

CMPS 102 Homework 3

2 study problems, Turning in by April 26 is optional

Reading: Chapters 6, 7, Section 28.2, Sections C.1, C.2, C.3.

1. In class before the midterm we discussed building heaps from n elements using a divide and conquer approach. The basic operation is “check that a root has more priority than its children and swap if necessary.” We had used the function $T(n)$ defined by $T(1) = 0$, $T(2) = T(3) = 1$ and for $n > 1$, $T(n) = 2T(\lfloor n/2 \rfloor) + \lg n$ as an upper bound on the number of these basic operations needed to build an n element heap. However, this analysis assumed that every parent’s two subtrees are equal size (or equivalently, that the heap was a complete binary tree - pg. 1090 of the text).

First, if a heap contains n nodes, what is the largest fraction of n that can appear in a single subtree? Justify your answer.

Next, consider the following way of building a heap of n elements. If $n = 2^k - 1$, then the heap is complete binary tree and we can use the divide and conquer process discussed in class. On the other hand, if $2^{k-1} - 1 < n < 2^k - 1$ then we can add $2^k - 1 - n$ “dummy” items with lower priority than any possible “real” items, creating a new $2^k - 1$ element build-heap problem. This technique of adding dummy elements to get a convenient input size is called “padding”. Show that building a heap with padding requires at most $2n$ basic three-way comparison operations (where n is the number of *original*, unpadding elements). (Hint: analyze $G(k)$, the time it takes to heapify $2^k - 1$ elements, and show that $G(k)$ is at most something like $2^k - ak - b$ for some values of a and b).

2. Assume that you have an array of n objects, and want to determine which of the objects (if any) is a “majority element”, occurring at least $\lfloor n/2 \rfloor + 1$ times in array. These objects are not ordered, so “ $<$ ” or “ $>$ ” comparisons are not possible, but you can test two objects for equality. Discover and analyze a divide and conquer algorithm for this problem that uses $O(n \lg n)$ equality tests on the objects.

(Hard!) For extra credit (on an optional assignment??), find a divide and conquer algorithm using only $O(n)$ equality tests. For the extra credit part, you can assume that n is a convenient size (i.e. c^k for some small integer c). The algorithm I have in mind does not strictly fit the “divide and conquer” template, as the outermost recursion does some extra work that is not done at every level.