Exercise sheet 2.

Data structures SoSe 2020

László Kozma, Katharina Klost

Due 12:00, May 8th, 2020

Exercise 1 Heap variants 2×4 *Points*

(a) Design an *approximate heap* data structure that allows (i) inserting a key, and the operations (ii) *extract-small*, and (iii) *extract-large*, which extract (i.e. output and remove from the data structure) an arbitrary key that is among the smallest 25%, respectively largest 25% of the keys currently in the data structure. All operations should take constant amortized time. (Give a precise analysis and proof of correctness.)
Hint: A few stacks may be sufficient.

Bonus question +5p: A more general approximate heap takes a parameter ε (when it is created) and supports the operations (i) insert, and (ii) *extract(k)*, which, for an arbitrary parameter k, extracts a key with rank between $k - \varepsilon n$ and $k + \varepsilon n$. The operations should have amortized runing time $O(\log \frac{1}{\varepsilon})$. Design such a data structure.
Hint: A balanced binary search tree may be useful.

(b) Suppose that a priority queue is used in such a way that every extract-min operation returns a key that is at least as large as all previously extracted keys. Further assume that all keys are integers in $\{1, \ldots, K\}$. Design a priority queue that supports this kind of usage, in which an arbitrary sequence of m operations (insert, extract-min, or decrease-key) can be performed in total time $O(m + K)$.
Hint: An array may be useful.

Exercise 2 Sorted arrays 2×4 *Points*

(a) Recall the binary counter example in amortized analysis. Consider a variant where flipping the k-th least significant bit has cost 2^k. Suppose that the counter has n binary digits. Show that the amortized cost of an increment operation is $O(n)$.

(b) Suppose you implement a data structure that allows storing a set of keys, supporting the operations of *insert*ing a key into the data structure, and *search*ing, i.e. determining whether a key is currently in the data structure. The implementation is based on a collection of sorted arrays. The sizes of the arrays are powers of two and no two arrays have the same size.

Inserting creates a new array of size 1, then repeatedly merges pairs of arrays that have the same size, until all array sizes are distinct.

What are the amortized costs of insert and search?
Hint: the result of part (a) may be useful.

Exercise 3 Selection from sorted lists 4 *Points*

We are given t lists, each sorted in increasing order. Describe an efficient algorithm for finding the k-th smallest item over all lists. We assume that the items are pairwise distinct and that the total number of items is at least k. What is the best running time you can achieve in terms of t and k?

Total: 20 points. Have fun with the solutions!