Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

20th October 2016

Datenstrukturen & Algorithmen Exercise Sheet 5 AS 16

Hand-in: Thursday, 27th October 2016 before the start of the lecture at 10:00 in the entrance area of ML D28. Please staple all sheets together and use this sheet as the cover page. Fill out the first two fields of the form below.

Exercise class (Room & Day): _____

Submitted by: _____

Corrected by: _____

Bonus points: ____

Exercise 5.1 *Searching for equal numbers.*

Given two arrays A and B, where each array contains m, respectively n, pairwise different numbers. You should decide using an appropriate comparison procedure if there exist numbers that are both in A and in B.

a) Describe a naive procedure that finds all such pairs of numbers in unsorted arrays and describe the worst-case running time of this procedure by analyzing how many comparison operations are performed depending on the sizes of the arrays m and n.

b) Assume that A and B are sorted. Describe a procedure that solves the problem defined above in worst-case running time $\mathcal{O}(m+n)$.

Exercise 5.2 *Extended Heaps.*

In this exercise we are considering an array $A[1..|A|]$ representing a Min-Heap. We want to use A to maintain a set of n keys. Describe how the following operations can be implemented efficiently (i.e., with a running time in $\mathcal{O}(\log n)$).

a) MIN: Computes the smallest key.

b) REPLACE(i,k): Removes the key $A[i]$ and replaces it by k.

c) INSERT(k): Inserts a new key with value k in the heap.

d) DELETE(i): Removes the key $A[i]$ from the heap.

Note: Of course you have to make sure that the heap property is maintained after each operation. You can assume that A was chosen large enough to store all occurring keys.

Exercise 5.3 *Searching in sorted arrays.*

In the lecture you have seen binary search and interpolation search. While the former always uses at most $\mathcal{O}(\log n)$ steps, the latter needs $\mathcal{O}(\log \log n)$ steps on average, but $\Theta(n)$ steps in the worst case.

a) Specify an infinitely large family of examples (for every sequence length n), where interpolation search is faster than binary search and an infinitely large family of examples, where binary search is faster than interpolation search.

b) Interpolation search has a running time in $\mathcal{O}(n)$ in the worst case and a running time in $\mathcal{O}(\log(\log(n)))$ on average. How can you reach the worst case running time of binary search ($\mathcal{O}(\log n)$) and the average running time of interpolation search with the same procedure?