

---

## HEAD-BANGING SESSION 0 SOLUTIONS

### FALL 2008 CS 473: ALGORITHMS

---

**Problem 1. [Tiling Squares]**

The solution is a simple proof by induction. The base case is  $n = 1$ , a  $2 \times 2$  board. Clearly if one square is removed, the remainder can be tiled by one L-shaped tromino.

Assume that any board of size  $2^n \times 2^n$ , with one square removed, can be tiled.

Now, given a board of size  $2^{n+1} \times 2^{n+1}$ , with one square removed, it can be divided into four  $2^n \times 2^n$  smaller boards, one of which contains the removed square. In the remaining three  $2^n \times 2^n$  boards, we'll remove a tile from each by taking the adjacent center corners of the three to give an L-shaped hole.

The inductive hypothesis will show how to cover the four smaller boards, since they each have a square removed, and are size  $2^n \times 2^n$ . The L-shaped hole can also be covered by a tromino. This covers the original board except for the original missing square!

**Problem 2. [Word Count]**

A string  $s$  is called *valid* if  $s$  has no two 1s at distance 2 from each other. Define

$$A_n = \{s \mid s \in \{0, 1\}^n \text{ and } s \text{ is valid}\}$$

and define  $a_n = |A_n|$ .

Consider any string  $s$  in  $A_n$ . If it starts with 0, then the last  $n - 1$  characters of  $s$  is an element in  $A_{n-1}$ . If  $s$  starts with 1, instead, then the third bit has to be 0 in  $s$ . If the second bit in  $s$  is 0, then the last  $n - 3$  bits of  $s$  is an element in  $A_{n-3}$ . If the second bit in  $s$  is 1, then the fourth bit in  $s$  has to be 0 and the last  $n - 4$  bits of  $s$  is an element in  $A_{n-4}$ .

Now, to show the inclusion other way around, observe that any string in  $A_n$  prefixed with 0 is an element in  $A_n$ . Any string in  $A_{n-3}$  prefixed with 100 is an element in  $A_n$  and any string in  $A_{n-4}$  prefixed with 1100 is an element in  $A_n$ .

Thus we have shown that  $A_n$  is partitioned into the sets  $0A_{n-1}$ ,  $100A_{n-3}$  and  $1100A_{n-4}$ , hence we can conclude that  $a_n = a_{n-1} + a_{n-3} + a_{n-4}$  for all  $n \geq 5$ . For the base cases, define  $a_1, a_2, a_3, a_4$  by explicitly computing them.

**Problem 3. [Asymptotically Equivalent]**

We will first prove that  $\log n! = O(\log n^n)$ . It is obvious to see that  $n! \leq n^n$ .

We will next prove that  $n! \geq n^{n/2}$  (thus proving that  $\log n! = \Omega(n \log n)$ ). Observe that  $(n - i + 1)i \geq n$  for all  $1 \leq i \leq n$  and hence the result.

Using the above two results, we can conclude that  $\log n! = \Theta(n \log n)$ .

To prove that  $n! = o(n^n)$ , we will prove that  $n!/n^n = o(1)$ . Observe that

$$n!/n^n \leq 1/n = o(1)$$

**Problem 4. [1-trees]**

A graph  $G$  is a 1-tree if and only if  $m = n$  and  $G$  is connected. Do you see why? From this characterization we can obtain the following algorithm.

If  $m \neq n$ , REJECT.

Check if  $G$  is connected.

If  $G$  is connected, ACCEPT, otherwise, REJECT.

One can check if  $m = n$  in  $O(n)$  time (if  $m$  is not given explicitly) by scanning the edge list of each vertex and stopping as soon as the number of edges exceeds  $n$ . This ensures that the first step does not take more than  $O(n)$  time. Checking whether a graph is connected takes  $O(m)$  time using either BFS or DFS. Since  $m = n$  from the first step (otherwise the algorithm stops and rejects), this step also takes  $O(n)$  time.

An alternative algorithm is to combine the two steps into a slight modification of BFS or DFS.