CSCI 3650 - Algorithms **Name___ Answers ___**
Exam I **February 18, 2009**

This exam has 8 questions. Please make sure that you have them all.

1. Use back substitution to solve this recurrence exactly: T(0) = 3, and T(n) = T(n – 1) + 7 for $n > 0$.
   Just repeatedly substitute using the recurrence:
   $T(n) = T(n-1) + 7$
   $= T(n-2) + 7 + 7$
   $= T(n-3) + 7 + 7 + 7$
   $= \ldots$
   $= T(n-n) + 7\times n$
   $= 3 + 7n$

2. Use the Master Theorem to solve each of these recurrences. In each case, show all work.
   a. $T(n) = 3T(n/4) + n^2$
   Compare n^($\log_4 3$) with n^2.
   $\log_4 3$ is less than 1, since 3 < 4, so that n^($\log_4 3$) is polynomially smaller than n^2.
   So by the master method, $T(n) = \Theta(n^2)$

   b. $T(n) = 4T(n/3) + 1 + 2 + 3 + \ldots + n$
   First recall that for any $k \neq -1$, $1^k + 2^k + \ldots + n^k = \Theta(n^{k+1})$
   This implies that $1 + 2 + 3 + \ldots + n = \Theta(n^2)$
   Compare n^($\log_3 4$) with n^2.
   $\log_4 3$ is less than 2, since 4 < 9, so that n^($\log_3 4$) is polynomially smaller than n^2.
   So by the master method, $T(n) = \Theta(n^2)$

   c. $T(n) = 200T(n/2) + n!$
   Here, we compare n^($\log_2 200$) with n!. Since $\log_2 200$ is a constant, less than 8 in fact, we are comparing n! to a polynomial, and any polynomial is little-o of n!. That is, for any constant k, $n^k = o(n!)$. Thus $T(n) = \Theta(n!)$.

3. Arrange the following functions in order on the table below. If two functions f and g satisfy $f = \Theta(g)$, then they should be on the same row of the table.

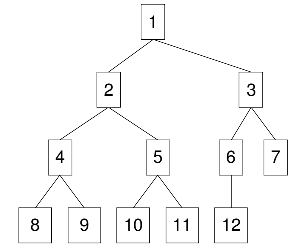
$\log_2(n)$, $\log_3(n)$
$\sqrt{n}$
n, $n + 1$, $5n$
$\log(n!)$
n^2, $n^2 + n$, $4^{\log_2 n}$
n^{20},
5^n
$n!$, h
n^n, g

To the left are shown the functions, with functions that are theta of each other on the same row, and all functions on one row are little-o of functions on the rows below.

Here are some general rules that help to assign these orders:
- Changing the base of a logarithm changes its value by a constant factor.
- By Stirling's formula, $\log(n!) = \Theta(n \log n)$
- $a^{\log_b(x)} = x^{\log_b(a)}$. This helps with the n^2 row
- If $g = o(f)$, then $f + g = \Theta(f)$
- $n! = \Omega(n^k)$ for any constant k

4. Prove (by finding an n_0 and c as appropriate) that $20n^2$ is $O(n^3)$.
Note that $n^3 = n \times n^2$, which will be $\geq 20 \times n^2$ when $n \geq 20$. So we can let $c = 1$ and $n_0 = 20$.

5. To the right, draw a min heap with 12 items, numbered 1-12.

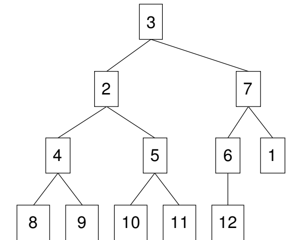


a. How many leaves does this heap have? *6*
b. What is the height of this heap? *3*
c. Draw the array that corresponds to this heap, assuming it is indexed from 1.

1	2	3	4	5	6	7	8	9	10	11	12
	Parent		*A[4]*				*left*	*right*			

d. On the array, indicate the element A[4], its parent and its children.

6. Copy the heap from the previous problem.



a. Show the result of performing Max_Heapify(A, 1) on this heap. (Note that although the heap I asked you to build was a min heap, we are now calling **<u>Max</u>**_Heapify.)

7. Suppose a heap contains n elements.

a. What is the worst-case running time of a call to Max_Heapify(A, i), where $1 \leq i \leq n$? *Explain your answer.*
Max_Heapify swaps an element with its largest child, if that child is larger than itself, and then recursively calls Max_Heapify on that child. Thus the worst case running time would be some constant times the depth of the recursion. This is equal to the height of the element i, which in the worst case is height log n. Thus O(log n) is the worst case running time.

b. What is the worst-case running time of a call to Build_Max_Heap(A) ? *Explain your answer, though you do not need to re-derive the expression.*
Build_Max_Heap calls Max_Heapify on all elements of the array which are not leaves. The running time of each call is the height of the element on which it's called, and the sum of the heights of all the elements of a heap is O(n). We proved this in class, and it is also the extra credit problem on this exam.

c. Write pseudocode for Heap_Sort(A), where A has n elements.

```
Heap_Sort(A)
    Build_Max_Heap(A)                O(n) time
    for i from n downto 2 do         loop n times
        swap A[1] with A[i]          O(1) time
        heaplength = heaplength - 1  O(1) time
        Max_Heapify(A, 1)            O(log n) time
```

d. Explain how the running times of the various parts of your pseudocode above yields an overall running time of O(n log n) for Heap_Sort. *The running times are shown to the right of each line of code.*
The reason that we can't do better than O(log n) time for the calls to Max_Heapify is that they are all called from the root of the heap (A[1]), and the average distance from the root to elements in the heap is O(log n). Thus, the sum of all running times will be O(n log n).

8. Suppose that $N = 2^n - 1$. Find a formula for the sum of the heights of the elements of a heap with N elements. *There are 2^{n-1} elements of height 0, 2^{n-2} elements of height 1, 2^{n-3} elements of height 2, etc..., through 1 element of height $n-1$. Let S be the sum of these heights. Then:*

$$S = 1 \times 2^{n-2} + 2 \times 2^{n-3} + 3 \times 2^{n-4} + \ldots + (n-1) \times 2^0$$

Usual trick: multiply by 2 $\quad 2S = 1 \times 2^{n-1} + 2 \times 2^{n-2} + 3 \times 2^{n-3} + \ldots + (n-1) \times 2^1$

Subtract... $\quad 2S - S = 1 \times 2^{n-1} + 1 \times 2^{n-2} + 1 \times 2^{n-3} + \ldots + 1 \times 2^1 - (n-1) \times 2^0 = 2^n - n - 1$