Math 55 Fall 2012
HW 4 Solutions

2.5 #20. Given $|A|=|B|$ and $|B|=|C|$, let $f: A \to B$ and $g: B \to C$ be bijective. I claim that $g \circ f: A \to C$ is bijective. One way to see this is using 2.3 Exercise 33: both ~~fun~~ f and g are injective, so $g \circ f$ is injective, and both f and g are ~~surj~~ surjective, so $g \circ f$ is surjective. Or, you can verify that $f^{-1} \circ g^{-1}$ is a 2-sided inverse to $g \circ f$:

$$f^{-1} \circ g^{-1} \circ g \circ f = f^{-1} \circ id_B \circ f = f^{-1} \circ f = id_A \; ;$$
$$g \circ f \circ f^{-1} \circ g^{-1} = g \circ id_B \circ g^{-1} = g^{-1} \circ g = id_C \, .$$

2.5 #34. We can give ~~a~~ bijective functions from $\mathbb{R}$ to $(0,1)$ and ~~[illegible]~~ vice versa. The function $\tan x : (-\pi/2, \pi/2) \to \mathbb{R}$ is bijective with inverse $\tan^{-1} x$, so $f(x) = \tan(\pi(x - \frac{1}{2})) : (0,1) \to \mathbb{R}$ is bijective with inverse $\frac{1}{2} + \frac{\tan^{-1} x}{\pi}$.

3.1 #18.

```
algorithm lastmin ( a_1, ..., a_n : integers )
[first, determine the minimum]
m := a_1
for i = 2 to n
        if a_i < m then m := a_i
[now find the last a_i which is equal to m]
for i = n to 1 in steps of -1 :
        if m = a_i return i .
```

Remark A more clever 'one-pass' way to do this is ~~[illegible]~~ find the minimum m by scanning from the last element to the first, keeping track of the index i each time we find $a_i < m$. At the end of this scan, i is the location of the last minimum.

3.1 #44 (We did this in the lecture)

Here is a recursive algorithm that does the job:

algorithm findplace ($a_1, \ldots, a_n$: increasing integers ; x : integer)
if $n=0$ then return 0
$k := \lfloor (n+1)/2 \rfloor$
if $x \le a_k$ then return findplace ($a_1, \ldots, a_{k-1}$; x)
else return findplace ($a_{k+1}, \ldots, a_n$; x) + k.

3.1 #48 Here are all the comparisons done by insertion sort, and the rearrangement of the list which it makes after each comparison

	List	Compare
(original input) →	7 4 3 8 1 5 4 2	$7 \overset{?}{<} 4$
	4 7 3 8 1 5 4 2	$4 \overset{?}{<} 3$
	3 4 7 8 1 5 4 2	$3 \overset{?}{<} 8$
		$4 \overset{?}{<} 8$
		$7 \overset{?}{<} 8$
		$3 \overset{?}{<} 1$
	1 3 4 7 8 5 4 2	$1 \overset{?}{<} 5$
		$3 \overset{?}{<} 5$
		$4 \overset{?}{<} 5$
		$7 \overset{?}{<} 5$
	1 3 4 5 7 8 4 2	$1 \overset{?}{<} 4$
		$3 \overset{?}{<} 4$
		$4 \overset{?}{<} 4$
	1 3 4 4 5 7 8 2	$1 \overset{?}{<} 2$
		$3 \overset{?}{<} 2$
done →	1 2 3 4 4 5 7 8	

This required 15 comparisons.

Now the same using binary search to locate each insertion point. (I'll assume when we do binary search on a list of even length n, we compare with the $\lfloor (n+1)/2 \rfloor$-th element.)

start →	7 4 3 8 1 5 4 2	$7 \overset{?}{<} 4$
	4 7 3 8 1 5 4 2	$4 \overset{?}{<} 3$
	3 4 7 8 1 5 4 2	$4 \overset{?}{<} 8$
		$7 \overset{?}{<} 8$
		. . . . .
		$4 \overset{?}{<} 1$
		$3 \overset{?}{<} 1$
	1 3 4 7 8 5 4 2	$4 \overset{?}{<} 5$
		$7 \overset{?}{<} 5$
	1 3 4 5 7 8 4 2	$4 \overset{?}{<} 4$
		$1 \overset{?}{<} 4$
		$3 \overset{?}{<} 4$
	1 3 4 4 5 7 8 2	$4 \overset{?}{<} 2$
		$3 \overset{?}{<} 2$
		$1 \overset{?}{<} 2$
done →	1 2 3 4 4 5 7 8 2	

This required 14 comparisons.

3.1 #64 First let's be clear what the problem is: we ask whether there could be an algorithm X that take as input the data A - a description of an algorithm (i.e., a program)
I - an input for A

and produces output
T if A ever prints 1 in its output when given input I (whether A(I) terminates or not)

F otherwise.

Of course X should terminate on every input pair (A, I).

The answer is that no such algorithm X exists. The reason is that if it did, there would be an algorithm to solve the halting problem, as follows:

Given input (B, I), construct from the code for B a code for an algorithm A which does the same thing as B, except

that when B would generate output, A prints nothing, and if B terminates, then A prints 1. It's clearly possible to design an algorithm which takes input B and outputs such an A. Now combine this with X: given input (B, I), construct (A, I) and compute X(A, I). The output is T if B halts on I, F otherwise, so this would solve the halting problem. Since we know this is impossible, X does not exist.

Additional problem

If N = 13, first weigh coins 1 through 9 against 9 good coins. If the result is unbalanced, we now have the problem of finding a counterfeit coin among 9 coins when the bad coin is known to be heavy or light. We saw in class how to do this with two more weighings.

If the first weighing balances, we have the heavy-or-light problem with N = 4 (for coins 10, 11, 12, 13). Now weigh 10, 11, 12 against 3 good coins. If the result is unbalanced we have the 3-coin problem with bad coin known to be heavy or light, which we can solve in one weighing.

If the second weighing balances, then 13 is the bad coin, and we can weigh it against a good coin to determine whether it is heavy or light.

If N = 14, we have to distinguish between 28 possibilities: 14 (which coin is bad) × 2 (whether it is heavy or light). But 3 weighings can only distinguish $3 \times 3 \times 3 = 27$ cases, at most.