CS 332 Exam 1 - Review Problems

Fall, 2002

Problem 1: Do the following problems from the course textbook.

- (a) Exercise 1-1, page 25.
- (b) Exercise 1-2, page 25.
- (c) Exercise 1-4, page 25.

Problem 2: Prove or disprove.

- (a) $2^{(2+n)} \in O(2^n)$
- (b) $2^{(2n)} \in O(2^n)$

Problem 3: Show that $\ln n! \in O(n \ln n)$.

Problem 4: Which is a better worst case time, $O(n)$ or $O((\ln n)^2)$? Give a brief justification.

Problem 5: The natural numbers N are the non-negative integers, $N = \{ 0, 1, 2, 3, 4, ... \}$. Let a *natural number function* be a function from N to N { i.e., that takes a natural number as an input and returns a natural number as an output). Find two natural number functions $f(n)$ and $g(n)$ such that $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O(f(n))$.

Problem 6: Do Exercise 3-11, pages 79-80, from the course textbook.

Problem 7: Do Exercises 8.1-1 and 8.2-2, page 155, from the book "Introduction to Algorithms, 1st edition" by Cormen, Leiserson, and Rivest (these two problems are about the Partition() function used in the QuickSort algorithm).

Problem 8: The following two problems are based on Exercises 17.3-1 and 17.3-2 from the book "Introduction to Algorithms, 1st edition" by Cormen, Leiserson, and Rivest.

- (a) Prove that a binary tree that is not full cannot correspond to an optimal prefix code, that is, show that you can create another prefix code that encodes some of the letters in the alphabet using fewer bits.
- (b) What is the Huffman code for the following set of frequencies, based on the first eight Fibonacci numbers?

  a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Problem 9: Below are six C functions that each compute the value of 2^n for positive integers *n*. Analyze each function for how many addition and multiplication operation are needed as a function of the input *n* . Which of the implementations of 2^n is the most efficient?

```
int two1 (int n)
{
   if (n == 0) {return 1;} else return 2*two1(n-1);}
}

int two2 (int n)
{
   if (n == 0) {return 1;} else {return two2(n-1)+two2(n-1);}
}

int two3 (int n)
{
   // Note call to two1.
   if (n == 0) {return 1;} else {return two3(n-1)+two1(n-1);}
}

int two4 (int n)
{
   if (n == 0) {return 1;}
   else if (n % 2 == 0) {int x=two4(n/2); return x*x;}
   else {return 2*two4(n-1);}
}

int two5 (int n)
{
   if (n == 0) {return 1;}
   else if (n%2 == 0) {two5(n/2) * two5(n/2)}
   else {return 2*two5(n-1);}
}

int two6 (int n)
{
   // Note call to two4.
   if (n == 0) {return 1;} else {return two6(n-1)+two4(n-1);}
}
```