University of Waterloo
CS240 - Spring 2020
Assignment 3

Due Date: Wednesday July 8, 5pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of July 8** along with your answers to the assignment – i.e. **read, sign and submit A03-AcInDe.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't "last minute").

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please refer to the course webpage for guidelines on submission. Submit your written solutions electronically as a PDF with file name a3Submit.pdf using MarkUs. We will also accept individual question files named a3q1.pdf, a3q2.pdf, . . . if you wish to submit questions as you complete them.

For all questions asking you to design an algorithm, you should include a description of the method, justification of correctness, and runtime analysis.

Problem 1 [7 marks]

Let A be an unsorted array of n integers in the range $[0, n^{42}]$. Design an algorithm that finds the minimum (non-negative) difference between any two numbers in this array. For instance, if the input was $[82, 32, 55, 78, 148]$, then the answer would be 4, witnessed by the pair 78 and 82. Your algorithm must take $O(n)$ time. It is important that your solution is explicit about how you represent the data.

Problem 2 [7 marks]

We want to prove the following: there is no comparison-based algorithm that can merge m sorted arrays of length m into a unique sorted array of length m^2 doing $O(m^2)$ comparisons. We argue by contradiction, and we assume that it is possible, so that we have such an algorithm (which we call FastMerge).

Modify MergeSort in order to use FastMerge, and derive a contradiction. You may need to solve the following recurrence relation: $T(n) = \sqrt{n}T(\sqrt{n}) + O(n)$.When solving this recurrence, you can disregard issues relating to the fact that $\sqrt{n}$ is not necessarily an integer. As a hint, the solution gives $T(n)$ to be $\in \omega(n)$ and $\in o(n \log n)$.

Problem 3 [4+4 marks]

This question is about insertion and deletion of elements in an AVL Tree.

1. Consider the AVL tree shown in Figure 1. Draw the tree again while replacing $b_0, b_1, \ldots, b_9$ with the respective balance factors (i.e., the height of the right subtree minus the height of the left subtree) on each node. A few have been filled in for you as an example.

   Perform the operation Insert(5) on the tree. Draw the tree before each rotation (including the intermediate tree in a double rotation), if any, and draw the final tree. Keep the balance factors updated up until any call to fix is required.

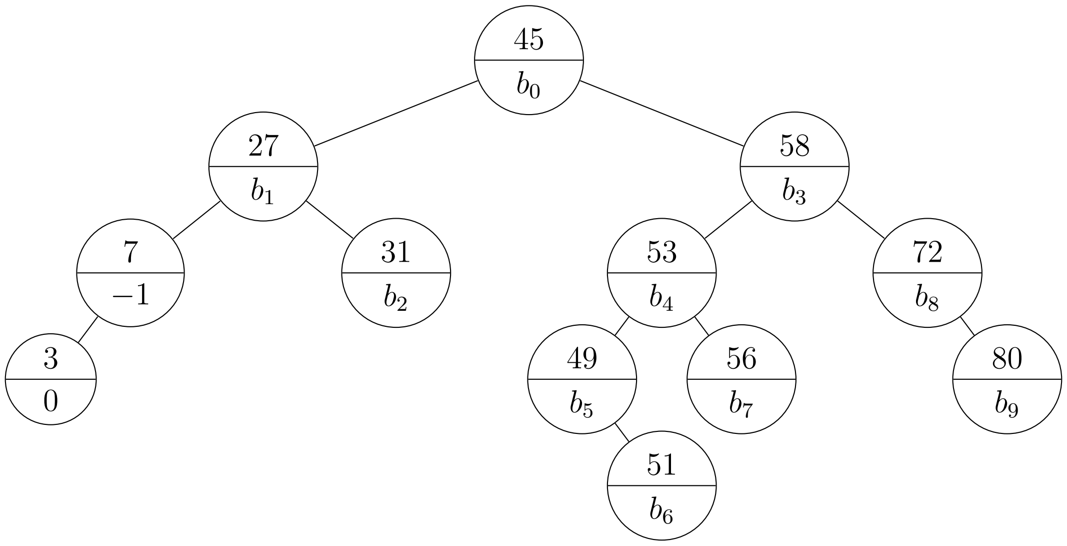

Figure 1: AVL tree of problem 3.

2. Consider the tree in Figure 1 (*not* the tree obtained after the insertion in part a). Perform the operation delete(27) on the tree, swapping with the inorder successor. Draw the tree before each rotation (including the intermediate tree in a double rotation), if any, and draw the final tree. Keep the balance factors updated up until any call to fix is required. Draw the final tree with balance factors.

Problem 4 [4+6 marks]

Suppose you wish to modify an AVL tree so as to support an operation `ithSuccessor`, in addition to the standard operations `insert, delete, find`. The operation `ithSuccessor` takes two parameters, x and $i \geq 0$, and returns the ith inorder successor of the node x. We will always assume that this successor exists (but maybe not in the subtree rooted at x). For $i = 0$, this is x itself.

We assume that the nodes have the following fields:

- `key` – the key of the node;
- `left` – pointer to the left child;
- `right` – pointer to the right child;
- `balance` – balance factor of the node;
- `parent` – pointer to the parent of the node;
- `isLeft` – is true if the node is a left child of its parent;
- `isRight` – is true if the node is a right child of its parent;
- `numLeft` – holds the number of nodes in the left subtree of the node;
- `numRight` – holds the number of nodes in the right subtree of the node.

1. Give an algorithm `ithNode(x, i)` which returns the ith inorder node in the subtree rooted at x, assuming that this subtree has at least i elements. Hence, $i = 1$ corresponds to the min element in the subtree, and $i = m$ (where m is the number of nodes in the subtree rooted at x) corresponds to the max element in the subtree. Your algorithm should take time $O(\log(m))$; don't forget to justify why.

2. Give the algorithm for `ithSuccessor(x, i)`, ensuring that this operation takes time $O(\log(n))$, if n is the number of nodes in the whole AVL (again, don't forget to justify why). Hint: determine if there are enough nodes in your right subtree; if so, find the ith successor there. Otherwise, you will have to go up the tree, and discuss whether x is a left or right child. You should use the algorithm `ithNode(x, i)` you just wrote.

Problem 5 [3+3+2+2+3+7 marks]

The goal of this problem is to show that deleting a single item in an AVL-tree of height h may require $\lfloor h/2 \rfloor \in \Omega(h)$ rotations.

Define a family $(T_h)_{h \geq -1}$ recursively in the following manner: T_{-1} is empty and T_0 is a single node; to form T_h, we start with a single node and take a copy of T_{h-2} and a copy of T_{h-1} as the left and the right children of the root, respectively.

1. Draw $T_0, \ldots, T_4$.

2. For $h \geq 0$, what is the height of T_h? Prove your claim.

3. Prove that for $h \geq 0$, T_h satisfies the balance requirements of an AVL tree.

4. On T_3, what are the leaves which require $\lfloor h/2 \rfloor = \lfloor 3/2 \rfloor = 1$ rotation upon deletion? Pick one and show the resulting tree.

5. Same question with T_4, but now with $\lfloor h/2 \rfloor = \lfloor 4/2 \rfloor = 2$ rotations.

6. Prove by induction that for $h \geq 0$, there exists a leaf ℓ_h in T_h such that deleting ℓ_h causes $\lfloor h/2 \rfloor$ rotations and the resulting tree has height $h-1$. Remember to ensure your solution has all the components of a complete inductive proof.

Problem 6 [4+6+4 marks]

1. Starting with an empty skip list, insert the seven keys $54, 15, 51, 53, 47, 68, 36$. Draw your final answer as on Slide 7 in Module 5. Use the following coin tosses to determine the heights of towers (note, not every toss is necessarily used):

$$T, T, H, H, T, H, T, H, H, T, H, H, T, T, H, T, H, H, T, T, H, H, H, T, \ldots$$

2. The worst case time for searching in a singly linked list is $\Theta(n)$. Now consider a variation of a skip list which has fixed height $h = 3$ even though n can become arbitrarily large. Level S_0 contains the keys $-\infty, k_1, k_2, \ldots, k_n, \infty$. Level S_3 contains only $-\infty$ and ∞. Describe subsets of keys that should be included in levels S_1 and S_2 so that searching in the skip list has worst case cost $\Theta(n^{1/3})$. Provide justification for the runtime of your skip list.

3. Consider a skip list in which we build new towers as follows:

> When adding an element to the skip list, we flip two coins at the same time, until we see at least one tail. The number of times we toss both coins and obtain two heads is the height of the tower.

Using the method for building heights described in the above quote, derive the expected height of any single tower (not the entire skip list). You may use the following equality without proof: for any $0 < p < 1$, $\sum_{i=0}^{\infty} ip(1-p)^i = \frac{1-p}{p}$.