CSC265 F18: Assignment 2

Due: October 10, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources.
- Your submission should be no more than 7 pages long, in a single-column US Letter or A4 page format, using at least 9 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2018-09`
- This is a *group assignment.* This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (16 marks)

An IMPORTANCE TREE is a variant of a Binary Search Tree (BST), in which each element, in addition to its key, is also given an importance number, which is an integer. A valid IMPORTANCE TREE must satisfy all the properties of a BST, and, moreover, the importance number of any tree node must be greater than or equal to the importance numbers of its children: we call this the *importance property*.

In the following questions you can assume that in an IMPORTANCE TREE T, every node u has fields $u.key$, $u.imp$, $u.parent$, $u.lchild$, $u.rchild$, storing, respectively, the key, the importance number, and pointers to the parent, left, and right child of u.

Part a. (7 marks)

An IMPORTANCE TREE T supports an INSERT(key, imp) operation, which inserts into T a new element with key set to key and importance set to imp. Describe how to implement INSERT so that its running time is $O(h)$ where h is the height of T. Specify your algorithm in clear and precise English (and, optionally, pseudocode), and justify its running time and also why it preserves the properties of an IMPORTANCE TREE.

HINT: First insert as in any BST. Then use left and right rotations to make sure that the importance property is satisfied.

Part b. (9 marks)

Suppose you are given two IMPORTANCE TREE data structures T_1 and T_2, so that every element in T_2 has key greater than that of every element of T_1. Describe a procedure MERGE(T_1, T_2) which returns a new IMPORTANCE TREE T containing the union of the elements from T_1 and T_2. The procedure should run in time $O(h_1+h_2)$, where h_1, h_2 are, respectively, the heights of T_1 and T_2. Specify your algorithm implementing MERGE in clear and precise English (and, optionally, pseudocode), and justify its running time and also why it returns a valid IMPORTANCE TREE T containing the elements of T_1 and T_2. You can assume that MERGE takes pointers to the roots of T_1 and T_2 as input, and returns a pointer to the new merged tree.

Question 2. (12 marks)

Recall the Quicksort algorithm. In particular, we will use the QUICKSORT(A, p, r) algorithm described in Chapter 7 of the textbook. This algorithm takes an unsorted array $A[1..n]$ of integers, and two indexes p and r. At completion it has sorted the subarray $A[p..r]$. For completeness, the algorithm is given below. (This pseudocode is the same as in the textbook.)

```
QUICKSORT($A, p, r$)
1  if $p < r$
2      $q$ = PARTITION($A, p, r$)
3      QUICKSORT($A, p, q - 1$)
4      QUICKSORT($A, q + 1, r$)
```

```
PARTITION($A, p, r$)
1  $x = A[r]$
2  $i = p - 1$
3  for $j = p$ to $r - 1$
4      if $A[j] \leq x$
5          $i = i + 1$
6          exchange $A[i]$ with $A[j]$
7  exchange $A[i + 1]$ with $A[r]$
8  return $i + 1$
```

Here, the PARTITION(A, p, r) procedure selects $x = A[r]$ as the ***pivot***. It rearranges the subarray $A[p..r]$ so that all its entries which have a key less than or equal to x are placed before x, and all entries which have a key greater than x are placed after x. Then it returns q which is the new index of x after the rearrangement. See the book for more details.

For the purposes of this question, when QUICKSORT(A, r, r) is called, we consider $A[r]$ as the pivot, even though PARTITION is not called in this case. With this definition, you can check that every element of A becomes a pivot at some point during the execution of QUICKSORT($A, 1, n$).

Important Observation. Notice that the only time QUICKSORT(A, p, r) compares two entries of A is during the PARTITION procedure when every element of $A[p\,..\,r-1]$ is compared with the pivot $x = A[r]$.

Assume that all entries of A are distinct Let us list the the entries in the array A as $x_1, \ldots, x_n$, in the order in which they become pivots during an execution of QUICKSORT$(A, 1, n)$. Give x_i an importance score $n - i + 1$, so that the first element that becomes a pivot has score n, the next one $n - 1$ and so on. Let T be a IMPORTANCE TREE (as defined in the previous question) whose elements have $x_1, \ldots, x_n$ as keys, and the element with key x_i has importance score $n - i + 1$ as just defined.

Give an algorithm which, given as input *only* the IMPORTANCE TREE T defined above, computes in worst-case running time $O(n)$ the exact number of comparisons between entries of A performed by QUICKSORT$(A, 1, n)$. Describe your algorithm in clear and precise English, and also using pseudocode. Justify why it correctly computes the number of comparisons, and why it runs in time $O(n)$.