CS2020: Data Structures and Algorithms (Accelerated)

Discussion Group Problems for Week 4

For: February 9/10, 2011

Problem 1. (Height of a Heap)

Problem 1.a. What is the *minimum* number of elements in a heap (stored in memory as a complete binary tree) of height h? (Recall that height is 0-based.)

Problem 1.b. Show that an n-element heap has height $\lfloor \log_2 n \rfloor$.

Problem 2. (Structure of a Heap)

We know that in a Max Heap, the maximum element is at the root. Where do you find the minimum element? (Assume that all the elements are distinct.) Is it possible to efficiently search for the smallest element in a heap?

Problem 3. (Max Heap Operations)

Problem 3.a. Insert {5, 3, 2, 7, 6} in an initially empty Max Heap one at a time (a.k.a `Build_Heap` v1 shown in the lecture).

Problem 3.b. Execute `Build_Heap` v2 on the same sequence of numbers. (Recall that this version of `Build_Heap` constructs the heap more efficiently than inserting each element one at a time.)

Problem 3.c. Do you get *the same* Max Heap for both of the previous two methods of constructing a heap?

Problem 3.d. Repeat the above exercise (constructing the heap using the two different version of `Build_Heap`) for the following sequence of numbers: {5, 3, 2, 7, 6, 8, 4}. Now, do you get the same Max Heap? Why?

Problem 3.e. Perform `Heap_Sort` step by step on the Max Heap using the previous sequence of numbers. If there is more than one possible Max-Heap from the previous part, explore how `Heap_Sort` runs differently on the two different Max Heaps.

Problem 4. (Stable Sorting)

Recall that a stable sorting algorithm is one identical elements are not reordered. For example, if an array A contains two copies of the key 2 (which we identify as $2a$ and $2b$ do disambiguate) where $2a$ precedes $2b$, then in the array output by the sorting algorithm, $2a$ still precedes $2b$.

Is `Heap_Sort` a stable sorting algorithm? For example, if we have these numbers {5a, 2, 3a, 3b, 5b, 5c} where 'a/b/c' is to 'differentiate' the duplicate numbers, can `Heap_Sort` guarantee that the output will always be {2, 3a, 3b, 5a, 5b, 5c}?

Prove that `Heap_Sort` is stable or find a counter example if it is not.

Problem 5. (Fuzzy Sorting)

In this problem, instead of sorting integers, we are going to sort intervals. A (closed) interval consists of a pair of integers $\langle a, b\rangle$ where $a \leq b$. The interval $\langle a, b\rangle$ contains every real number in the set $\{k : a \leq k \leq b\}$.

Given two intervals $x = \langle a, b\rangle$ and $y = \langle c, d\rangle$, we say that $x < y$ if $b < c$. We say that $y < x$ if $d < a$. Otherwise, if neither $x < y$ nor $y < x$, then we say that x and y are overlapping.

Consider the following sorting problem. You are given as input a sequence of n distinct intervals $\{\langle a_1, b_2\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_n, b_n\rangle\}$. You may assume that no interval contains another interval, i.e., if $a_i \leq a_j$, then $b_i \leq b_j$. The goal is to output a permutation $\langle i_1, i_2, \ldots, i_n\rangle$ of the intervals such that for all j, there exists a $c_j \in [a_{i_j}, b_{i_j}]$ satisfying $c_1 \leq c_2 \leq \cdots \leq c_n$. Notice that this is equivalent to outputting a permuation of intervals where, for each $j < n$, either: $i_j < i_{j+1}$, or i_j and i_{j+1} are overlapping.

First, give an $O(n \log n)$ solution to the interval-sorting problem. Next, suppose that each interval in your input is guaranteed to overlap at least $d-1$ other intervals. Given an $O(n \log(n/d))$ algorithm to sort the n intervals. Thus, if $d = \Theta(1)$, the algorithm runs in $O(n \log n)$ time. If $d = n$, i.e., if all the intervals overlap, then the algorithm runs in $O(n)$ time. (For this problem, do not worry about space usage. Your sorting algorithm does not have to be in-place.)