Technische Universität München
Institut für Informatik
Theoretical Computer Science

Fundamental Algorithms 2 - Solution Examples

Exercise 1 (Real Complexity)

Suppose HomeComputer is 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:

$$\begin{aligned}
T_1(n) &= 6\,000\,000 \cdot n \in O(n) \\
T_2(n) &= 60\,000 \cdot n \ln n \in O(n \ln n) \\
T_3(n) &= 0.003 \cdot n^2 \in O(n^2) \\
T_4(n) &= 10^{-6} \cdot n^3 \in O(n^3) \\
T_5(n) &= 10^{-18} \cdot 2^n \in O(2^n)
\end{aligned}$$

For each algorithm compute the size $n_{\max}$ 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:

	1 second	1 minute	1 hour	1 day	1 month (30 d)	1 year (365 d)
$n_{\max}(T_1)$	166	10 000	600 000	$1.44 \cdot 10^7$	$4.32 \cdot 10^8$	$5.26 \cdot 10^9$
$n_{\max}(T_2)$	2 169	87 847	$3.95 \cdot 10^6$	$7.91 \cdot 10^7$	$2.01 \cdot 10^9$	$2.20 \cdot 10^{10}$
$n_{\max}(T_3)$	577 350	$4.47 \cdot 10^6$	$3.46 \cdot 10^7$	$1.69 \cdot 10^8$	$9.29 \cdot 10^8$	$3.24 \cdot 10^9$
$n_{\max}(T_4)$	100 000	391 486	$1\,53 \cdot 10^6$	$4.42 \cdot 10^6$	$1.37 \cdot 10^7$	$3.15 \cdot 10^7$
$n_{\max}(T_5)$	89	95	101	106	110	114

Exercise 2 (MergeSort)

Compute the number of comparisons between array elements that will be performed by MergeSort on an array of size $n = 2^k$, $k \in \mathbb{N}$ in the best case, i.e., compute this number exactly.

Solution:
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.

With $n = 2^k$, we get the following recurrence for the number $C(n)$ of comparisons in the best case:

$$C(n) = C\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right) + \frac{n}{2} = 2 \cdot C\left(\frac{n}{2}\right) + \frac{n}{2}$$

Of course, $C(1) = 0$. We try to solve this recurrence by substitution, and guess

$$C(n) := an \ln_2 n + b$$

as the solution. For $n = 1$, we get:

$$C(1) = a \ln_2 1 + b = b = 0 \quad \Leftrightarrow \quad b = 0$$

And for $n > 1$:

$$
\begin{aligned}
an \ln_2 n = C(n) &= 2 \cdot C\left(\frac{n}{2}\right) + \frac{n}{2} \\
an \ln_2 n &= 2\left(a\frac{n}{2} \ln_2 \frac{n}{2}\right) + \frac{n}{2} \\
an \ln_2 n &= 2\left(a\frac{n}{2} (\ln_2 n - 1)\right) + \frac{n}{2} \\
an \ln_2 n &= an \ln_2 n - an + \frac{n}{2} \\
0 &= -an + \frac{n}{2} \\
a &= \frac{1}{2}
\end{aligned}
$$

Hence, in the best case, MergeSort will require $\frac{n}{2} \ln_2 n$ 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 proof 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 $\leq a_{lk}$, and therefore $< a_{lj}$.

Consequently, 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.