**Hwk #3 – Fund Algs, Su03 , Dan Barrish-Flood**

Due in class Thurs july 17, 2003, hard copy only.

Please use word processor, or *print neatly*. Of course, if you wish, you can word process all portions except those which require drawing of some sort, which you will please do neatly.

Steven will not accept your homework once they have been returned and the soln's posted. Usually, what this comes down to is, get them in by the due date in lecture. Recall the only way you can submit is by bringing hard copy to class, or handing hard copy to Steven *personally* at some other time (NOT in his mailbox, or under his door).

PLEASE STAPLE your hwk, thanks!

### #1

(ex. 8.1-4) [10 pts] You are given a sequence $S$ of $n$ elements to sort. The input sequence consists of $n/k$ subsequences, each containing $k$ elements. The elements in a given sub-sequence are all smaller than the elements in the next subsequence and greater than the elements in the previous subsequence. Thus all that is needed to sort the whole sequence $S$ of length $n$ is to sort the $k$ elements in each of the $n/k$ subsequences. Show an $\Omega(n \log k)$ lower bound on the number of comparisons (think decision tree) needed to solve this variant of the sorting problem. For this question, *you may not simply combine the lower bounds for the individual subsequences!*

Here's how to get started (and almost finished :) Consider the decision tree of height $h$ for any comparison sort for $S$. Since the elements of each subsequence can be in any order, any of the $k!$ permutations correspond to the final sorted order of a subsequence, and, since there are $n/k$ such subsequences, each of which can be in any order, there are $(k!)^{n/k}$ permutations of $S$ that could correspond to the sorting of some input order. So any decision tree for sorting $S$ must have at least $(k!)^{n/k}$ leaves. Take it from here... you will get to a point when you are faced with $\log(k!)$, and will need to re-write that without the "!". You can either use the fact that $\log(k!) = \Theta(k \log k)$ (recall Stirling), or make use of the observation that $k!$ has its $k/2$ largest terms being at least $k/2$ each.

### #2

[5 pts]

Consider the first 2-3 tree below: Harry the Ent says "When you insert a new item into a 2-3 tree, if that new item becomes a third item in a leaf (i.e. the leaf previously held 2 items), there is no need to update the guides that reside in the internal nodes, they will all remain perfectly valid." You respond to Harry, "Codswallop! I just inserted 9 into that tree (see the second picture), it became the third item in a leaf, and now the guides are all out of whack. E.g., the 8 at the root is no longer the max value in its left-most subtree". It turns out that Harry's claim is right (not that you need to prove it here), but *why is your argument flawed?*
#3. [5 pts] Characterize, in a English sentences (and you may also draw pictures of trees if you wish), those 2-3 trees which will *increase in height* if a record is inserted. You may assume no duplicates.

#4. [5 pts] Consider the 2-3 tree shown below. It appears that the two circled nodes (an interior node and a leaf) contain roughly the same amount of info. (OK, the leaf doesn't have any children, of course.) Why is it likely that the leaf actually contains a good deal more info than the interior node?
#5. [5 pts] Insert the following integers into a binary search tree in this order, and draw the resulting tree: 7, 2, 9, 0, 5, 6, 8, 1.

#6. [5 pts] Suppose you have a tree of n nodes satisfying the max-heap property. Can the keys of this max-heap be printed in sorted order in O(n) time? Why or why not? Hint: recall the exact difference between a binary search tree and a max-heap.

#7. [5 pts] If a node x in a binary search tree has two children, then x's successor has no left child. Explain why this is true, in a English sentences, and pictures if you wish.

#8. [10 pts] (Exercise 12.3-3, CLRS, p. 264). We can sort a given set of n numbers by first building a binary search tree (using TREE_INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-and best-case running times for this sorting algorithm (expressed in Θ notation)? Explain, at least briefly, how you came up with your answers.

#9. [10 pts] We know that any exponential grows faster than any polynomial. By way of example, c^n grows faster than n^k (c and k are const, c>1). Are there any functions "in between", that is, functions that grow faster than any polynomial yet slower than any exponential? Sure there are. Find some function f(n) that satisfies this "in between" property.

Note: k and c must not appear in the expression for f(n).

Hint: convert n^k to 2^{h(n)}, and c^n to 2^{g(n)} where h(n) and g(n) are some functions of n.

#10. [10 pts] Put the following ten functions in increasing order, for sufficiently large n. This is like comparing pairs of functions, except that you're comparing ten, not two. Some functions may be indentical to others, which you will indicate in your answer. Hint: when comparing functions, sometimes it is helpful to take logs (as you know); also it is often helpful to rewrite functions in the form 2^{f(n)}, which might not be as obvious. Recall that n = 2^{ln n} (of course we're talking "log base 2 of n"):

n^{ln n}, 2, (ln n)^n, n^{1/lg n}, 10001n^{10010001}, 1.0000001^n, (1/2)^n, 10010n+1.0000000001^n, (ln n)^{lg n}, n^{lg ln n}

#11. [10 pts] Write down the recurrence equation for the running times of the following functions, but do not solve them:

(A)

function V(n: int): int;
    if n ≤ 1
        then return (1)
    else return (2 * V(n-1))
end; {V}
\((B)\)

**NOTE:** Returning \(n^2\) just takes constant time, so that just contributes \(+ I\) in your running time; I mention this because I recall that I misspoke in rec, saying that would contribute \(n^2\) to the running time. I.e. the bottom line of function \(X\) would contribute the same to the running time, i.e. ending in \(+I\), were it any of these:

```
else return (3 * X(n-1) + X(n-2) + n^2)
else return (3 * X(n-1) + X(n-2) + 1)
else return (3 * X(n-1) + X(n-2))
```

```
function X(n: int): int;
    if n <= 1
        then return (1)
    elseif n == 2 then return (4)
    else return (3 * X(n-1) + X(n-2) + n^2)
    
```

```
eof
```