Technische Universität München
Fakultät für Informatik
Lehrstuhl für Algorithmen und Komplexität
Prof. Dr. Harald Räcke
Richard Stotz

Winter term 2018/19
Final exam
February 28, 2019

Efficient Algorithms and Data Structures I

Last Name	First Name	Matriculation No.	
....	....	....	
Hall	**Row**	**Seat No.**	**Signature**
....	....	....	....

General Information for the Examination

- Please keep your identity card readily available.
- Do not use pencils. Do not write in red or green ink.
- You are not allowed to use anything except a handwritten A4 sheet.
- Verify that you have received 8 pages, each printed on one side.
- You have 150 minutes to answer the questions.
- Unless noted otherwise, you must justify all your answers.
- It is best to explain algorithms by words. Otherwise use pseudocode.
- Do not write programs as these will cost you points.

Left Examination Hall from ...... to ...... / from ...... to ......
Submitted Early at ......
Special Notes:

	A1	A2	A3	A4	A5	A6	A7	Σ	Examiner
Max.	10	5	3	6	6	6	6	42	
1st									
2nd									

Question 1 (2+2+2+2+2 Points)

True or False? Justify your answer! Without justification, no points are awarded.

(a) If $T(n) = 8T(n/2) + n^2$ for sufficiently large n, then $T(n) \in \Theta(n^2)$.

(b) The left and right subtree of the root of a splay tree differ in height by at most a factor of 2.

(c) After 1000 inserts, the expected number of lists in a randomized skiplist is at least 10.

(d) Any 2-independent set of hash functions is a universal set of hash functions.

(e) Any bipartite graph $G = (L \cup R, E)$ with $|L| = |R|$ in which every node has degree at least 2, has a perfect matching.

Question 2 (2+3 Points)

(a) The recursion $T(n)$ is given by

$$T(n) = T(n/4) + T(n/16) + \sqrt{n} \ .$$

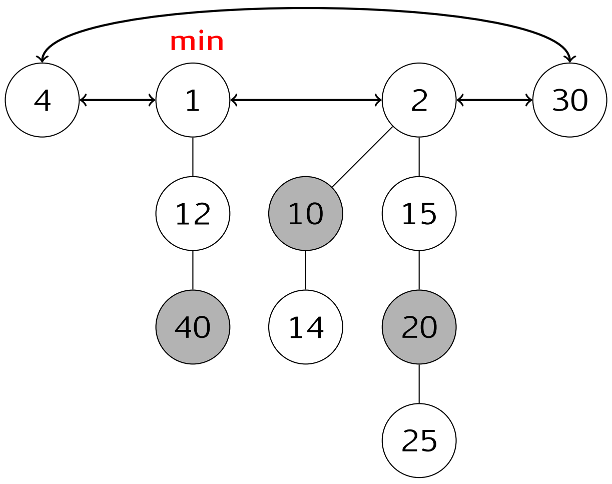
Assuming that $T(n)$ is constant for sufficiently small n, show that $T(n) \in \Theta(\sqrt{n})$.

(b) Solve the following recurrence relation using generating functions:

$$a_n = 2a_{n-1} - a_{n-2} + n + 1 \text{ for } n \geq 2, \quad a_0 = 1, a_1 = 4 \ .$$

Question 3 (1+1+1 Points)

Perform the following operations sequentially on the Fibonacci Heap shown below so that it remains a Fibonacci Heap. Show what the heap looks like after each operation (always carry out each operation on the result of the previous operation).
Nodes that are marked appear in gray.



(a) Insert(7)

(b) DeleteMin()

(c) Delete(25)

Question 4 (3+3 Points)

(a) Let $G = (L \cup R, E)$ be an unweighted bipartite graph. A matching M in G is *maximal*, if for any edge $e \notin M$, $M \cup \{e\}$ is not a matching.

Prove or disprove: Any maximal matching contains at least half as many edges as a maximum matching.

(b) Let $G = (V, E)$ be a directed graph with unit capacities, and suppose that for each node v the number of edges into v is equal to the number of edges out of v. That is, for all $v \in V$

$$|\{(u,v) \mid (u,v) \in E\}| = |\{(v,w) \mid (v,w) \in E\}|.$$

Let x, y be two nodes of G, and suppose that the capacity of the minimum $x-y$-cut is k.

Prove or disprove: The minimum $y-x$-cut also has capacity k.

Question 5 (2+4 Points)

(a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any ENQUEUE or DEQUEUE operation is $O(1)$.

(b) A quack is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that 3 operations are possible:

- QPUSH: add a new item to the left end of the list
- QPOP: remove the item on the left end of the list
- QPULL: remove the item on the right end of the list

Implement a quack using 3 stacks and $O(1)$ additional memory, so that the amortized time for any QPUSH, QPOP, or QPULL operation is $O(1)$.

For both parts of the question, you must clearly describe the implementation of your data structure. The only access you have to the stacks is through the standard subroutings PUSH and POP.
Prove that your implementation achieves the desired amortized running times **by using a suitable potential function**.
You do not need to show that your implementation works correctly.

Question 6 (4+2 Points)

We want to maintain a data structure D on a set of integers under the normal INIT, INSERT, DELETE, and FIND operations, as well as a SUBAVERAGE operation, defined as follows:

- INIT(D): Create an empty structure D.
- INSERT(D, x): Insert x in D.
- DELETE(D, x): Delete x from D.
- FIND(D, x): Return pointer to x in D.
- SUBAVERAGE(D, a): Return the average value of elements x in D with $x \leq a$.

(a) Describe how to modify a standard red-black tree in order to implement D, such that INIT is supported in $\mathcal{O}(1)$ time and INSERT, DELETE, FIND, and SUBAVERAGE are supported in $\mathcal{O}(\log n)$ time.

(b) Prove that your implementation reaches the required time bounds.

You do not need to prove that your implementation works correctly.

Question 7 (4+2 Points)

A library wants to move its collection to secondary facilities $F_1, \dots, F_f$. Facility F_j has capacity of b_j items and its access cost per item is v_j. The items in the collection are split into g groups $G_1, \dots, G_g$ according to their usage frequency. Group i contains a_i items. Extensive data collection shows that each item of group i will be requested r_i times during each timestep.
We need to find a distribution of the items to the facilities that minimizes the total access cost.

(a) Model this problem as a minimum cost flow problem. Describe precisely how your network is constructed and specify what a flow unit represents.

(b) Show that the simple rule that repeatedly assigns items with the greatest retrieval rate r_i to the facility with the lowest access cost v_j is an optimal strategy.

```
An algorithm must be seen to be believed.
                          - Donald E. Knuth
```