COP-4534: Algorithm Techniques

Homework 3

DUE: Sunday October 22 at 11:55 PM

- Please remember that all submissions must be typeset. Handwritten submissions will NOT be accepted. These must be uploaded to SCIS moodle in PDF format only.

- Please remember to type your name on top of your submission.

1. (20 points) In the AVL tree data structure, we need to add an extra field height($x$) to bookkeep the height of each node $x$ so that whenever a node becomes unbalanced, we can perform rotations to correct this. How much space is required for this field for each node? Can you improve the space complexity for each node to $O(1)$?

2. (20 points) Design a data structure which stores a sorted list of elements and supports the following operations:
   - INSERT($x$): insert a new element $x$ into the list
   - DELETE($x$): delete an existing element $x$ from the list
   - QUERY($x$): search in the list for element $x$
   - RANK($x$): find the rank of element $x$ in the list ($x$ can be already in the list or a new element)
   - SELECT($k$): select the $k$th smallest element in the list

3. (20 points) Given an array of $n$ integers, how to find the maximum element with as few comparisons as possible? What if we need to find both the maximum and the second maximum elements? What about the $k$ largest elements in the array? (Hint: use extra space to build a tree)

4. (20 points) Suppose you are given an unsorted list of $n$ distinct numbers. However, the list is close to being sorted in the sense that each number is at most $k$ positions away from its original position in the sorted list. For example, suppose the sorted list is in ascending order, then the smallest element in the original list will lie between position 1 and position $k + 1$, as position 1 is its position in the original sorted list. Design an efficient algorithm that sorts such a list. The running time of your algorithm should depend on both $n$ and $k$ (a helpful sanity check whether your running time bound makes sense is that, when $k$ is close to $n$, then the input list is just a generic unsorted list).

5. (20 points) Let $H_1$ and $H_2$ be two (binary) max-heaps with $n_1$ and $n_2$ elements respectively. If every element in $H_1$ is larger than every element in $H_2$, design an algorithm that merges these two heaps into one (binary) max-heap in $O(n_2)$ time (assume that both $H_1$ and $H_2$ are large enough to hold $n_1 + n_2$ elements).