fundamental Algorithms 2

– Solution examples –

Exercise 1

HOME COMPUTER shall be a machine that can perform $10^9$ operations per second. Consider that we have five different algorithms for a specific problem. For each algorithm $i$, we know the number of operations $T_i(n)$ it will perform on a problem of size $n$:

- $T_1(n) = 6 \, 000 \, 000 \cdot n$
- $T_2(n) = 60 \, 000 \cdot n \log n$
- $T_3(n) = 0.003 \cdot n^2$
- $T_4(n) = 10^{-6} \cdot n^3$
- $T_5(n) = 10^{-18} \cdot 2^n$

For each algorithm compute the size $n_{\text{max}}$ of the largest problem the respective algorithm can solve within 1 second (1 minute, 1 hour, ...). Enter the maximal problem sizes into the following table:

<table>
<thead>
<tr>
<th></th>
<th>1 second</th>
<th>1 minute</th>
<th>1 hour</th>
<th>1 day</th>
<th>1 month (30 d)</th>
<th>1 year (365 d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}(T_1)$</td>
<td>166</td>
<td>10 000</td>
<td>600 000</td>
<td>1.44 · $10^7$</td>
<td>4.32 · $10^8$</td>
<td>5.26 · $10^9$</td>
</tr>
<tr>
<td>$n_{\text{max}}(T_2)$</td>
<td>2 169</td>
<td>87 847</td>
<td>3.95 · $10^6$</td>
<td>7.91 · $10^7$</td>
<td>2.01 · $10^9$</td>
<td>2.20 · $10^{10}$</td>
</tr>
<tr>
<td>$n_{\text{max}}(T_3)$</td>
<td>577 350</td>
<td>4.47 · $10^9$</td>
<td>3.46 · $10^7$</td>
<td>1.69 · $10^8$</td>
<td>9.29 · $10^9$</td>
<td>3.24 · $10^{10}$</td>
</tr>
<tr>
<td>$n_{\text{max}}(T_4)$</td>
<td>100 000</td>
<td>391 486</td>
<td>1.53 · $10^6$</td>
<td>4.42 · $10^6$</td>
<td>1.37 · $10^7$</td>
<td>3.15 · $10^8$</td>
</tr>
<tr>
<td>$n_{\text{max}}(T_5)$</td>
<td>89</td>
<td>95</td>
<td>101</td>
<td>106</td>
<td>110</td>
<td>114</td>
</tr>
</tbody>
</table>

Exercise 2

Compute the number of comparisons that will be performed by MERGESORT in the best case.

Solution:

Note: we only count comparisons between array elements!

In MERGESORT, comparisons are only performed during the MERGE-step. In the best case, the first element of one partition will be compared to (and found to be larger than) all $\frac{n}{2}$ elements of the other partition. After copying these $\frac{n}{2}$ elements of the other partition into the array, the elements of the first partitions will be copied to the array without performing any comparison. Thus, in the best case, MERGE will require $\frac{n}{2}$ comparisons.
If we assume \( n = 2^k \), we get the following recurrence for the number \( C(n) \) of comparisons in the best case:

\[
C(n) = C\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right) + \frac{n}{2} = 2C\left(\frac{n}{2}\right) + \frac{n}{2}
\]

Of course, \( C(1) = 0 \).

We try to solve this recurrence by substitution, and guess

\[
C(n) := an \log_2 n + b
\]

as the solution.

For \( n = 1 \), we get:

\[
C(1) = a \log_2 1 + b = b = 0 \iff b = 0
\]

And for \( n > 1 \):

\[
\begin{align*}
an \log_2 n &= C(n) = 2C\left(\frac{n}{2}\right) + \frac{n}{2} \\
\iff an \log_2 n &= 2 \left(\frac{n}{2} \log_2 \frac{n}{2}\right) + \frac{n}{2} \\
\iff an \log_2 n &= 2 \left(\frac{n}{2} \log_2 n - 1\right) + \frac{n}{2} \\
\iff an \log_2 n &= an \log_2 n - an + \frac{n}{2} \\
\iff 0 &= -an + \frac{n}{2} \\
\iff a &= 1 \frac{1}{2}
\end{align*}
\]

Hence, in the best case, MERGESORT will require \( \frac{n}{2} \log_2 n \) comparisons between array elements (for \( n = 2^k, k \in \mathbb{N} \)).

**Exercise 3 (Sorting)**

Prove or disprove the following statement:

If we sort each row of a matrix, and, after that, sort each column of the matrix, the rows of the matrix will still be sorted afterwards.

**Solution:**

Let \( (a_{ij}) \) be the matrix after the entire sorting procedure. Without loss of generality, we will assume that the sorting is done is ascending order. Our prove will be by contradiction:

Assume that there is a row \( l \) that is not in sorted order, i.e. there are two column indices \( j < k \) such that \( a_{lj} > a_{lk} \):

As the \( k \)-th column of the matrix is sorted, it contains at least \( l \) elements that are \( \leq a_{lk} \) (\( a_{lk} \) included). To each of these \( l \) elements belongs an element of the \( j \)-th column that was in the same line of the matrix before the columns where sorted. These \( l \) elements are all smaller or equal to their corresponding element in the \( k \)-th column, because the lines of the matrix where already sorted at this time. Hence, these \( l \) elements of the \( j \)-th column are all \( \leq a_{kj} \), and therefore \( < a_{lj} \).

Hence, there are at least \( l \) elements in column \( j \) that are that are smaller than \( a_{lj} \). Therefore, in column \( j \), at least one element smaller than \( a_{lj} \) has to be placed below \( a_{lj} \). This means that the \( j \)-th column can not be in sorted order, which contradicts our assumption.