Universität Konstanz

Introduction to Computation for the Social Sciences

Assignment 9

Prof. Dr. Karsten Donnay, Marius Giebenhain, Stefan Scholz
Winter Term 2019 / 2020

Please solve the exercises below and commit your solutions to our GitHub Classroom until Jan, 21st midnight. Submit all your code in one executable file (*py* / *ipynb*) and your text in one text file (*txt* / *md* / *pdf*). You can score up to 10 points in this assignment. You will get individual feedback in your repository.

Exercise 1: Quicksort Algorithm (2 Points)

In *Session 7*, we discussed the quicksort algorithm. In this exercise, we first review it again conceptually before we turn towards its implementation and an analysis of its complexity in the next two exercises.

a) Illustrate the logic of the sorting algorithm step-by-step using the sequence {5, 1, 3, 6, 7, 4, 8, 2}. Assume, without loss of generality, that the first element in each recursive step is the pivot.
b) Now consider the sequence {5, 1, 3, 1, 7, 4, 8, 2}. Illustrate in this example why the algorithm is not stable.

Exercise 2: Recursive Implementation of Quicksort (6 Points)

The quicksort algorithm implements the divide and conquer principle using the pivot element to (dynamically) partition the sequence into smaller and smaller slices. This can be done recursively until the smallest partition is already sorted.

a) Write a function `partition` in Python that robustly implements the step that ensures through re-shuffling that all elements left (right) of the pivot are smaller (larger) than the pivot. The function should return the pivot element to enable further recursive splits.

b) Now use this function to write a recursive implementation of `quicksort`. Ensure that the input sequence gets first randomly reshuffled as this improves average efficiency. Explain why this holds. Note that it is the simplest to write a separate "wrapper" function for the overall function call that only takes the sequence as input and one function that executes the actual recursion step for a given interval on the sequence.

Exercise 3: Complexity of Quicksort (2 Points)

In *Session 7*, we saw a quick estimation of the complexity of quicksort. Here we examine its worst- and best-case complexity in more detail.

a) Show that the worst-case complexity, i.e., the case where each interval is split as inefficiently as possible, is $O(n^2)$.

b) Now argue why the opposite case, i.e., the best case in which intervals are all evenly split, scales as $O(n \log n)$.

Note: The estimation of the average complexity of quicksort follows similar arguments but is much more involved. Hence, it is not part of this assignment…