

MA/CSSE 473 – Design and Analysis of Algorithms

Homework 04 – 90 points Updated for Summer, 2017

Most of these problems are variations on the "Tiling with L-shaped Trominoes" example from Day 4 (in 2017) class. For another brief discussion, see <http://www.cut-the-knot.org/Curriculum/Geometry/Tromino.shtml> I have placed on Moodle an excerpt from the Johnsonbough book on which my in-class discussion was based.

Here is pseudocode for the tiling algorithm from class. Many details are omitted, but it conveys the idea.

```
def tileWithTrominoes(n, s):
    """n is the dimension of the deficient board to be tiled, and is assumed to be a power of 2.
    s gives the location of the missing square that makes the board deficient."""
    if n == 2:
        # place the tromino so it covers the three squares
        return
    # Consider the four squares in the center of the board.
    # Three of them are in quadrants of the board that do
    # not contain s. Place a tromino on those three squares. Now let
    # s1, s2, s3, and s4 be m and the three squares that we just covered with a tromino.

    tileWithTrominoes(n/2, s1) # assume that we have a mechanism
    tileWithTrominoes(n/2, s2) # for translating these tilings
    tileWithTrominoes(n/2, s3) # to the appropriate quadrants
    tileWithTrominoes(n/2, s4) # of the original board.
```

Hint: drawing grids with Trominoes on your computer. Over the years, I have tried different ways of drawing grids that contain trominoes, and Excel is the tool that works best for me. I make each cell have the same height and width, then use background colors to indicate which cells contain trominoes.

1. (5) Write a recurrence relation for the running time of this algorithm, and use the Master Theorem to show that its solution is $\Theta(n^2)$. Thanks to Mike Jones for the following additional Master Theorem reference: http://www.math.dartmouth.edu/archive/m19w03/public_html/Section5-2.pdf

Previous questions and answers from Piazza:

Q: Does dimension n refer to the n in a $2n \times 2n$ checkerboard? **A:** No. n is the actual dimension (assumed to be a power of 2). For example, for a 4×4 board, $n = 4$. Notice the division of n by 2 in the recursive calls. If n were what you think it is, then it would be $n-1$ in the recursive calls.

Q: Shouldn't the base case in the algorithm be $n == 1$? **A:** It could be, but what does it mean to tile an empty board (a 1×1 deficient board is empty)? So $n == 2$ is a more natural base case.

2. (2) Show that no 9×9 deficient rectangular board can be tiled with trominoes.

Previous questions and answers from Piazza:

Q: When the problem states "Show that no 9×9 deficient rectangular board can be tiled with trominoes". Does this mean we need to create a proof? **A:** Yes, you need a proof. But it can be a VERY short proof. There is a reason why this problem is worth so few points!

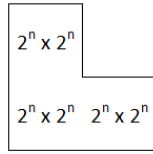
3. (8) Show that a 5×5 board with the upper-left corner square missing can be tiled with trominoes.

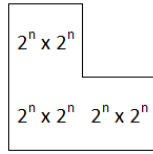
Previous questions and answers from Piazza:

Q: in the case of "Show that a 5×5 board with the upper-left corner square missing can be tiled with trominoes", we can just show an example of the tiling? **A:** Yes. A single example suffices to show that it can be done.

4. (10) Show a deficient 5×5 board that cannot be tiled with trominoes, and prove that this board cannot be tiled.

5. (5) Show that any $2i \times 3j$ board (not deficient) can be tiled with trominoes, for all positive integers i and j .



6. (10) A $2^n \times 2^n$ L-shape is a figure of the form  with no missing squares, or it is a rotation of this figure by a multiple of 90° . Write a recursive algorithm that tiles any $2^n \times 2^n$ L-shape with trominoes. Each recursive call should tile a smaller L-shape. You may express it in English, pseudo code, or code from any programming language that is likely to be known by most people in this course. Feel free to use diagrams as part of your algorithm description.

Previous note posted on Piazza:

In the problem that asks you to show how to tile an L-shape recursively, you are to do just that. Each recursive call should be to tile a smaller L-shape. As we did in class when tiling a deficient square, you can show how the recursive calls work by drawing a picture plus a few words; no detailed algorithm is necessary.

For the next problem, note that a $2^n \times 2^n$ deficient square can be split up into a $2^{n-1} \times 2^{n-1}$ deficient square and an L-shape. Your new algorithm should call the L-shape algorithm from the previous problem to tile the L-shape, and call your new algorithm recursively to tile the smaller deficient square.

7. (10) Use the preceding exercise as the basis for a different recursive algorithm (different from the one at the top of the page) for tiling (with trominoes) any $2^n \times 2^n$ deficient rectangular board with trominoes.
8. (10) Show that any deficient 7×7 board can be tiled with trominoes. There will be several different cases, depending on the location of the missing square. You are allowed to use the results of some of the previous problems.

9. (15) More induction practice. This problem comes from Weiss, Chapter 18.

A (rooted) *binary tree* T is either empty, or it has a distinguished node called the *root* plus two disjoint subtrees T_L and T_R , each of which is a binary tree (commonly known as the *left* and *right subtrees* of T). **Can you see the difference between this definition and the definition of Extended Binary Trees from the day 4 slides?**

- a. (10 points) Use (strong) mathematical induction on the number of nodes in the tree to prove the following properties of rooted binary trees: If all of a binary tree's leaves are L_1, L_2, \dots, L_M and their depths (distance from the tree's root) are d_1, d_2, \dots, d_M , respectively, then

$$\sum_{i=1}^M 2^{-d_i} \leq 1.$$

You may want to draw a few tree shapes and do the arithmetic to convince yourself that this is true before you prove it. You must use the above definition of binary tree as either empty or a root with two binary subtrees as the basis for your induction, or show (also by induction) that whatever basis you use is equivalent. I.e., Your base case should be the empty tree and the induction step should assume that the property is true for smaller trees (in particular, the two subtrees) and then show that it is true for the whole tree).

- b. (5 points) Give a condition on the tree shape that is necessary and sufficient for replacing the "less than or equal to" sign in the statement with an "equals" sign. You do not have to supply a proof for this part.

10. (15) Largest interval overlap. Suppose we have an alphabetical list of N famous people who are no longer alive, along with the birth and death year of each person. For simplicity, assume that
- all of the birth years are AD.
 - If person A died in year y and person B was born the same year, then A died before B was born, so there is no overlap.
 - No famous person died in the same year that he/she was born.
 - The order of magnitude of N is at least 10^4 .

Give an efficient algorithm that determines the maximum number of these famous people who were alive at the same time, and the time interval in which all of them were simultaneously alive. If there are two different time intervals with the same maximum number of living celebrities, then you can return any of those intervals.

A more sophisticated problem might also ask for the names of the people who were alive during the "max" time interval, but I am not asking you to do that, so the names of the people are actually irrelevant to the problem.

You can assume that the input data is given as an array of arrays. Each element of the outer array is an array of two numbers: the birth and death year for one famous person.

For example (I used this for testing), here is data representing 6 people who all lived between 1 and 17 AD.

```
data = [[2, 9], [3, 12], [5, 17], [1, 3], [10, 12], [7, 13]]
```

So the first person was born in year 2 and died in year 9 ... the last person was born in 7 and died in 13.

Answer for this case: Four of these people were alive from year 7 to year 9. Another possible answer for the same data: four were alive from year 10 to year 12.

You should indicate the time and space efficiency of your algorithm, as a function of N and of the current year.

It is not very hard to come up with an algorithm for this problem. It is harder to produce an *efficient* algorithm.