Institut for Matematik og Datalogi
Syddansk Universitet, Odense

20. Feburary 2025

Exercise Sheet Week 2

DSK814

The assignments for the practice classes (also called exam classes or e-classes) are divided into two groups:

A. A set of tasks to be solved *in the practice sessions* while the instructor is available for questions.

   These tasks must not be solved in advance, just make sure you have read the material from the lecture. Please feel free to work in your study groups during the lessons as it is a great help to solve the exercises together.

B. Another set of problems similar to the first set to be solved *at home*.

   The answers to these problems are briefly reviewed at the end of the practice lesson with regard to possible solutions, but the problems must be solved at home (preferably together with your study group) before the next tutorial. These tasks are then briefly summarized at the beginning of the next practice session.

Note that we mark slightly difficult tasks with $(*)$ and more difficult tasks with $(**)$. Due to the topic of asymptotic analysis of growth rate, the exercises in Part II are more mathematically than usual.

Experience shows that participants' familiarity with programming varies quite a bit. If you don't feel particularly strong in that area, it is important to spend some time on the given programming assignments. The exercises provided on this sheet will enable you to develop algorithms from the initial idea to pseudocode level to finished code. Furthermore, they are a good warm-up for the project in DM507/DS814. If you already have programming experience, we suggest putting more effort into the other assignments.

A(I). Solve in the practice sessions

1. Cormen et al., 4. Edition: Exercise 2.2-3 (page 33) [Cormen et al., 3. Edition: Exercise 2.2-3 (page 29)]. Answer the question for best-case running time as well. Note that the exercise references a previous problem for the description of linear search, you don't need to solve that problem, just read the short description of linear search.

2. Cormen et al., 4. Edition: Exercise 2.1-1 (page 24) [Cormen et al., 3. Edition: Exercise 2.1-1 (page 22)]. Just perform InsertionSort by hand, you don't need to illustrate it as it is done in the book.

3. Implement InsertionSort in Java or Python based on the pseudocode in Cormen et al., 4. Edition: page 19 [Cormen et al., 3. Edition: page 18]. Test your code by generating arrays/lists with different contents and sorting them. Note that the array indices in the pseudocode start with index 1, while Java and Python start with index 0. Therefore, remember to change it appropriately, i.e. sometimes use an index that is one less.

4. (∗) Cormen et al., 4. Edition: Problem 2-4 (page 47) [Cormen et al., 3. Edition: Problem 2-4 (page 41)]. Solve questions a, b and c.

   Hint for c: For the original input $A[1...n]$, let I denote the total number of inversions and d_i denote number of elements in $A[1..(i-1)]$ which are strictly larger than $A[i]$. Let t_i be defined as in Cormen et al., 4. Edition: lower half of page 29 [Cormen et al., 3. Edition: bottom of page 25 (where it is called t_j, because indices i are j swapped in the two editions)]. For $d_i = t_i - 1$ and $I = \sum_{i=2}^{n} d_i$, use the formula of Cormen et al., 4. Edition: upper half of page 30 [Cormen et al., 3. Edition: middle of page 26] to infer that the runtime is $O(n + I)$.

B(I). Solve at home before tutorial in week 3

1. Continue the task of implementing InsertionSort (i.e., A(I).3) as follows: Add timing to your code by inserting two calls to the method `System.currentTimeMillis()` for Java and `time.time()` for Python—one at the start and one at the end of InsertionSort. Look up the method functionality in the Java/Python online documentation and the code examples from the lectures this week. Only time measure the

sorting itself, not the part of the program that generates the array/list contents.

Let n be the number of items to sort. Run your code with increasingly sorted input and decreasingly sorted input (which are the best and worst case for InsertionSort) for at least 5 different values of n. Choose different values of n such that the program takes a time of 100 to up to 5000 milliseconds—note that the values are not the same for the best and worst case. Repeat each run 3 times and compute the average number of milliseconds as this will mitigate the fluctuations from background processes. Divide the resulting numbers by n (for best case input) and n^2 (for worst case input), respectively. Check if your numbers fit the theoretical analysis—according to the analysis, the resulting numbers should be constant (for the best and worst case numbers, separately), see the graphs on the slides from the lecture.

Run your code with random `int`'s as input. Use `java.util.Random` and its method `nextInt()` to generate them in Java. In Python, use `random.randint()` to generate them. Are the running times closer to the best case or the worst case?

2. (∗) Cormen et al., 4. Edition: Exercise 2.3-8 (page 45) [Cormen et al., 3. Edition: Exercise 2.3-7 (page 39)].

   Hint: Start by sorting the numbers. You can use the fact that this can be done in $O(n \log n)$ (e.g. with Mergesort).

A(II). Solve in the practice sessions

1. Cormen et al., 4. Edition: Exercise 2.3-1 (page 44) [Cormen et al., 3. Edition: Exercise 2.3-1 (page 37)].

2. For
   $$f(n) = 0.1n^2 + 5n + 25$$
   show that $f(n) = \Theta(n^2)$ and $f(n) = o(n^3)$.

   Hint: use propositions (1), (2) from the slides on analyzing algorithm running times.

3. Show that the following functions are sorted by increasing asymptotic growth rate:
   $$1,\ \log n,\ \sqrt{n},\ n,\ n \log n,\ n\sqrt{n},\ n^2,\ n^3,\ n^{10},\ 2^n$$

More precisely, show that for all pairs $f(n), g(n)$ holds that $f(n) = o(g(n))$.

Hint: use propositions (1)–(4) from the slides on analyzing algorithm running times.

4. (∗) Cormen et al., 4. Edition: Exercise 3.2-1 (page 62) [Cormen et al., 3. Edition: Exercise 3.1-1 (page 52)]. Here, use the actual definition of the Θ-notation from the book/slides, not propositions (1)–(4) from the slides.

B(II). Solve at home before tutorial in week 3

1. Cormen et al., 4. Edition: Exercise 2.3-6 (page 44) [Cormen et al., 3. Edition: Exercise 2.3-5 (page 39)]. Expand on the task as follows:
   (a) Illustrate the algorithm with a drawing
   (b) Create pseudocode for the algorithm (as in the task text).
   (c) Write a program in Java or Python based on your pseudocode.
   (d) Test the correctness of your implementation by generating arrays/lists (Java/Python) with the contents $1, 3, 5, 7, ...$ and performing searches. Test with empty arrays/lists, long arrays/lists, and try searching for numbers that are present/ not present in the array. Remember: do not cheat (yourself and your own learning) by looking at code from the web or elsewhere.
2. Cormen et al., 4. Edition: Exercise 2.3-7 (page 45) [Cormen et al., 3. Edition: Exercise 2.3-6 (page 39)].
3. Cormen et al., 4. Edition: Exercise 3.2-3 (page 62) [Cormen et al., 3. Edition: Exercise 3.1-4 (page 53)].

   Hint: remember the rules for calculating powers, see Cormen et al., 4. Edition: in the middle of page 65 [Cormen et al., 3. Edition: in the middle of page 55].
4. (∗) Cormen et al., 4. Edition: Exercise 3.3-4, question b (page 70) [Cormen et al., 3. Edition: Exercise 3.2-3 (page 60)].

   Hint: use the definition of $n!$ and general logarithm rules, see Cormen et al., 4. Edition: bottom of page 66 [Cormen et al., 3. Edition: bottom of page 56], especially the second rule.