Institut for Matematik og Datalogi
Syddansk Universitet, Odense

13. March 2025

# Exercise Sheet Week 5

# DSK814

## A. Solve in the practice sessions

1. Cormen et al., 4. Edition: Exercise 6.4-4 (page 172) [Cormen et al., 3. Edition: Exercise 6.4-4 (page 160)].

   Show that the worst-case running time of HEAPSORT is $\Omega(n \log n)$.

   *Hint: Is* HEAPSORT *a comparison-based algorithm?*

2. Cormen et al., 4. Edition: Exercise 8.2-1 (page 210) [Cormen et al., 3. Edition: Exercise 8.2-1 (page 196)].

   Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array $A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]$.

3. Cormen et al., 4. Edition: Exercise 8.2-3 (page 211) [Cormen et al., 3. Edition: Exercise 8.2-3 (page 196)].

   Suppose that we were to rewrite the for loop header in line 11 of the COUNTING-SORT as

   11 **for** $j = 1$ **to** $n$.

   Show that the algorithm still works properly, but that it is not stable. Then rewrite the pseudocode for counting sort so that elements with the same value are written into the output array in order of increasing index and the algorithm is stable.

4. Cormen et al., 4. Edition: Exercise 8.2-6 (page 211) [Cormen et al., 3. Edition: Exercise 8.2-4 (page 197)].

   Describe an algorithm that, given $n$ integers in the range 0 to $k$, preprocesses its input and then answers any query about how many of

the $n$ integers fall into a range $[a : b]$ in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

5. Cormen et al., 4. Edition: Exercise 8.3-2 (page 215) [Cormen et al., 3. Edition: Exercise 8.3-2 (page 200)].

   Which of the following sorting algorithms are stable: InsertionSort, Mergesort, Heapsort and Quicksort? Give a simple scheme that makes any comparison sort stable. How much additional time and space does your scheme entail?

   *Hint for the second question: expand element keys.*

6. Perform RadixSort with radix 10 on the numbers

   $$747, 765, 544, 754, 431, 231, 222.$$

   Show the result after each iteration.

7. (∗) Cormen et al., 4. Edition: Exercise 8.3-5 (page 215) [Cormen et al., 3. Edition: Exercise 8.3-4 (page 200)].

   Show how to sort $n$ integers in the range 0 to $n^3 - 1$ in $O(n)$ time.

   *Hint: consider 3 digit numbers $d_2d_1d_0$ with radix $n$, i.e. $x = d_2 \cdot n^2 + d_1 \cdot n^1 + d_0 \cdot n^0$, where $0 \leq d_i \leq n-1$ for $i = 0, 1, 2$. For any number $x$, these digits can be computed repeatedly using integer division and the modulo operation with $n$ (see Cormen et al., 4. Edition: Formula (3.11) page 64 [Cormen et al., 3. Edition: Formula (3.8) page 54]), similiar to the algorithm on page 10–12 on these number system slides (where radix $n$ equals 2).*

## B. Solve at home before tutorial in week 6

1. Cormen et al., 4. Edition: Exercise 8.3-1 (page 214) [Cormen et al., 3. Edition: Exercise 8.3-1 (page 199)].

   Using Figure 8.3 as a model, illustrate RadixSort on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB.

2. (∗) Cormen et al., 4. Edition: problem 7-5 (page 202) [Cormen et al., 3. Edition: problem 7-4 (page 188)].

**Algorithm 1:** TRE-QUICKSORT$(A, p, r)$

```
while p < r do
    q = PARTITION(A, p, r)
    TRE-QUICKSORT(A, p, q − 1)
    p = q + 1
```

The QUICKSORT procedure of Section 7.1 makes two recursive calls to itself. After QUICKSORT calls PARTITION, it recursively sorts the low side of the partition and then it recursively sorts the high side of the partition. The second recursive call in QUICKSORT is not really necessary, because the procedure can instead use an iterative control structure. This transformation technique, called tail-recursion elimination, is provided automatically by good compilers. Applying tail-recursion elimination transforms QUICKSORT into the TRE-QUICKSORT procedure stated in Algorithm 1.

a. Argue that TRE-QUICKSORT$(A, 1, n)$ correctly sorts the array $A[1 : n]$.

b. Compilers usually execute recursive procedures by using a stack that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is called, its information is pushed onto the stack, and when it terminates, its information is popped. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires $O(1)$ stack space. The stack depth is the maximum amount of stack space used at any time during a computation.

   Describe a scenario in which TRE-QUICKSORT's stack depth is $\Theta(n)$ on an $n$-element input array.

c. Modify TRE-QUICKSORT such that the worst-case stack depth is $\Theta(\log n)$. Maintain the $O(n \log n)$ expected running time of the algorithm.

   *Hint: choose which part to call recursively instead of always using the left part.*