CS 173, Spring 2010
Midterm 2, Solutions

Problem 1: Short answer (11 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. Moriarty has discovered an algorithm to sort n numbers in *decreasing order* in $O(n^{1.6})$ time. Is this the *fastest* such sorting algortithm?

Solution: No. Mergesort runs in $O(n \log n)$ which is faster.

(b) What is the big-theta running time of the algorithm that solves the towers of Hanoi problem?

Solution: $\Theta(2^n)$

(c) If the root node in a full m-ary tree is an internal node, what is the minimum number of leaves in the tree?

Solution: m. The root must have at least one child, so it must have m children (since the tree is full).

(d) (3 points) If a binary tree T has n leaves, what are the minimum and maximum height possible heights for T?

Solution: Minimum height is $\lceil \log_2 n \rceil$. There is no maximum height, because some large piece of the tree might be a vertical sequence of nodes with a single child, which increases the height without adding leaves.

(e) Is the worst-case running time of linear search $\Omega(\log n)$?

Solution: Yes. Its worst-case running time is $\Theta(n)$ which is $\Omega(\log n)$.

Problem 2: Short proof I (7 points)

(a) (4 points) Prove that $\sum_{i=0}^{n} 2^i$ is $O(2^n)$. 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: Let $k = 1$ and $C = 2$. Then, for any $n \geq k$, $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \leq 2^{n+1} = C(2^n)$.

Since $\sum_{i=0}^{n} 2^i \leq C(2^n)$ for any $n \geq k$, $\sum_{i=0}^{n} 2^i$ is $O(2^n)$.

(b) (3 points) Suppose that $g : A \to B$ and $f : B \to C$. Use a concrete counter-example to disprove the claim that if $f \circ g$ is one-to-one, then f is one-to-one.

Solution: Suppose that $A = \{1\}$, $B = \{a, b, c\}$ and $C = \{7, 8\}$. Now, let $g(1) = a$ and $f(a) = f(b) = f(c) = 8$. Then $f \circ g$ is one-to one, but f is not. [A large range of different examples would work here, but it is critical to use a specific concrete example rather than a general argument.]

Problem 3: Short proof II (8 points)

(a) (4 points) Let $g : \mathbb{N} \to \mathbb{N}$ be onto, and let $f : \mathbb{N}^2 \to \mathbb{Z}$ be defined by

$$f(n, m) = (m - 1)g(n)$$

Prove that f is onto.

Solution: Suppose that x is any integer. Let $z = |x|$. Since g is onto, there is a natural number y such that $g(y) = z$. Then if x is non-negative, $f(y, 2) = (2 - 1)g(y) = g(y) = z = x$. So $f(y, 2) = x$. If x is negative, then $f(y, 0) = (0 - 1)g(y) = -g(y) = -z = x$. So $f(y, 0) = x$.

Since this argument works for any choice of x, we've shown that f is onto.

(b) (4 points) Recall the definition of $X \subseteq Y$: for every p, $p \in X \to p \in Y$. Prove the following claim: if $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.

Solution: Suppose that $A \subseteq B$ and $B \subseteq C$. Let x be an element of A. Since $A \subseteq B$ and $x \in A$, $x \in B$ (definition of $\subseteq$). Since $B \subseteq C$ and $x \in B$, $x \in C$ (definition of $\subseteq$). We have shown that if $x \in A$, then $x \in C$, which means that $A \subseteq C$ (definition of $\subseteq$ again).

Problem 4: Induction I (9 points)

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

$f(0) = 2$

$f(1) = 3$

$\forall n \geq 1,\ f(n+1) = 3f(n) - 2f(n-1)$

Use strong induction on n to prove that $\forall n \geq 0,\ f(n) = 2^n + 1$.

Solution:

Base case(s):

$f(0) = 2 = 2^0 + 1$

$f(1) = 3 = 2^1 + 1$

Inductive hypothesis:

Suppose that $f(n) = 2^n + 1$ for $n = 0, 1, \ldots, k$.

Rest of the inductive step:

$f(k+1) = 3f(k) - 2f(k-1)$.

By the inductive hypothesis, we know that $f(k) = 2^k + 1$ and $f(k-1) = 2^{k-1} + 1$. By substituing these values, we get $f(k+1) = 3(2^k + 1) - 2(2^{k-1} + 1)$.

So $f(k+1) = 3(2^k+1) - 2(2^{k-1}+1) = 3 \cdot 2^k + 3 - 2 \cdot 2^{k-1} - 2 = 3 \cdot 2^k - 2^k + 1 = 2 \cdot 2^k + 1 = 2^{k+1} + 1$

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

Problem 5: Induction II (9 points)

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

$f(1) = 0$

$\forall n \geq 2,\ f(n) = 1 + f(\lfloor n/2 \rfloor)$

Use strong induction on n to prove that $\forall n \geq 1,\ f(n) \leq \log_2 n$. You cannot assume that n is a power of 2. However, you can assume that the log function is increasing: if $x \leq y$ then $\log x \leq \log y$.

Solution: Base case(s): $f(1) = 0$ and $\log_2 1 = 0$ So $f(1) \leq \log_2 1$.

Inductive hypothesis: Suppose that $f(n) \leq log_2 n$ for $n = 1, \ldots, k-1$.

Rest of the inductive step: We can assume that $k \geq 2$ (since we did $n = 1$ for the base case). So $\lfloor k/2 \rfloor$ must be at least 1 and less than k. Therefore, by the inductive hypothesis, $f(\lfloor k/2 \rfloor) \leq \log_2(\lfloor k/2 \rfloor)$.

We know that $f(k) = 1 + f(\lfloor k/2 \rfloor)$, by the definition of f. Substituting the result of the previous paragraph, we get that $f(k) \leq 1 + \log_2(\lfloor k/2 \rfloor)$.

$\lfloor k/2 \rfloor \leq k/2$. So $\log_2(\lfloor k/2 \rfloor) \leq \log_2(k/2) = (\log_2 k) + (\log_2 1/2) = (\log_2 k) - 1$.

Since $f(k) \leq 1 + \log_2(\lfloor k/2 \rfloor)$ and $\log_2(\lfloor k/2 \rfloor) \leq (\log_2 k) - 1$, $f(k) \leq 1 + (\log_2 k) - 1 = (\log_2 k)$. This is what we needed to show.

Problem 6: Algorithms (8 points)

Consider two lists $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ containing n items each. The following algorithm returns *True* if the two arrays contain at least one common element, and returns *False* otherwise. LinearSearch$(x_1, \ldots, x_n, b)$ returns the position of b in the list $x_1, \ldots, x_n$ or 0 if b is not in the list.

```
01 HasCommon (a_1, ..., a_n and b_1, ..., b_n)
02     for i := 1 to n
03         begin
04         if (LinearSearch(a_1,...,a_n, b_i) ≠ 0)
05             return True
06         end
07     return False
```

(a) Explain briefly in English how HasCommon works.

Solution: HasCommon considers each element of the b array in turn and uses linear search to see if it's in the a array. If this process turns up an element that's in both arrays, the search terminates immediately, returning True. If the search terminates without finding a common element, HasCommon return False.

(b) Give a big-theta bound on the running time of HasCommon. For full credit, you must show work or briefly explain your answer.

Solution: The call to LinearSearch at line 04 takes $\Theta(n)$ time in the worst case. This call is inside a loop that runs n times. Everything else in the function takes constant time. So the total running time is $n \cdot \Theta(n)$ which is $\Theta(n^2)$.

(c) Suppose we assume that both input lists are sorted, and replace LinearSearch with BinarySearch. What is the big-theta running time of this new version?

Solution: Since BinarySearch runs in $\Theta(\log n)$ time, worst case, the new running time will be $\Theta(n \log n)$

Problem 7: Recurrences (8 points)

For each recurrence, check the box that correctly characterizes its solution. Justification is not required.

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

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

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

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

$T(1) = c$
$T(n) = \frac{1}{2}T(n-1) + d$

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

$T(1) = c$
$T(n) = 3T(n/2) + dn$

$O(n)$: ☐ $\theta(n \log n)$: ☐
$\theta(n^{\log_2 3})$: ☑ $\theta(n^2)$: ☐

Hint: what algorithm might this be? (It was Karatsuba's algorithm, by the way.)

Some problems on the conflict exam were the same, or quite similar to, problems on the main exam. Here are solutions to ones that are very different.

Conflict Problem 1: Short answer (11 points)

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

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

Solution: The maximum number of leaves is 2^h.

(b) (3 points) Define what it means for a tree T to be a full complete m-ary tree.

Solution: T is a full complete m-ary tree if each node either has no children or has m children, and all the leaves are at the same height.

Conflict Problem 7: Recurrences (8 points)

$T(1) = c$
$T(n) = nT(n-1)$

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

Conflict Problem 2: Short proof I (7 points)

(a) (4 points) Prove that $\sum_{k=0}^{n} k$ 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: Let $K = 1$ and $C = 1$. Then, for any $n \geq K$, $n \leq n^2$.

Then $\sum_{k=0}^{n} k = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \leq \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2 = Cn^2$.

Since $\sum_{k=0}^{n} k \leq C(n^2)$ for any $n \geq K$, $\sum_{k=0}^{n} k$ is $O(n^2)$.

(b) (3 points) Suppose that $g : A \to B$ and $f : B \to C$. Use a concrete counter-example to disprove the claim that if $f \circ g$ is onto, then g is onto.

Solution: Suppose that $A = \{1\}$, $B = \{a, b, c\}$ and $C = \{7, 8\}$. Now, let $g(1) = a$ and $f(a) = f(b) = 7$ and $= f(c) = 8$. Then $f \circ g$ is onto, but g is not. [A large range of different examples would work here, but it is critical to use a specific concrete example rather than a general argument.]

Conflict Problem 3: Short proof II (8 points)

(a) (4 points) Let $g : \mathbb{N} \to \mathbb{N}$ be one-to-one, and define $f : \mathbb{N} \to \mathbb{Z}$ as

$$f(n) = \begin{cases} g(n) & \text{if } n \text{ is even} \\ -g(n) & \text{if } n \text{ is odd} \end{cases}$$

Prove that f is one-to-one.

Solution: First, notice that the definition of f implies that $|f(n)| = g(n)$ for any natural number n.

Let a and b be natural numbers and suppose that $f(a) = f(b)$. Then $|f(a)| = |f(b)|$. So $g(a) = g(b)$. Since g is one-to-one, this implies that $a = b$, which is what we needed to show.

(b) (4 points) Recall the definition of $X \subseteq Y$: for every p, $p \in X \to p \in Y$. Prove the following claim: for any sets A, B, C, if $A \subseteq B$, then $A \times C \subseteq B \times C$.

Solution: Suppose that $A \subseteq B$ and let (x, y) be any element of $A \times C$. Then $x \in A$ and $y \in C$, by the definition of Cartesian product. Since $x \in A$ and $A \subseteq B$, $x \in B$. So $x \in B$ and $y \in C$. So $(x, y) \in B \times C$ by the definition of Cartesian product.

We've shown that every element of $A \times C$ is in $B \times C$, so $A \times C \subseteq B \times C$.

Conflict Problem 6: Algorithms (8 points)

Consider a list $a_1, a_2, \ldots, a_n$ of n items. The following algorithm returns *True* if all the elements in its input list are distinct, and returns *False* if some item appears more than once. LinearSearch$(x_1, \ldots, x_n; b)$ returns the *first* position of b in the list $x_1, \ldots, x_n$, or 0 if b is not in the list.

```
01 AllDistinct(a_1, ..., a_n)
02     if (n = 1)
03         return True
04     for i := n down to 2
05         begin
06         if (LinearSearch(a_1, ..., a_{i-1}; a_i) ≠ 0)
07             return False
08         end
09     return True
```

(a) Explain briefly in English how AllDistinct works.

Solution: AllDistinct considers every element of the list in turn and uses LinearSearch to look for a duplicate among the earlier elements in the list. It returns False if this process finds a duplicate and True otherwise.

(b) Give a big-O bound on the running time of AllDistinct. For full credit, you must show work or briefly explain your answer.

Solution: The call to LinearSearch on element i takes $O(i)$ time. So the work for LinearSearch across all the iterations of the loop is $O(\sum_{i=2} ni) = O(\frac{n(n+1)}{2} - 1) = O(n^2)$. The other parts of the function take constant time, or constant time for each iteration of the loop (i.e. therefore $O(n)$ time). So the whole function runs in $O(n^2)$ time.

(c) Suppose we assume that the input list is sorted, and we replace the function call LinearSearch$(a_1, \ldots, a_{i-1}; a_i)$ with LinearSearch$(a_{i-1}; a_i)$. What is the big-theta running time of this new version?

Solution: For this verion, each call to LinearSearch takes only constant time. So the whole inside of the loop takes constant time. So the whole function takes only $O(n)$ time.