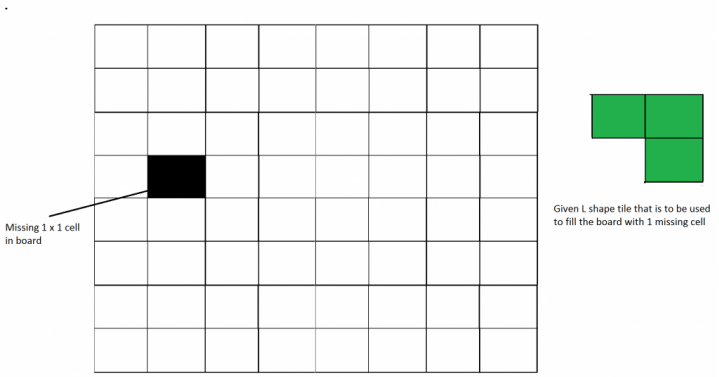
CS 325
Due Friday, 9 Feb

Homework #4
Divide and Conquer

In this assignment, you will solve three simple and amusing problems using the divide and conquer method. In each case, you will be able to prove that your algorithm is correct and deduce how the running time grows with the problem size.

The Tiling Game

The game is played on a board with $2^k \times 2^k$ squares. An adversary removes from play exactly 1 square of the board. To win, you must exactly cover each of the remaining squares with non–overlapping *tiles*.



Problems

1. Write a program which wins the *Tiling Game* using a divide and conquer technique.
2. Demonstrate that your program works by running it for various reasonable board sizes and with different squares removed. Print the final tiled board of a few games each for boardsizes 8×8 and 16×16. For example, the result of a 4×4 board size with square $(2, 2)$ removed, may look like this:

   ```
   1 1 4 4
   1 X 5 4
   2 5 5 3
   2 2 3 3
   ```

3. Prove by induction that your program always wins, for any $2^k \times 2^k$ boardsize and no matter what square the adversary removes.
4. Write and solve a difference equation for the running time of your program.
5. Plot the running times of your program as a function of the board size. Do the run times you measure agree with your run–time analysis from #4?

Find the ODD Coin

1. You have a large number of coins and a pan balance. You may put any number of coins in each pan of the balance. The balance will tell you if the set of coins in in one pan weighs the same as the set of coins in the other pan, or it will tell you which set is heavier.
   Somewhere among the coins is one coin which has different weight than the other coins.
   All the coins, except this one odd coin, have exactly the same weight.
   The problem is to find the odd coin

2. Design a *divide and conquer* algorithm to solve this coin problem.
   You may assume that the number of coins is a power of 3.

3. Give a difference equation for the number of weighings used by your algorithm.
   Solve this difference equation.
   Give a difference equation for the run time of your algorithm.
   Solve this equation to "Big Oh" order.

4. To simulate the balance, you should have a routine which takes as input two subranges of integers which indicate the two subsets of coins you want to compare, and your routine should return one of **heavier**, **lighter**, or **equal** depending on the location of the odd coin and its weight relative to the other coins. Your balance routine will need to know about the position and weight of the odd coin, but it can only communicate this information back to your program by use of the three words **heavier**, **lighter**, and **equal**. You should use a random number generator to decide the position and weight of the odd coin.

5. Now run your program for various numbers of coins, but use several diffferent runs for each number of coins. Plot both the running time and the number of calls to balance as functions of the number of coins.

6. Let n be the number of coins.
   Does your program always take the same time and make the same number of weighings for each problem with n coins?
   How well does your run time and weighing data fit with the predictions from your difference equations?