

Final Examination

2007 March 22, pm3:15-5:15

- This is an open text and open notes exam.
 - Write your answers on your own paper. The instructor will have blank paper in case you have none.
1. The max-heap implementation in the text is that of a binary heap. We wish to generalize that to *d-ary heaps*. For example, a *ternary heap* is a *d-ary heap* where $d = 3$. In a ternary heap, every node (*i*) has one key and $d = 3$ children and (*ii*) satisfies the max-heap property - the key stored there is at least as large as any key stored in its children.
 - (a) If a ternary heap is stored in an array, show how, given an index *i* of a node, to find that node's left, middle, and right children.
 - (b) What is the height of a ternary tree? What is the height of a *d-ary tree* (in terms of *d* and *n*)?
 - (c) Describe how to perform an insertion in a ternary heap.
 - (d) Think about the `extractMax` method (you do not need to describe it). How long does it take for a ternary heap? For a *d-ary heap*?

[13 points]

2. Here is a lower bound problem for your consideration.
 - (a) Professor Helifino claims to have found a wonderful way to search for an element in a sorted list - *ZippySearch*. At $O(\lg \lg n)$ time it beats binary search, and is comparison based, Prof. H says. Explain why this cannot be the case.
 - (b) What do you get when you cross a rhinoceros with an elephant?

[6 points]

3. Consider a binary tree (not a binary search tree), each node of which contains the fields *key*, *weight*, *lchild*, and *rchild*. The first two are of type `int`, the other two are pointers. This problem will use only the *weight* field, so from now on we ignore the *key* field.

The problem here is to find the heaviest weight path from the root to a null node. That is, we want to start at the root, and proceed to a leaf, adding up the weights. What is the largest value you can get from a given tree going from top to bottom?

Write a routine which, given (the root to) a tree *T*, returns the total weight of the heaviest path. (*Don't read this subliminal message: recursion, $O(n)$, short code.*)

[10 points]

4. Suppose that everyone in class has submitted their favorite picture (say as a jpeg file), which has then been put into an array A of size n . Because they are not comparable elements, we cannot say whether $A[i] \leq A[j]$, however let's suppose that we can tell (in $O(1)$ time) whether $A[i] = A[j]$.

Our goal is to see whether any one picture got the *majority* ($> \frac{n}{2}$) of the votes. For comparable elements, we could just find the median, count its votes, and return the median if it won a majority. Here though, the median is not well-defined.

Find an $O(n)$ method to find the majority winner (if there is one).

Hint:

- pair up elements of A arbitrarily
- compare each pair - if they are not equal, get rid of both; if they are equal, keep one
- argue that after one round at most $n/2$ elements remain
- continue until only one element is left
- argue that if there is a majority element, it will remain at the end
- don't forget to show you are $O(n)$ time

[11 points]

5. Regarding binomial heaps, these are to be min-heaps (as in the text).
- Into an initially empty binomial heap, insert the values: 21, 4, 7.
 - Into another one, insert the values: 10, 8, 5, 2, 15, 12, 20, 27, 18.
 - Merge the two heaps above.
 - From the result of the merge, perform an `extractMin`.

[13 points]

6. Everyone loves the recurrence relation. Solve
- $T(n) = 5T(n/4) + n$
 - $T(n) = 16T(n/4) + n^2$
 - $T(n) = 15T(n/4) + n^2$
 - $T(n) = 5T(4n/5) + n$

[8 points]

7. Regarding red-black trees
- Into an initially empty RB tree insert the values: 10, 8, 5, 2, 15, 12, 20, 27, 18.
 - From the RB tree below (dotted lines mean red), delete 1.

[14 points]

Total: 75 points

