University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Schneider

Algorithms and Data Structures
Winter Term 2019/2020
Exercise Sheet 1

Remark: For this exercise sheet, watch the first two video lectures given on the lecture website. Ignore the part on organization in the video. If you have questions about the exercise sheet (or this course in general) you can ask them in the forum. The link for the forum is on the website.

Exercise 1: Bubblesort

The following pseudocode describes the BUBBLESORT algorithm with input array A of length n.

Algorithm 1 BUBBLESORT($A[0,\ldots,n-1]$)

```
for i = 0 to n − 2 do
    for j = 0 to n − 2 do
        if A[j] > A[j+1] then
            SWAP(j, j+1)        ▷ operation SWAP(j, j+1) swaps array entries A[j] and A[j+1]
```

(a) Assume BUBBLESORT runs on input $A = [27, 8, 19, 5, 23, 12]$. Give A after the end of each iteration of the outer for-loop.

(b) Give the (worst-case) runtime for BUBBLESORT as a function of n. Explain your answer.

Remark: To simplify things, you may assume that each cycle of a loop (inner or outer) takes one unit of time and neglect the time required for other operations.

(c) Argue why BUBBLESORT is correct (i.e., array A is always sorted after the algorithm is finished).

Exercise 2: Counting Sort

The following pseudocode describes the COUNTINGSORT algorithm which receives an array $A[0\ldots n{-}1]$ as input containing values in $[0..k]$. Additionally there is an Array `counts`$[0\ldots k]$ initialized with 0.

Algorithm 2 COUNTINGSORT(A, `counts`) ▷ **integer arrays** $A[0\ldots n{-}1]$, `counts`$[0\ldots k]$

```
for i ← 0 to n − 1 do
    counts[A[i]] ++                ▷ ++ is the increment operation
i ← 0
for j ← 0 to k do
    for ℓ ← 1 to counts[j] do
        A[i] ← j
        i ++
```

(a) Assume COUNTINGSORT runs on input $A = [5, 2, 3, 0, 5, 3, 4, 2, 5, 0, 1, 3, 5, 0, 0]$. Give A and `counts` after the algorithm has terminated.

(b) Give the (worst-case) runtime of CountingSort as a function of n and k. Explain your answer.

Remark: To simplify this, you may assume that each cycle of a loop (inner or outer) takes one unit of time. You may neglect the time required by other operations.

(c) Argue why CountingSort is correct (i.e., array A is always sorted after the algorithm is finished).

Exercise 3: Implementation

(a) Implement the above two algorithms in a programming language of your choice (in the lecture and exercise class we will see/use Python).[1]

(b) Test your implementation of both algorithms with random inputs as follows. Generate input arrays *of length 10, 100 and 1000* respectively, each *filled with randomly generated integer values ranging from 0 to 1000*. Run each algorithm on each input and check the correctness.

(c) Implement some functionality to measure the elapsed time of your algorithms from start to finish (e.g., by using the python-module *time*). Run the algorithms again with the above inputs and note down the elapsed times. What do you observe?

[1] As a side-note: In this course we assume that you have some (very) basic programming skills, enabling you to implement short pseudo codes like the ones given above in a programming language of your choice. Since this course is more on the theoretical side, we will not ask much more than that in terms of programming skills. If you never attended some programming-course and/or experience difficulties to implement the above algorithms, please try to catch up using literature, tutorials and/or contact us.