DM582 Exercises - Sheet 8

Mads Anker Nielsen Tobias Samsøe Sørensen
Safran Thestrup Preisel

April 7, 2026

This document contains exercises from the course DM582 (spring 2025). Most exercises are from the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein (CLRS), the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos (KT), and the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

References to CLRS refer to the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein.

References to KT refer to the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos.

References to Rosen refer to the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

References to BG refer to the book *Computer Algorithms: Introduction to Design and Analysis, 3rd edition* by Sara Baase and Allen Van Gelder.

This document will inevitably contain mistakes. If you find some, please report them to your TA so that we can correct them.

Sheet 8

Exercise from course website

Exercise

You know that red-black tree has many good properties. Now we'll add another: We'll show that rebalancing is amortized constant. So, yesterday you just knew that carrying out n operations would lead to at most $O(n \log n)$ rebalancing work. After today, you'll know that it's actually at most $O(n)$ rebalancing work.

Please see the sheet with a more compact representation of the red-black tree rebalancing operations. We refer to the operations in Figure 4 as (Ra) through (Rd) and the ones in Figure 5 as (Ba) through (Be) ('R' for "red" and 'B' for "black"). Half-colored nodes are either red or black. You already know that an insertion may create a red conflict (two consecutive red nodes) and that a deletion may create a black conflict (a doubly-black node - indicated by "-"). No operation creates additional problems but the red or black conflicts may be moved up in the tree until they disappear.

1. Call an operation *finishing* if it removes the conflict. Determine which operations are finishing. For (Ra), consider two cases, depending on whether the parent of the operation (that we don't see in the figure) is red or black.

2. Though (Bb) does not appear to be finishing, I would like to consider it finishing. Why?

3. Refer to a configuration in the tree where a black node has two black children as BBB and a configuration where a black node has two red children as BRR. We define the potential function $\Phi(T) = \#BBB + 2\#BRR$, where $\#BBB$ is the number of BBB configurations in T.

4. If there are k updates, how much can the insertions and deletions themselves plus all the finishing operations increase the potential? Conclude that if we can show that all non-finishing operations decrease the potential by at least one, then rebalancing is amortized constant.

5. Show that (Ba) decreases the potential - be careful with the sur-

roundings to make sure no new configurations in the potential function appears.

6. Show that the non-finishing case of (Ra) decreases the potential. Here you have to look quite a bit at what the surroundings must be.

CLRS, Problem 16-2

Hint: First consider how it would be good to produce a sorted sequence from sorted sequences of length 1, 2, 4, 8, etc.

Exercise

Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. You can improve the time for insertion by keeping several sorted arrays. Specifically, suppose that you wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \log_2(n+1) \rceil$, and let the binary representation of n be $(n_{k-1}, n_{k-2}, \ldots, n_0)$. Maintain k sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array A, is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.

a. Describe how to perform the `SEARCH` operation for this data structure. Analyze its worst-case running time.

b. Describe how to perform the `INSERT` operation. Analyze its worst-case and amortized running times, assuming that the only operations are `INSERT` and `SEARCH`.

CLRS, Problem 16-3

Note that even if you have trouble with some sub-question, you can without problem just assume a solution and continue with the following sub-questions.

Just a note for your information: Sub-question a can also be solved without using auxiliary storage but that's a little harder and not our focus here.

Exercise

Consider an ordinary binary search tree augmented by adding to each

node x the attribute $x.size$, which gives the number of keys stored in the subtree rooted at x. Let α be a constant in the range $1/2 \leq \alpha < 1$. We say that a given node x is α-balanced if $x.left.size \leq \alpha \cdot x.size$ and $x.right.size \leq \alpha \cdot x.size$. The tree as a whole is α-balanced if every node in the tree is α-balanced. The following amortized approach to maintaining weight-balanced trees was suggested by G. Varghese.

a. A 1/2-balanced tree is, in a sense, as balanced as it can be. Given a node x in an arbitrary binary search tree, show how to rebuild the subtree rooted at x so that it becomes 1/2-balanced. Your algorithm should run in $\Theta(x.size)$ time, and it can use $O(x.size)$ auxiliary storage.

b. Show that performing a search in an n-node α-balanced binary search tree takes $\Theta(\log n)$ worst-case time.

c. For the remainder of this problem, assume that the constant α is strictly greater than 1/2. Suppose that you implement INSERT and DELETE as usual for an n-node binary search tree, except that after every such operation, if any node in the tree is no longer α-balanced, then you "rebuild" the subtree rooted at the highest such node in the tree so that it becomes 1/2-balanced. We'll analyze this rebuilding scheme using the potential method. For a node x in a binary search tree T, define $\Delta(x) = |x.left.size - x.right.size|$. Define the potential of T as

$$\phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x)$$

where c is a sufficiently large constant that depends on α. Argue that any binary search tree has nonnegative potential and also that a 1/2-balanced tree has potential 0.

d. Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

e. Show that inserting a node into or deleting a node from an n-node α-balanced tree costs $O(\log n)$ amortized time.