Computer Science CSC263H
St. George Campus

May 27, 2019
University of Toronto

Homework Assignment #2

Due: June 10 , 2019, by 11:59pm

- You must submit your assignment as a PDF file of a typed (**not** handwritten) document through the MarkUs system by logging in with your CDF account at:

  `markus.teach.cs.toronto.edu/csc263-2019-05`

  To work with one or two partners, you and your partner(s) must form a group on MarkUs.

- If this assignment is submitted by a group of two or three students, the PDF file that you submit should contain for each assignment question:

  1. The name of the student 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.

- The PDF file that you submit must be 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.

- 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: `https://github.com/csc263-summer2019/csc263-s19/wiki/Course-Information-Sheet#assignment-collaboration`.

- 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.

Question 1. (15 marks) In this question, you must use the insertion and deletion algorithms as described in the notes for AVL and red-black trees

a. (5 marks) Insert keys 15, 26, 24, 30, 12, 20, 22, 9, 17, 19, 16, 18 (in this order) into an initially empty AVL tree, and show the resulting AVL tree T, including the balance factor of each node.

b. (5 marks) Delete key 24 from the above AVL tree T, and show the resulting AVL tree, including the balance factor of each node.

c. (5 marks) Insert keys 15, 26, 24, 30, 12, 20, 22, 9, 17, 19, 16, 18 (in this order) into an initially empty red-black tree and show the resulting red-black tree T, include clearly the colour of each node as your output.

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. (15 marks) In this question, you must use the insertion and deletion algorithms as described in the notes for AVL and red-black trees

a. (5 marks) Prove or disprove: "For any AVL tree T and any key x in T, if we let $T' = \text{DELETE}(T, x)$ and $T'' = \text{INSERT}(T', x)$, then $T'' = T$."

- State clearly whether you are attempting to prove or disprove the statement.
- If proving, give a clear general argument;
- if disproving, give a concrete example where you show clearly each tree T, T', T'' with the balance factors indicated beside each node.

b. (5 marks) Find an AVL tree T and a key x in T such that calling $\text{DELETE}(T, x)$ causes **two** rebalancing operations to take place. Your tree T should be as small as possible, in terms of the number of nodes, so that this takes place (you do not have to prove that it is indeed the smallest).

- Show your original tree T and the key x,
- then show the result of each rebalancing operation.
- Make sure to clearly indicate the balance factor next to each node.

c. (5 marks) Prove or disprove: "Any red-black tree is also AVL Tree, that is that every red-black tree is height balanced"

- State clearly whether you are attempting to prove or disprove the statement.
- If proving, give a clear general argument;
- if disproving, give a concrete example where you show clearly a red-black tree that is not an AVL tree. ensure you label the colouring and balance factors for each node

Question 3. (15 marks) Give a simple, linear-time algorithm that determines if a Binary Search Tree (BST) satisfies the AVL balancing condition. The algorithm's input is a pointer to the root of a BST T where each node u has the following fields:

- an integer *key*,
- *left* and *right* which are pointers to the left and right children of u in T (if u has no left or right child, then $u.left$ = NIL or $u.right$ = NIL, respectively). **There is no balance factor or height information already stored in any node.**

The algorithm's output is TRUE if T satisfies the AVL balancing condition, and FALSE otherwise.

The worst-case running time of your algorithm ***must be*** $\Theta(n)$ where n is the number of nodes in T.

Describe your algorithm by giving its ***pseudo-code***, and explain why its worst-case running time is $\Theta(n)$.

Question 4. (20 marks) We want an efficient algorithm for the following problem. The algorithm is given an integer $m \geq 2$, and then a (possibly infinite) sequence of distinct keys are input to the algorithm, ***one at a time***. A *query* operation can occur at any point between any two key inputs in the sequence. When a *query* occurs, the algorithm must return, ***in sorted order***, the m smallest keys among all the keys that were input before the *query*. Assume that at least m keys are input before the first *query* occurs.

For example, suppose $m = 3$, and the key inputs and query operations occur in the following order:

$$20, 15, 31, 6, 13, 24, \textit{query}, 10, 17, \textit{query}, 9, 16, 5, 11, \textit{query}, 14, \ldots$$

Then the first *query* should return 6, 13, 15; the second *query* should return 6, 10, 13; the third *query* should return 5, 6, 9.

Describe a simple algorithm that, for every $m \geq 2$, solves the above problem with the following worst-case time complexity:

- $O(\log m)$ to process each input key, and
- $O(m)$ to perform each *query* operation.

Your algorithm must use a data structure that we learned in class without any modification to the data structure.

To answer this question, you must:

1. State which data structure you are using, and describe the items that it contains.
2. Explain your algorithm ***clearly*** and ***concisely***, in English.
3. Give the algorithm's ***pseudo-code***, including the code to process an input key k, and the code for *query*.
4. Explain why your algorithm achieves the required worst-case time complexity described above.

Solutions that are unclear, inefficient, or overly complex may not get any points.

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

Question 5. (0 marks)

From CLRS: 12.1-1, 12.2-1, 12.2-4, 12.2-7

Question 6. (0 marks)

The task in this question is to compute the medians of all prefixes of an array. As input we are given the array $A[1\,..\,n]$ of arbitrary integers. Using a heap data structure, design an algorithm that outputs another array $M[1\,..\,n]$, so that $M[i]$ is equal to the median of the numbers in the subarray $A[1\,..\,i]$. Recall that when i is odd, the median of $A[1\,..\,i]$ is the element of rank $(i+1)/2$ in the subarray, and when i is even, the median is the average of the elements with ranks $i/2$ and $i/2+1$. Your algorithm should run in worst-case time $O(n \log n)$.

a. Describe your algorithm in clear and concise English, and also provide the corresponding pseudocode. Argue that your algorithm is correct.

b. Justify why your algorithm runs in time $O(n \log n)$.