Problem Set 3
(due Monday, October 26)

Problem 1 Suppose that you have a (max) heap, a number \(x\), and a positive integer \(k\). Design an algorithm to determine whether the \(k\)th largest item in the heap is greater than, less than, or equal to the number \(x\). Make sure your algorithm runs in \(O(k)\) time regardless of how big the heap is.

*Hint:* You do not need to find the \(k\)th largest item; you need only determine its relationship with \(x\).

Problem 2 A \(d\)-ary (max) heap is like a binary (max) heap, but instead of 2 children, nodes have \(d\) children.

(a) How would you represent a \(d\)-ary heap in an array?

(b) What is the height of a \(d\)-ary heap of \(n\) elements, in terms of \(n\) and \(d\)?

(c) Give an efficient implementation of Extract-Max. Analyze its running time in terms of \(d\) and \(n\).

(d) Give an efficient implementation of Insert. Analyze its running time in terms of \(d\) and \(n\).

(e) Give an efficient implementation of \textsc{Heap-Increase-Key}(\(A, i, k\)), where \(A[i] \leq k\), which sets \(A[i] := k\) and updates the heap structure appropriately. Analyze its running time in terms of \(d\) and \(n\).

Problem 3 Suppose that you have a sequence of \(n\) items, where the size \(s_i\) of the \(i\)th item satisfies \(0 \leq s_i \leq 1\). The bin-packing problem is to pack the items into bins of size 1 (the fewer bins, the better). Each bin can hold a subset of items whose total size is at most 1.

Consider the following strategy for bin packing, called the first-fit heuristic: Take each item in turn, and place it into the first bin that it will fit into.

A naive implementation of first-fit packing would take \(\Theta(n^2)\) time, because each insertion could take \(\Theta(n)\) time. Describe a faster implementation for achieving the same first-fit packing, which takes only \(O(n \log n)\) time.

*Hint:* Keep track of the current capacity of each bin. Imagine that these capacities are the leaves of a tree. Now imagine a “tournament” among these capacities.