Computer Science CSC263H
St. George Campus

January 30, 2020
University of Toronto

Homework Assignment #3

Due: February 13, 2020, by 5:30 pm

- **You must submit your assignment through the Crowdmark system.** You will receive by email an invitation through which you can submit your work. If you haven't used Crowdmark before, give yourself plenty of time to figure it out!
- You must submit a **separate** PDF document with for **each** question of the assignment.
- To work with one or two partners, you and your partner(s) must form a **group** on Crowdmark (one submission only per group). We allow groups of up to three students, submissions by groups of more than three students will not be graded.
- The PDF file that you submit for each question must be typeset (**not** handwritten) and clearly legible. To this end, we encourage you to learn and use the LaTeX typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- If this assignment is submitted by a group of two or three students, for each assignment question the PDF file that you submit should contain:
  1. The name(s) of the student(s) who *wrote* the solution to this question, and
  2. The name(s) of the student(s) who *read* this solution to verify its clarity and correctness.
- By virtue of submitting this assignment you (and your partners, if you have any) acknowledge that you are aware of the homework collaboration policy that is stated in the csc263 course web page: `http://www.cs.toronto.edu/~sam/teaching/263/#HomeworkCollaboration`.
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.
- The total length of your pdf submission should be no more than 4 pages long in a 10pt font.

Question 1. (1 marks) In this question, you must use the insertion and deletion algorithms as described in the "Balanced Search Trees: AVL trees" handout posted on the course web site.

a. Insert keys 22, 3, 4, 15, 25, 0, 8, 29, 19, 12, 7, 11 (in this order) into an initially empty AVL tree, and show the resulting AVL tree T, including the balance factor of each node (only the final tree should be shown).

b. Delete key 19 from the above AVL tree T, and show the resulting AVL tree, including the balance factor of each node.

In each of the above questions, only the final tree should be shown: intermediate trees will be disregarded, and not given partial credit.

Question 2. (1 marks) A real estate agency maintains information about a set of houses for sale. For each house it maintains a record (p, s), where p is the price and s is the floor area of the house. Each record of this type is called a *listing*. You must design a data structure D of listings that supports efficient implementation of the following operations:

INSERT(D, x): If x is a pointer to a new listing, insert the listing pointed to by x into D.

DELETE(D, x): If x is a pointer to a listing in D, remove that listing from D.

MAXAREA(D, p): Return the maximum floor area of listings whose price is at most p. If no listing has price at most p, then return -1.

The worst-case run time of each of the above operation should be $O(\log n)$ where n is the number of listings in the data structure D.

In this question, you must explain how to implement D as an *augmented AVL tree*, and explain how each of the above operation is executed. Since this implementation of D is based on a data structure and algorithms described in class and in the AVL handout, you should focus on the extensions and modifications needed here. Note that you can *not* assume that prices are distinct: there may be several house listings that have the *same* price (and may have different floor areas).

a. Give a precise and full description of your data structure. Illustrate this data structure by giving example of it on some collection of house listings of your own choice.

b. Explain how each of the above three operations can be implemented in $O(\log n)$ time and why, in each case, your algorithm achieve this time complexity. For the INSERT(D, x) and DELETE(D, x) operations, your explanation should be in clear and concise English. For the MAXAREA(D, p), you should give the pseudo-code.

Question 3. (1 marks)

Give an algorithm for the following problem. The input to the algorithm are two unsorted sequences $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ of n distinct positive integers each, and a positive number d. The algorithm should determine whether it possible to walk horizontally east for some distance $x \in X$ and then walk vertically north for some distance $y \in Y$, and end up distance d from where we started: if it is possible then the algorithm should output "YES", otherwise it should output "NO".

The ***expected*** running time of your algorithm should be $O(n)$, under some assumptions that we discussed in class.

HINT: What data structures or algorithms that we learned in class involve reasoning about probability?

Note: Do ***not*** assume that the input numbers are small, or they have a small range: they could be ***arbitrary***. So a solution based on a direct access table or on linear time sorting is ***not*** acceptable.

a. Describe your algorithm in clear and precise English, and then give the pseudo-code.

b. Explain why the ***expected*** running time of your algorithm is $O(n)$. Explicitly state every assumption that you need for your complexity analysis.

c. What is the ***worst-case*** running time of your algorithm? Use the Θ notation and justify your answer.

[***The questions below will not be corrected/graded. They are given here as interesting problems that use material that you learned in class.***]

Question 4. (0 marks) We say that a binary search tree *slightly balanced* if for every node, the height of its left subtree differs from the height of its right subtree (in absolute value) by at most 2. Recall the height of a tree is the number of edges in a longest path from the root to a leaf, and that we regard the height of an empty tree as being -1.

For every integer $h \geq 0$, let m_h be the smallest possible number of nodes that can be in a slightly balanced binary search tree of height h.

a. Give a recurrence relation for m_h and justify your answer. Make sure to state all the base cases.
b. Prove by induction that for every $h \geq 0$, $m_h \geq 1.4^h - 1$.
c. Use the result in **b.** to compute a good upper bound on the maximum height of any slightly balanced binary search tree with n nodes.

Question 5. (0 marks)

Consider an abstract data type that consists of a *set* S of integers on which the following operations can be performed:

ADD(i): Adds the integer i to S. If this integer already is in S, then S does not change

AVERAGE(t): Returns the average of all elements of S that are less than or equal to the integer t. If all the elements of S are greater than t, then return 0.

Describe how to implement this abstract data type using an augmented AVL tree T. Each operation should run in $O(\log n)$ worst-case time, where $n = |S|$. Since this implementation is based on a data structure and algorithms described in class and in the AVL handout, you should focus on describing the extensions and modifications needed here.
a. Give a precise and full description of your data structure. In particular, specify what data is associated with each node, specify what the key is at each node, and specify what the auxiliary information is at each node. In particular, what is (are) the augmented field(s), and what identity should this (these) fields satisfy. Illustrate this data structure by giving an example of it (with a small set of your own choice).
b. Describe the algorithm that implements each one of the two operations above, and explain why each one takes $O(\log n)$ time in the worst-case. *Your description and explanation should be in clear and concise English.* For the operation AVERAGE, you should also give the algorithm's high-level pseudocode.

Question 6. (0 marks)

You are tasked with designing a data structure that maintains a set of books for an online bookstore. A ***book*** is defined as a 3-tuple $(identifier, price, rating)$, where

- $identifier$ is a unique positive integer which identifies the ***book***.
- $price$ is a positive real number which represents the price of the ***book*** in dollars.
- $rating$ is a real number in the range $[0, 5]$, which represents how good the ***book*** is.

Note that you can *not* assume that *prices* and *ratings* are unique: there may be several ***books*** which have the same *price*, and there may be several ***books*** which have the same *rating*.

a. Name a standard data structure D that can store the set of ***books*** such that each of the following operations can be done in $O(\log n)$ time in the worst-case (where n is the number of ***books*** in D):

ADDBOOK(D, x): Insert x which is a new ***book*** of the form $(identifier, price, rating)$ into D.

SEARCHBOOK(D, id): Return the *price* and *rating* of the ***book*** in D whose $identifier$ is id. If there is no ***book*** in D whose $identifier$ is id, return NIL.

Briefly describe what each node u of your data structure contains, and what is the key that you are using when adding a new ***book***. Which standard operations of your data structure you are using to implement ADDBOOK and SEARCHBOOK?

In the Parts **b, c, d, e** below:

- For each tree that you define, describe what each node of your tree contains, and what is the key that you are using when inserting a new item into this tree.
- For each operation, you should give ***a clear and concise*** English description of the algorithm implementing the operation.
- Also give brief explanations of why each of your algorithms achieve the worst-case time complexity specified in the subquestion (where n is the number of ***books*** in the data structure).

b. Modify D to support the following operation, in addition to the operations of Parts **a**:

BESTBOOKRATING(D, p): Let r be the maximum *rating* among all ***books*** in D whose *price* is at most p. Then return r. If no ***book*** has *price* at most p, then return -1.

In addition to the English description of the algorithm implementing the above operation, ***you must also give the corresponding pseudocode.*** Also describe in English any changes you make to the implementation of the previously defined operations. The worst-case time complexity of each operation should be $O(\log n)$.

HINT: Use an *augmented* AVL tree, in addition to the data structure used to implement Part **a**.

c. Modify D to support the following operation, in addition to the operations of Parts **a, b**:

ALLBESTBOOKS(D, p): Let r be the maximum *rating* among all ***books*** in D whose *price* is at most p. Then return a pointer to a list of *all* ***books*** in D whose *rating* is r. If no ***book*** has *price* at most p, then return NIL.

Also describe in English any changes you make to the implementation of the previously defined operations (don't use pseudo-code). The worst-case time complexity of each operation should be $O(\log n)$.

HINT: Use another AVL tree, in addition to the data structures used to implement Parts **a, b**.

d. Modify D to support the following operation, in addition to the operations of Parts **a, b, c**:

INCREASEPRICE(D, p): Increase the *price* of *every* ***book*** in D by p dollars. Assume that p is a positive real number.

Also describe in English any changes you make to the implementation of the previously defined operations (don't use pseudo-code). The worst-case time complexity of the INCREASEPRICE operation should be $O(1)$. The worst-case time complexity of the previously defined operations should remain $O(\log n)$.