COMP 271 Design and Analysis of Algorithms
2003 Spring Semester
Question Bank 1

There is some overlap between these question banks and the tutorials. Solving these questions (and those in the tutorials) will give you good practice for the midterm. Some of these questions (or similar ones) will definitely appear on your exam! Note that you do not have to submit answers to these questions for grading. Your TAs will discuss answers to selected questions in the tutorials.

Problem 1. Prove by induction. For all $n \geq 1$:

$$\sum_{i=1}^{n} i(i-1) = \frac{n(n-1)(n+1)}{3}.$$

Problem 2. Suppose $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$. Which of the following are true? Justify your answers.

(a) $T_1(n) + T_2(n) = O(f(n))$

(b) $\frac{T_1(n)}{T_2(n)} = O(1)$

(c) $T_1(n) = O(T_2(n))$

Problem 3. For each pair of expressions (A, B) below, indicate whether A is O, Ω, or Θ of B. Note that zero, one, or more of these relations may hold for a given pair; list all correct ones. Justify your answers.

(a) $A = n^3 + n \log n$; $B = n^3 + n^2 \log n$.

(b) $A = \log \sqrt{n}$; $B = \sqrt{\log n}$.

(c) $A = n \log_3 n$; $B = n \log_4 n$.

(d) $A = 2^n$; $B = 2^{n/2}$.

(e) $A = \log(2^n)$; $B = \log(3^n)$.

Problem 4. Give asymptotic upper bounds for $T(n)$. Make your bounds as tight as possible. You may assume that n is a power of 2.

(a)

$$\begin{aligned} T(1) &= T(2) = 1 \\ T(n) &= T(n-2)+1 \qquad \text{if } n > 2 \end{aligned}$$

(b)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2)+1 \qquad \text{if } n > 1 \end{aligned}$$

(c)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2)+n \qquad \text{if } n > 1 \end{aligned}$$

(d)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2 \cdot T(n/2)+1 \qquad \text{if } n > 1 \end{aligned}$$

(e)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2 \cdot T(n/2)+n \qquad \text{if } n > 1 \end{aligned}$$

(f)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T\left(\frac{n}{2}\right)+n^2 \qquad \text{if } n > 1 \end{aligned}$$

Problem 5. Consider the *mergesort* algorithm for sorting a set of n points.

(a) Draw the recursion tree for this algorithm for $n = 13$.

(b) How many levels are there in the recursion tree?

(c) How many comparisons are done at each of the levels in the worst case?

(d) What is the total number of comparisons needed?

(e) Generalize your results for parts (b–d) for arbitrary n (you may assume that n is a power of 2). Give your answers using the $O()$ notation.

Problem 6. Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

Problem 7.

```
float unknown(int n)
{
 if (n <= 1)
      return(1.0);
 else
      return(unknown(n-1) + unknown(n-2));
}
```

(a) What does the above function compute?

(b) Executing the function for $n = 6$ results in the function being recursively invoked with the argument $n = 1, 2, 3, 4$, and 5. Draw a recurrence tree to illustrate this fact. How many times is `unknown(i)` executed for $1 <= i <= 5$?

(c) How many additions are performed to compute `unknown(6)`?

(d) Assuming that each addition takes constant time, write a recurrence relation for the running time of `unknown(n)`.

(e) Show that the time to compute `unknown(n)` is at least as bad as $\Omega(1.5^n)$. (Hint: Use induction.)

(f) Using the result of part (e), give a lower bound on the time it would take to compute `unknown(100)` on a computer that can do 1 million additions per second? (For this question, just consider additions in determining the running time; thus you may ignore costs such as the cost of testing whether $n \leq 1$ in the *if* statement.)

(g) Can you suggest a more efficient way of computing the same function? How long does it take your program to compute `unknown(n)`? Is your algorithm faster than the above recursive program? If yes, what design idea did you exploit to achieve this speedup?