CS 173, Fall 2009
Midterm 2, Solutions

Problem 1: Short answer (12 points)

The following require only short answers. Justification/work is not required, but may increase partial credit if your short answer is wrong.

(a) Prof. Grindelow claims that insertion sort has the same big-O running time as bubble sort. Is this correct?

Solution: Yes.

(b) What is the big-O running time of Karatsuba's fast multiplication algorithm? [Corrected at exam to say it must be a tight bound, aka big-Θ.]

Solution: $O(n^{\log_2 3})$. It's ok to use Θ in place of O, or to approximate $\log_2 3$ as 1.6 (or even 1.5).

(c) Exactly one of the following two statements is correct. Which one is it?

(1) For every integer p, there is an integer q such that $p \leq q$.

(2) There is an integer q such that for every integer p, $p \leq q$.

Solution: Statement 1.

(d) Is it true that $n!$ is $O(2^n)$?

Solution: No. $n!$ grows faster.

(e) If a binary tree has height h, what is the maximum number of leaves it could have?

Solution: 2^h

(f) Is the root node of a tree always an internal node?

Solution: No. The root is normally an internal node, but it isn't internal when the tree consists of a root with no children.

Problem 2: Short proofs (10 points)

(a) Let $f : \mathbb{Z}^2 \to \mathbb{Z}$ be defined by $f(x, y) = xy + yx^2 - x^2$. Prove that f is onto. Do this directly from the definition of onto.

Solution: (5 points) Suppose that q is any element of $\mathbb{Z}$. Then $f(q, 1) = q \cdot 1 + 1 \cdot q^2 - q^2 = q$. So every element q in the codomain has a preimage $(q, 1)$. So f must be onto.

Notice that it doesn't work so well to set the first coordinate to 1, because then you end up with $f(1, y) = 2y - 1$. That isn't onto when the input y is required to be an integer.

(b) Prove that $3n + 20$ is $O(n^2)$. Prove this directly from the definition of what it means for a function f to be $O(g)$ (where g is another function), being careful to justify your algebraic steps and put them into logical order.

Solution: (5 points) Suppose that $k = 5$ and $C = 4$. Suppose that $n \geq k$. Then $n^2 \geq 25$ so $n^2 \geq 20$. Moreover, $n \geq 1$, so $n^2 \geq n$, and thus $3n^2 \geq 3n$. Combining these two equations, we find that $3n + 20 \leq 3n^2 + n^2 = 4n^2 = Cn^2$.

We've now show that for this choice of k and C, $3n+20 \leq Cn^2$ for any $n \geq k$. Therefore $3n + 20$ is $O(n^2)$.

Problem 3: Induction (10 points)

Let function $f : \mathbb{N} \to \mathbb{Z}$ be defined by

$f(0) = 2$

$f(1) = 7$

$f(n) = f(n-1) + 2f(n-2)$, for $n \geq 2$

Use strong induction on n to prove that $f(n) = 3 \cdot 2^n + (-1)^{n+1}$ for any natural number n.

Base case(s):

Solution: $n = 0$: $f(n) = 2$ and $3 \cdot 2^n + (-1)^{n+1} = 3 \cdot 1 + (-1)^1 = 3 - 1 = 2$.

$n = 1$: $f(n) = 2$ and $3 \cdot 2^n + (-1)^{n+1} = 3 \cdot 1 + (-1)^1 = 3 - 1 = 2$.

Some people proved two bases cases, but they were for $n = 2$ and $n = 3$. Those aren't wrong, but you need to prove the claim for the two earlier values as well.

Inductive hypothesis:

Solution: Suppose that $f(n) = 3 \cdot 2^n + (-1)^{n+1}$, for any natural number $n \leq k$.

Rest of the inductive step:

Solution: [We need to show that $f(k+1) = 3 \cdot 2^{k+1} + (-1)^{k+2}$.]

From the definition of f, $f(k+1) = f(k) + 2f(k-1)$. By the induction hypothesis, $f(k) = 3 \cdot 2^k + (-1)^{k+1}$ and $f(k-1) = 3 \cdot 2^{k-1} + (-1)^k$. Substituting these into the first equation, we get

$$\begin{aligned} f(k+1) &= f(k) + 2f(k-1) = 3 \cdot 2^k + (-1)^{k+1} + 2(3 \cdot 2^{k-1} + (-1)^k) \\ &= 3 \cdot 2^k + (-1)^{k+1} + 2 \cdot 3 \cdot 2^{k-1} + 2(-1)^k \\ &= 3 \cdot 2^k + (-1)^{k+1} + 3 \cdot 2^k + 2(-1)^k \\ &= 3 \cdot 2^{k+1} + (-1)^{k+1} + 2(-1)^k \\ &= 3 \cdot 2^{k+1} + (-1)^{k+1} - 2(-1)^{k+1} \\ &= 3 \cdot 2^{k+1} - (-1)^{k+1} \\ &= 3 \cdot 2^{k+1} + (-1)^{k+2} \end{aligned}$$

So $f(k+1) = 3 \cdot 2^{k+1} + (-1)^{k+2}$, which is what we needed to show.

Notice that the inductive step invokes the result for $k-1$ while trying to prove the result for $k+1$. So we need to use a "strong" inductive hypothesis and we need two bases cases.

Problem 4: Algorithms (10 points)

Consider the following pseudocode for a procedure named Util:

```
Util ( a_1,..., a_n: integers)
    if (n = 1) return (a_1)^2
    else
        begin
        m = ⌈n/2⌉
        p = Util(a_1,...,a_m)
        q = Util(a_{m+1},...,a_n)
        return p + q
        end
```

(a) Explain briefly in English what Util computes.

Solution: (3 points) The sum of the squares of the input numbers.

(b) Express $T(n)$, the running time of Util on an input list of length n, as a recurrence relation. Be sure to include an initial condition. But don't worry about using floors/ceilings/etc if you need to do integer division.

Solution: (4 points) If you assumed that it takes constant time to divide the input list in half, which would be true if everything was stored in an array, then the answer would be

$$T(n) = 2T(n/2) + c$$

(It's also ok to use $O(1)$ instead of a constant like c.)

A slightly less good (-1 point) answer would be

$$T(n) = 2T(n/2) + cn$$

This could actually be correct if your implementation involved linked lists or copying into temporary arrays. However, it's not a good reading of the problem, because it's better to implement that pseudocode using a single fixed array, just like most of the examples in lecture.

In either case, you need a base case like

$$T(1) = d$$

(c) Give a big-theta bound on the running time of Util. Briefly explain why your answer is correct or show work (e.g. some steps from unrolling your recurrence).

Solution: (3 points) Util takes $O(n)$ time.

For full credit on this part, your answer should be consistent with your answer to part (b). So if you gave the recurrence $T(n) = 2T(n/2) + cn$ in part (b), then you get full credit if your answer to this part is $O(n \log n)$.

Problem 5: Recurrences (8 points)

For each recurrence, check the box that correctly gives its big-O solution. Justification is not required.

Hint: is it similar to the recurrence of an algorithm you've seen? what happens if you unroll a step or two?

$T(1) = c$
$T(n) = T(n/2) + d$

$\theta(1)$: ☐ $\theta(\log n)$: ☑ $\theta(n)$: ☐ $\theta(n \log n)$: ☐

Solution: Like binary search.

$T(1) = c$
$T(n) = T(n-1) + d$

$\theta(n)$: ☑ $\theta(n \log n)$: ☐ $\theta(n^2)$: ☐ $\theta(2^n)$: ☐

Solution: You can unroll n times before reaching the base case. Each unrolling step produces one more copy of d.

$T(1) = c$
$T(n) = 2T(n-1) + d$

$\theta(n \log n)$: ☐ $\theta(n^2)$: ☐ $\theta(2^n)$: ☑ $\theta(n!)$: ☐

Solution: Same as the Tower of Hanoi solver. The recursion tree has height $O(n)$, thus $O(2^n)$ nodes.

$T(1) = c$
$T(n) = 2T(n/2) + dn$

$\theta(n)$: ☐ $\theta(n \log n)$: ☑ $\theta(n^2)$: ☐ $\theta(2^n)$: ☐

Solution: Same as mergesort. Differs from the previous because the tree height is only $O(\log n)$.