Recitation Apr 12

What does each removeMin call return within the following sequence of priority queue ADT operations: insert(5, A), insert(4, B), insert(7, F), insert(1, D), removeMin( ), insert(3, J), insert(6, L), removeMin( ), removeMin( ), insert(8, G), removeMin( ), insert(2, H), removeMin( ), removeMin( )?

At which positions of a heap might the third smallest key be stored?

At which positions of a heap might the largest key be stored?

Consider a situation in which a user has numeric keys and wishes to have a priority queue that is *maximum-oriented*. How could a standard (min-oriented) priority queue be used for such a purpose?

Is there a heap H storing seven entries with distinct keys such that a preorder traversal of H yields the entries of H in increasing or decreasing order by key? How about an inorder traversal? How about a postorder traversal? If so, give an example; if not, say why.

Let H be a heap storing 15 entries using the array-based representation of a complete binary tree. What is the sequence of indices of the array that are visited in a preorder traversal of H? What about an inorder traversal of H? What about a postorder traversal of H?

Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$. What is the running time of this version of quick-sort on a sequence that is already sorted?

Consider a modification of the deterministic version of the quick-sort algorithm where we choose the element at index $\lfloor n/2 \rfloor$ as our pivot. Describe the kind of sequence that would cause this version of quick-sort to run in $\Omega(n^2)$ time.

Is the bucket-sort algorithm in-place? Why or why not?

Suppose S is a sequence of n values, each equal to 0 or 1. How long will it take to sort S with the merge-sort algorithm? What about quick-sort?

Suppose S is a sequence of n values, each equal to 0 or 1. How long will it take to sort S stably with the bucket-sort algorithm?

Given a sequence S of n values, each equal to 0 or 1, describe an in-place method for sorting S.

Give an example input that requires merge-sort and heap-sort to take $O(n \log n)$ time to sort, but insertion-sort runs in $O(n)$ time. What if you reverse this list?