MATH 2112 / CSCI 2112
Assignment # 9
Due Wednesday, November 22, 2006

Note: You may use any of the O-, Ω-, and Θ- results derived or stated in class.
Section 9.2: # 41, 45

Section 9.4: # 31, 40
(Hint: For problem 31, the fact $\log x = O(\sqrt{x})$ might be useful for the Ω part)

1. Show (using the definition of O-) that if $f(x) = O(h(x))$ and $g(x) = O(h(x))$, then $f(x) + g(x) = O(h(x))$.
   (Hint: Use the triangle inequality)

2. Find functions f, g, and h such that $f(x) = \Omega(h(x))$ and $g(x) = \Omega(h(x))$, but $f(x) + g(x) \neq \Omega(h(x))$.

3. Factoring by trial division.
   The Prime Number Theorem states that $\pi(x)$, the number of primes up to x, has order $\Theta(x/\ln x)$. Let N be a positive integer with n binary digits (so $n = \lfloor \lg N \rfloor + 1$). The trial division factoring algorithm finds a divisor of N by dividing N by every prime up to $\sqrt{N}$. If any prime divides N evenly, the algorithm returns the factor. Otherwise, the algorithm returns "prime". Find a Θ-estimate for the worst case complexity (worst case number of division operations required) of the algorithm in terms of n.

4. Primality Testing.
   Recall Fermat's Little Theorem, which states that if p is prime, then for any a, $2 \leq a \leq p-1$, we have $a^{p-1} \equiv 1 \pmod{p}$. The contrapositive implies that for all a with $2 \leq a \leq p-1$, if $a^{p-1} \not\equiv 1 \pmod{p}$, then p is composite. Therefore, it is possible to test whether a number N is composite by evaluating $2^{N-1} \pmod{N}$. If this is not equal to 1, then N must be composite (and has failed the test with base 2). A number N that passes the test for some base is called a *pseudo-prime*. A composite number N that passes the test for every base is called a *Carmichael Number*.

   If we use the double-and-add method to compute powers, find an O-estimate for the number of multiplications required to test an odd positive integer N, which has n binary digits.

5. Bubble Sort.
   The Bubble Sort algorithm takes a list of n sortable elements, $\{a[1], a[2], \ldots, a[n]\}$, and compares consecutive elements, swapping them if necessary. The algorithm terminates when there is a pass where no elements are swapped. Here's the pseudo-code:

```
Input: Positive integer n, Sortable List {a[1],a[2],...,a[n]}.

Algorithm:
run := 1
while(run != 0)
```

```
    count := 0
    for i from 1 to n-1
        if a[i] <= a[i+1] then
        else
            count := count + 1  // counts how many switches made in this pass
            temp := a[i]
            a[i] := a[i+1]
            a[i+1] := temp      // swaps a[i] and a[i+1]
        end if
    next i

    if count == 0 then
       run := 0
    else
    end if
end while

Output: Sorted List {a[1],a[2],...,a[n]}
```

(a) How many comparisons are made in each `for` loop?

(b) Let n be a positive integer. How many comparisons must be made to sort the list $\{2, 3, 4, \ldots, n, 1\}$? (Don't forget to prove your answer)

(c) Explain why your answer in part b above gives a Ω-estimate for the worst-case complexity of Bubble Sort.

6. Quick Sort.
The Quick Sort algorithm is a divide-and-conquer algorithm similar to Merge Sort. Given a list of n sortable elements, we pick an element (possibly at random) to be the "pivot", and create three sublists: *pivot-less*, *pivot-equal*, and *pivot-greater*. Then we compare each element to the pivot and place all those less than the pivot in the pivot-less list, those equal to the pivot in the pivot-equal list, and those greater than the pivot in the pivot-greater list. We then repeat the algorithm in each of the pivot-less and pivot-greater sublists, until we end up with lists of zero or one element. The lists are then concatenated.

(a) How many comparisons are needed to create the three sublists?

(b) Assume that at each stage, the pivot is unique (so the pivot-equal list has only one element) and the pivot-less and pivot-greater lists have equal size. Let c_n be the number of comparisons needed to sort a list of n elements in this case. Find a recurrence relation for c_n.

(c) Use the recurrence relation in part b above to show that $c_n = O(n \lg n)$.

(d) Assume that you are very unlucky and at each stage the pivot is unique and the pivot-less list is empty. Let d_n be the number of comparisons needed to sort a list of n elements in this case. Find a recurrence relation for d_n.

(e) Find an explicit formula for d_n (Use $d_1 = 0$).