University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithms and Data Structures
Summer Term 2019
Sample Solution Exercise Sheet 5

Exercise 1: Priority Queues

Consider the following priority queue stored in an array:

$$H = [(3, L), (10, D), (8, E), (12, C), (13, B), (23, R), (9, F), (17, S), (14, M)]$$

Execute the following operations on H: H.`insert`$((7, N))$, H.`deleteMin`$()$, H.`changeKey`$((13, B)$,9$)$. Write down H after each operation including the repairing process. It may help if you draw H as a binary tree.

Sample Solution

After H.`insert`$((7, N))$:

$$H = [(3, L), (7, N), (8, E), (12, C), (10, D), (23, R), (9, F), (17, S), (14, M), (13, B)]$$

After H.`deleteMin`$()$:

$$H = [(7, N), (10, D), (8, E), (12, C), (13, B), (23, R), (9, F), (17, S), (14, M)]$$

After H.`changeKey`$((13, B)$,9$)$:

$$H = [(7, N), (9, B), (8, E), (12, C), (10, D), (23, R), (9, F), (17, S), (14, M)]$$

Exercise 2: Amortized Analysis

Consider the data structure `stack` in which elements can be stored in a 'last in first out' manner. For a stack `S` we have the following operations:

- `S.push(`x`)` puts element x onto `S`.
- `S.pop()` deletes the topmost element of `S`. Calling `pop` on an empty stack generates an error.
- `S.multipop(`k`)` removes the k top objects of S, popping the entire stack if `S` contains fewer than k objects.

Assume the costs of `S.push(`x`)` and `S.pop()` are 1 and the cost of `S.multipop(`k`)` is $\min(k, s)$ where s is the current number of elements in `S`.

Use the bank account paradigm to show that we can assign all three operations constant amortized costs.

Sample Solution

Define the amortized costs of the operations as follows:

`S.push(`x`)`	2
`S.pop()`	0
`S.multipop(`k`)`	0

For a sequence of n operations let be c_i the actual cost and a_i the amortized cost of operation $i \leq n$. The total actual costs equals the number of `push` operations plus the number of `pop` operation, including calls within `multipop`. But there can be at most as many `pop` operations as `push` operations when the stack is initially empty, so the actual costs are at most twice the number of `push` operations, i.e., $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} a_i$.