Algorithms and Data Structures
Constructor University Bremen
Dr. Kinga Lipskoch

Course: CH-231-A
March 12th, 2024

Homework 6

- Submit one ZIP file per homework sheet which contains one PDF file (including pictures, computations, formulas, explanations, etc.) and your source code file(s) with one makefile and without adding executable, object or temporary files.
- The implementations of algorithms has to be done using C, C++, Python or Java.
- The TAs are grading solutions to the problems according to the following criteria: https://grader.eecs.jacobs-university.de/courses/ch_231_a/2024_1/Grading_Criteria_ADS.pdf

Problem 6.1 *Bubble Sort & Stable and Adaptive Sorting* (9 points)

(a) (3 points) $Bubble\ Sort$ is a sorting algorithm that works by repeatedly iterating through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This is repeated until no swaps are needed, which indicates that the list is sorted. Write down the $Bubble\ Sort$ algorithm in pseudocode including comments to explain the steps and/or actions.

(b) (2 points) Determine and prove the asymptotic worst-case, average-case, and best-case time complexity of $Bubble\ Sort$.

(c) (2 points) Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values). Thus, a sorting algorithm is **stable** if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list. Which of the sorting algorithms $Insertion\ Sort$, $Merge\ Sort$, $Heap\ Sort$, and $Bubble\ Sort$ are stable? Explain your answers.

(d) (2 points) A sorting algorithm is **adaptive**, if it takes advantage of existing order in its input. Thus, it benefits from the pre-sortedness in the input sequence and sorts faster. Which of the sorting algorithms $Insertion\ Sort$, $Merge\ Sort$, $Heap\ Sort$, and $Bubble\ Sort$ are adaptive? Explain your answers.

Problem 6.2 *Heap Sort* (13 points)

(a) (6 points) Implement the $Heap\ Sort$ algorithm as presented in the lecture.

(b) (4 points) Implement a variant of the $Heap\ Sort$ that works as follows: In the first step it also builds a max-heap. In the second step, it also proceeds as the $Heap\ Sort$ does, but instead of calling $MAX\text{-}HEAPIFY$, it always floats the new root all the way down to a leaf level. Then, it checks whether that was actually correct and if not fixes the max-heap by moving the element up again. This strategy makes sense when considering that the element that was swapped to become the new root is typically small and thus would float down to a leaf level in most cases. Hence, one would save the additional tests when floating down the element. And, the fixing step (moving the element upwards again) would be a rare case. This variant is called *Bottom-up Heap Sort*.

(c) (3 points) Compare the original $Heap\ Sort$ and its variant from subpoint (b) for input sequences of different lengths (including larger input sequences). What can you observe?

How to submit your solutions

You can submit your solutions via *Grader* at **`https://grader.eecs.jacobs-university.de`** as a generated PDF file and/or source code files.
If there are problems with *Grader* (but only then), you can submit the file by sending mail to `klipskoch@constructor.university` **with a subject line that starts with CH-231-A**.

Please note, that after the deadline it will not be possible to submit solutions. It is useless to send solutions by mail, because they will not be graded.

This homework is due by Monday, March 18th, 23:00.