Technische Universität München
Institut für Informatik
Theoretical Computer Science

WS 2017/18

Fundamental Algorithms 3 - Solution Examples

Exercise 1 (Worst-Case)

Consider a partitioning algorithm that, in the worst case, will partition an array of m elements into two partitions of size $\lfloor \varepsilon m \rfloor$ and $\lceil (1-\varepsilon)m \rceil$, where ε is fixed and $0 < \varepsilon < 1$. Show that a QuickSort algorithm based on this partitioning has a worst-case complexity of $O(n \log n)$ (in terms of comparisons between array elements). *Hint:* Solve the recurrence by guessing the solution and finding the involved constants.

Solution:
Using that the partitioning step will require at most n comparisons, we get the following recurrence for the necessary number $C(n)$ of comparisons:

$$\begin{aligned} C(1) &= 0 \\ C(n) &= C(\varepsilon n) + C((1-\varepsilon)n) + n \end{aligned}$$

We guess $C(n) := an\log_2 n + b$ as the solution, and search constants a and b such that the recurrence is satisfied:

$n = 1$: We have

$$C(1) = a \cdot 1 \cdot \log_2 1 + b = 0 \quad \Leftrightarrow \quad b = 0,$$

hence, $C(n) = an\log_2 n$.

$n > 1$: We insert our guess into the recurrence:

$$\begin{aligned} an\log_2 n &= C(n) = C(\varepsilon n) + C((1-\varepsilon)n) + n \\ an\log_2 n &= a\varepsilon n\log_2(\varepsilon n) + a(1-\varepsilon)n\log_2((1-\varepsilon)n) + n \\ an\log_2 n &= a\varepsilon n\left(\log_2\varepsilon + \log_2 n\right) + a(1-\varepsilon)n\left(\log_2(1-\varepsilon) + \log_2 n\right) + n \\ an\log_2 n &= a\varepsilon n\log_2\varepsilon + a\varepsilon n\log_2 n + a(1-\varepsilon)n\log_2(1-\varepsilon) + a(1-\varepsilon)n\log_2 n + n \\ an\log_2 n &= a\varepsilon n\log_2\varepsilon + a\varepsilon n\log_2 n + an\log_2(1-\varepsilon) - a\varepsilon n\log_2(1-\varepsilon) + an\log_2 n - a\varepsilon n\log_2 n + n \\ 0 &= a\varepsilon n\log_2\varepsilon + an\log_2(1-\varepsilon) - a\varepsilon n\log_2(1-\varepsilon) + n \\ 0 &= an\left(\varepsilon\log_2\varepsilon + (1-\varepsilon)\log_2(1-\varepsilon)\right) + n \\ a &= \frac{-1}{\varepsilon\log_2\varepsilon + (1-\varepsilon)\log_2(1-\varepsilon)} \end{aligned}$$

Thus, the recurrence is satisfied if

$$C(n) = \frac{-n\log_2 n}{\varepsilon\log_2\varepsilon + (1-\varepsilon)\log_2(1-\varepsilon)}$$

Note that the constant a will be very large for values of ε that are close to either 0 or 1 (since $\varepsilon\log\varepsilon \to 0$ for $\varepsilon \to 0$). Thus, even very bad partitions will not destroy the $O(n\log n)$ complexity, provided that the respective partition sizes are bounded by εn and $(1-\varepsilon)n$. However, bad partitions will still lead to slow algorithms due to the large constant factor involved.

Exercise 2 (Iterative MergeSort)

The following iterative implementation of the MergeSort algorithm is proposed. The procedure MergeIP is equivalent to the procedure Merge discussed in the lecture, but can work directly on the array A (i.e., merges two adjacent sub-arrays of A).

Algorithm 1: MergeSortIt

```
Input: $A$: Array of size $n = 2^k$
Result: Array $A$ sorted
$k \leftarrow \log_2(n)$;
$m \leftarrow 2$;
for $L = 1$ to $k$ do
    for $i = 0$ to $(n/m) - 1$ do
        MergeIP($A[i \cdot m \;..\; i \cdot m + (m/2) - 1]$,
          $A[i \cdot m + (m/2) \;..\; i \cdot m + (m-1)]$,
          $A[i \cdot m \;..\; i \cdot m + (m-1)]$);
    end
    $m \leftarrow 2 \cdot m$;
end
```

1. Describe shortly and in plain words, how MergeSortIt compares to the recursive MergeSort implementation discussed in the lecture. For that purpose, draw a diagram that illustrates the sorting of some array with length 8 for MergeSortIt.
2. Formulate a loop invariant for the L-loop of the algorithm, and prove its correctness.

Solution:

1. In each iteration of the L-loop, two adjacent sub-arrays are merged. The lengths of the merged sub-arrays ($m/2$) is doubled from each L-loop iteration to the next. In that way, the *same* merging steps as for the recursive implementation of MergeSort are executed. The divide steps are implicitly performed on the array.
2. We propose the following loop invariant:

   > At entry of the L-loop, the array A consists of $\frac{2n}{m}$ sub-arrays of length $\frac{m}{2}$, where $m = 2^L$. Each of the sub-arrays is sorted.

   Here's a sketch of the proof:

   Initialisation: On the first entry, for $L = 1$ and $m = 2^1$, the length of the sub-arrays is claimed to be $\frac{m}{2} = 1$ with $\frac{2n}{2} = n$ sorted sub-arrays. This is satisfied, since sub-arrays of length 1 are always sorted.

   Maintenance: The i-loop will take $\frac{n}{m}$ pairs of two adjacent sub-arrays and merge them using the procedure MergeIP. Provided the correctness of MergeIP, this will lead to $\frac{n}{m}$ sub-arrays of twice the length, which satisfies the loop invariant for the next iteration. Note that m is multiplied by 2, to retain $m = 2^L$.

   Termination: At termination, $L = k + 1$ and thus $m = 2^{k+1} = 2n$. Hence, we have only $\frac{2n}{2n} = 1$ sub-array of length $\frac{2n}{2} = n$, which is sorted. This implies the correctness of the sorting algorithm.