Prof. Dr. Thomas Schultz
Olivier Morelle (morelle@uni-bonn.de)

B-IT
Bonn-Aachen International
Center for Information Technology
November 25, 2024

Winter term 2024/25

Computer Science for Life Scientists

Assignment Sheet 8

Solution has to be uploaded by December 04, 2024, 10:00 a.m., via eCampus

- This exercise can be submitted in **small groups** of 2-3 students. Submit each solution only once, but clearly indicate who contributed to it by forming a team in eCampus. Remember that all team members have to be able to explain all answers.
- Remember to include proper **documentation** in all your code, in particular, docstrings for functions.
- Please submit your answers as a single file in .ipynb or .pdf format.

If you have questions concerning the exercises, please use the forum on eCampus.

Exercise 1 (Correctness and running time analysis, *7 Points*)

Let A be an array with elements $A[1], \dots, A[n]$. Consider the following algorithm:

```
1: Algorithm(int[] A, int l, int r):
2: if $l < r$ then
3:     Find the minimum of the elements $A[l], \dots, A[r]$ and the corresponding index $i$.
4:     if $l \neq i$ then
5:         Swap $A[l]$ and $A[i]$.
6:     end if
7:     Algorithm(A,l+1,r)
8: end if
```

a) Which problem is solved by running `Algorithm(A,1,n)`? Prove your answer using mathematical induction. (3P)

b) How many comparisons are performed by `Algorithm(A,1,n)` when excluding the recursive call in line 7? How many comparisons are necessary overall, including all levels of recursion? Use Θ notation and simplify the term as much as possible. (2P)

c) How many times are array elements swapped when including the recursion (minimum, maximum)? Give an example input for both answers. (2P)

Exercise 2 (Greedy interval scheduling, *7 Points*)

In interval scheduling, we are given a set of tasks $\{j_1, j_2, j_3, \dots, j_n\}$ that each start at time s_j and finish at time f_j. A processor can only work on one task at a time. Please find provably optimal greedy strategies for the following two interval scheduling tasks, and implement them in Python. In your implementation, the problem instance should be described as a dictionary, whose keys are the job IDs and whose values are pairs of start and finish times. For example, one possible instance, illustrated in Figure 1, would be `{'a':(0,6),'b':(1,4),'c':(3,5),'d':(3,8),'e':(4,7),'f':(5,9),'g':(6,10),'h':(8,11)}`.

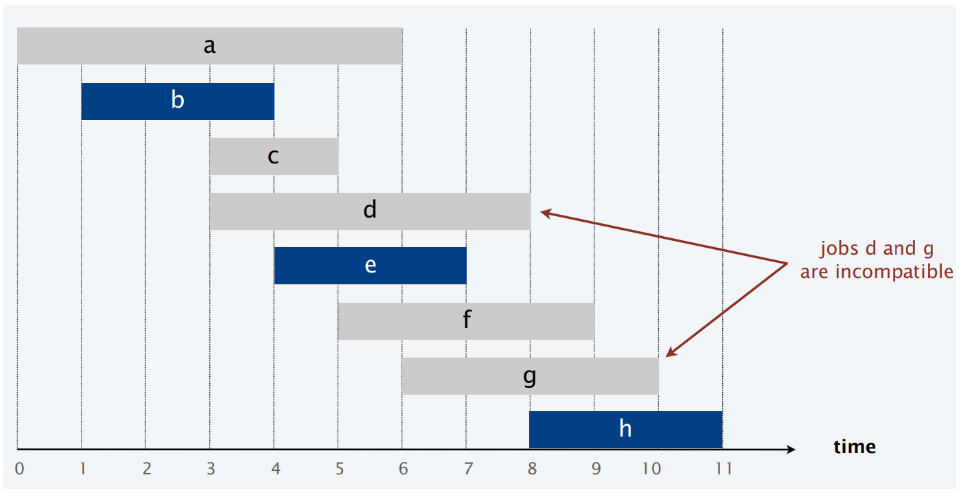

Figure 1: An illustration of interval scheduling, and what it means for two tasks to be incompatible.

a) Assume that only a single processor is available. Select a subset of tasks so that they are compatible (i.e., none of them overlap with each other), and that the number of selected tasks is optimal, i.e., any larger subset would contain at least one incompatible pair of tasks. (2P)

   Hint: As indicated in Figure 1, the optimal subset in this example would be $\{b, e, h\}$.

b) Find the minimum number of processors that is required to complete all tasks. Output a feasible assignment of tasks to this number of processors. (3P)

c) For each of the tasks above, present the time complexity and explain it. (2P)

Exercise 3 (Dynamic Programming, *11 Points*)

Searching for the longest common subsequence (LCS) of biological sequences is an important task in bioinformatics. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguously. For example, given a sequence S =“AGGTACCCGATC”, all “AGT”, “TAGAT”, “GGG”, are subsequences of S, but “TGCT” is not. In bioinformatics, the length of the longest common subsequence between two given sequences is a good measure for their similarity.

a) Write a recursive Python function to find the LCS of two input strings of size n and m. (3P)

b) For two sequences of size n, what is the running time complexity of your function in part a), where all possible subsequences are computed for both sequences? Briefly explain your answer. (2P)

c) This time, use dynamic programming to find the LCS of two input strings of size n and m. (4P)

d) Briefly explain what is the running time complexity of LCS function in part c). (2P)