Exercise Sheet 3
COMS10007 Algorithms 2018/2019

26.03.2019

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

1 Countingsort and Radixsort

1. Illustrate how Countingsort sorts the following array:

4	2	2	0	1	4	2

2. Illustrate how Radixsort sorts the following binary numbers:

   100110 101010 001010 010111 100000 000101

3. Radixsort sorts an array A of length n consisting of d-digit numbers where each digit is from the set $\{0, 1, \ldots, b\}$ in time $O(d(n + b))$.

   We are given an array A of n integers where each integer is *polynomially bounded*, i.e., each integer is from the range $\{0, 1, \ldots, n^c\}$, for some constant c. Argue that Radixsort can be used to sort A in time $O(n)$.

   Hint: Find a suitable representation of the numbers in $\{0, 1, \ldots, n^c\}$ as d-digit numbers where each digit comes from a set $\{0, 1, \ldots, b\}$ so that Radixsort runs in time $O(n)$. How do you chose d and b?

2 Loop Invariant for Radixsort

Radixsort is defined as follows:

```
Require: Array A of length n consisting of d-digit numbers where each digit
    is taken from the set {0, 1, ..., b}
1: for i = 1, ..., d do
2:     Use a stable sort algorithm to sort array A on digit i
3: end for
```

(least significant digit is digit 1)

In this exercise we prove correctness of Radixsort via the following loop invariant:

At the beginning of iteration i of the for-loop, i.e., after i has been updated in Line 1 but Line 2 has not yet been executed, the following holds:

The integers in A are sorted with respect to their last $i-1$ digits.

1. *Initialization:* Argue that the loop-invariant holds for $i = 1$.
2. *Maintenance:* Suppose that the loop-invariant is true for some i. Show that it then also holds for $i + 1$.

   Hint: You need to use the fact that the employed sorting algorithm as a subroutine is stable.
3. *Termination:* Use the loop-invariant to conclude that A is sorted after the execution of the algorithm.

3 Recurrences: Substitution Method

1. Consider the following recurrence:

   $$T(1) = 1 \text{ and } T(n) = T(n-1) + n$$

   Show that $T(n) \in O(n^2)$ using the substitution method.
2. Consider the following recurrence:

   $$T(1) = 1 \text{ and } T(n) = T(\lceil n/2 \rceil) + 1$$

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

   Hint: Use the inequality $\lceil n/2 \rceil \leq \frac{n}{\sqrt{2}} = \frac{n}{2^{\frac{1}{2}}}$, which holds for all $n \geq 2$. Use $n = 2$ as your base case.

4 Recurrences: Recursion Tree Method

1. Use a recursion tree to determine a good asymptotic upper bound on the recurrence

   $$T(n) = 1 \text{ for every } n \leq 10 \text{ and } T(n) = 4T(n/2 + 2) + n \text{ for every } n > 10 \ .$$

   Use the substitution method to verify your answer. (this is a difficult question!)

   Hint: Ignore the additive 2 for a rough analysis using the recursion tree. For the substitution method, use at least one lower order term.
2. Use a recursion tree to determine a good asymptotic upper bound on the recurrence

   $$T(1) = 1 \text{ and } T(n) = 2T(n-1) + 1 \ .$$

   Use the substitution method to verify your answer.

5 Fibonacci Numbers

Consider the algorithm ImprovedDynPrgFib(n) for computing the Fibonacci numbers as presented on slide 13 of Lecture 15. In this exercise, the goal is to prove that the algorithm indeed computes the nth Fibonacci number.

1. Give a suitable loop invariant (it should involve at least variables a and b).
2. Prove that the loop invariant is correct: It holds at the beginning of the algorithm, it is maintained throughout the algorithm, and we can conclude from the loop invariant that the algorithm indeed computes the nth Fibonacci number.