Albert-Ludwigs-Universität, Institut für Informatik
Prof. Dr. Fabian Kuhn
M. Ahmadi, P. Schneider, J. Uitto

November 16, 2017

Algorithm Theory - Winter Term 2017/2018
Exercise Sheet 3

Hand in by Thursday 10:15, November 30, 2017

Exercise 1: Dynamic Programming - MaxIS (5+5 Points)

Let $G = (V, E)$ be a graph. A set of nodes $I \subseteq V$ is called *independent* if for all $u, v \in I$ it holds that $\{u, v\} \notin E$. In other words, *no* two nodes in the independent set I are adjacent. A *Maximum* Independent Set (MaxIS) is an independent set with maximum cardinality (number of nodes in I).

(a) Devise an *efficient*[1] algorithm that uses the principle of dynamic programming and finds a maximum independent set in a rooted tree[2].

(b) Prove that your algorithm is correct, i.e. returns a maximum independent set and prove that it has the claimed runtime.

Exercise 2: Worst Case Analysis - Fibonacci Heaps (3+7 Points)

Fibonacci heaps are efficient in an *amortized* sense. However, the time to execute a *single* operation can be quite large. Show that in the worst case, (a) the `delete-min` operation and (b) the `decrease-key` operation can require time $\Omega(n)$ *for an arbitrary* n.

Hint: Describe a valid scenario where the `delete-min` *or* `decrease-key` *operation respectively requires at least linear time in* n *for an arbitrary* n*.*

Exercise 3: Amortized Analysis - Counting (4+6 Points)

Consider non-negative integers in their canonical bit-representation. Similarly to the lecture, we execute n increment operations (add 1) starting from 0. For the analysis, we add a little twist. Flipping the i^{th} bit b_i[3] now has a cost 2^i.

(a) Show that the amortized cost is super-constant (i.e. in $\omega(1)$).

(b) Show that the amortized cost is $\mathcal{O}(\log n)$.

Exercise 4: Amortized Analysis - Multisets (10 Points)

Let S be a multiset of integers (a set which allows duplicate values). We would like to change S as follows. Either we insert a new element into S, or we delete the $\lceil |S|/2 \rceil$ largest elements from S (in case of removing some but not all the duplicates of the same element, we break the tie arbitrarily). Design a data structure that supports two operations `Insert`(S, x) and `DeleteLargerHalf`(S) such that any sequence of m operations starting from an empty multiset takes $O(m)$ time. Prove the correctness and amortized runtime of your implementation of `Insert`(S, x) and `DeleteLargerHalf`(S).

Hint: There is an algorithm to find the median of a multiset of n *integers in* $O(n)$ *time. You can use it as a blackbox.*

[1] An algorithm is efficient if it has a runtime of $\mathcal{O}\big(p(n)\big)$, where $p(n)$ is a polynomial.
[2] The input graph is connected, has no cycles, has a dedicated root node, and each node knows its parent and children.
[3] Bit b_i is the bit in the i^{th} position ascending from the least significant bit.