CSC265F16: Homework Assignment #1

Due: October 4, by midnight

September 20, 2016

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.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2016-09`. *Note:* MarkUs will be populated with your usernames on Friday, September 23. Do not attempt to log on before that date. After that date, use your CDF account to log on to MarkUs.
- 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.
- One student in your group is responsible for writing the solution, and the other student is responsible for proof-reading and revising it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of the group member who wrote the solution, and also the *name*, *student ID*, and the *UTOR email address* of the group member who proof-read and revised it, in order for the submission to be accepted. *Make sure to specify which group member wrote the solution, and which group member 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, either from the textbook, or from a handout. 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, similarly to how algorithms and data structures are analyzed in the textbook. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (20 marks) Let $n = 2^k$. In this question you will design a data structure for the Partial-Sums abstract data type (ADT). The mathematical object in the Partial-Sums ADT is a sequence $a_1, \ldots, a_n$ of integers, and the ADT supports the following three operations:

- INIT initializes the data type to $a_1 = \ldots = a_n = 0$.
- ADD(i, t) sets a_i to $a_i + t$, where $1 \leq i \leq n$ and t is an integer;
- SUM(i) returns $a_1 + \ldots a_i$, where $1 \leq i \leq n$.

Your goal in this question is to design a data structure implementing the Partial-Sums ADT, so that INIT runs in worst-case time $O(n)$, and both ADD(i, t) and SUM(i) run in worst-case time $O(\log n)$.

HINT: Think of a complete binary tree with the a_i stored in the leaves. What do you store in the internal nodes?

a. Describe in detail a data structure implementing the Partial-Sums ADT. Draw an example with $n = 8$, and values of $a_1, \ldots, a_8$ of your choosing (but not $a_1 = \ldots = a_8 = 0$).

b. Describe in clear and precise English how to implement the three operations INIT, ADD(i, t), and SUM(i). Give pseudocode for ADD(i, t) and SUM(i). For each operation argue why it computes the correct output, and show that it runs in the required worst-case running time:

- INIT must run in time $O(n)$
- ADD(i, t) must run in time $O(\log n)$
- SUM(i) must run in time $O(\log n)$

c. For each $n = 2^k$, and each setting of $a_1, \ldots, a_n$, give an input i, $1 \leq i \leq n$, such that your implementation of SUM(i) runs in time $\Omega(\log n)$. Explain clearly and precisely why the running time of the algorithm is $\Omega(\log n)$ on this input.

Question 2. (20 marks) An R-HEAP is a *mergeable* heap data structure, which is an alternative to the binomial heaps that we saw in lecture. The data structure is shaped like a rooted binary tree: each node in the tree has at most 2 children. As usual, with each node u of the tree, we store the pointers $u.parent$, $u.left$, and $u.right$, respectively to the parent, left child, and right child of u. If u is the root of tree, then $u.parent = u$. If u has no left child, then $u.left =$ NIL, and, analogously, $u.right =$ NIL when u has no right child.

Each node of the tree is associated with an element of the heap, and, in addition to the pointers mentioned above, stores two values: the key of the element, in $u.key$, and a d-value, in $u.d$, which is defined recursively as follows:

- for any node u with at most 1 child, $u.d = 1$
- for any node u with 2 children, v and w, $u.d = 1 + \max\{v.d, w.d\}$.

The heap is required to satisfy the following two properties:

I. (*Min-Heap Order*) For any node u, and any child v of u, we require that $u.key \leq v.key$.

II. (*R-balance*) For any node u:

i. if u has only one child, it must be a right child

ii. if u has a left child v and a right child w, then $v.d \leq w.d$

Recall that the UNION(H_1, H_2) operation on a mergeable heap takes two heaps H_1 and H_2 (in our case, two R-HEAP data structures) and returns an R-HEAP that contains the union of the elements of H_1 and H_2.

a. Give an algorithm for UNION(H_1, H_2) running in worst-case time complexity $O(r_1.d + r_2.d)$, where H_1 and H_2 are R-HEAP data structures, r_1 is the root of H_1, r_2 is the root of H_2, and H_1 and H_2 are given to the algorithm by pointers to the roots r_1 and r_2, respectively. UNION(H_1, H_2) must return a pointer to the root of an R-HEAP that contains the union of the elements of H_1 and H_2. Describe your algorithm in clear and precise English, and give pseudocode. Show that the running time of the algorithm is $O(r_1.d + r_2.d)$.

HINT: Recursively take the union of one of the two heaps with a subtree of the other.

b. Show that for any n-node R-HEAP H with root node r, we have $r.d = O(\log n)$.

c. Recall that the height of a tree T with root r equals the number of edges in the longest path from r to a leaf of T. What is the largest possible height of an R-HEAP with n nodes?