CMPS 102 — Fall 2018 – Homework 1

Alyssa Melton

I have read and agree to the collaboration policy.

Collaborators: none

Solution to Problem 4

Before the algorithm, I will explain L placement which is a bit wordy to put inside it.
Each L takes up 3 square units, in this algorithm I will say "place an L that is shared between three quadrants. By this, I mean the centermost part of the L is in one quadrants, and the other two are in two separate quadrants. Similarly, we can think of this as a 2x2 box being placed such that there is a quarter of it in every quadrant, and one of the corners is taken away.
This is my way of saying to mark the center most corners of the quadrants of the square and connect them with an L. Each of these corners take the place of the given black box for each recursively divided square.

Given n and the coordinate of the black box (i, j)
Square = initial Square of side length 2^n
box = (i, j)

DivideSquare (Square, box)
    Divide Square into four quadrants and identify what quadrant (i, j) is in.
    Place an L so that it is shared between the three quadrants that do not contain (i, j).
    Until Square is full (or size 2x2),
        call DivideSquare recursively on each quadrant,
        passing the corner that was included in the L for that specific quadrant as 'box',
        or if part of the L was not in that quadrant, the original (i, j) as 'box'.

Claim 1. *This algorithm runs in time $O(2^{2n})$.*

Proof. Because we're given a box of side length 2^n, we know that the box's area must be 2^{2n}. Each tile is 3 square units and one box is left black/blank. Thus, we derive the number of L tiles to be $\frac{(2^{2n})-1}{3}$. If each placemet of an L tile is an $O(1)$ operation, we can see that the time complexity of L tile placement will be $O(2^{2n})$

Each call to DivideSquare divides the square into 4 parts, where DivideSquare is called again. Each time the square is divided into quadrants, the side length is cut in half, thus, we get $4 * DivideSquare(\frac{sidelength}{2})$.
The sidelength is given in the form of 2^n.
The recurrence relation for this algorithm is $T(n) = 4T(\frac{2^n}{2}) + O(2^{2n})$
To make this simpler, we can analyze this in terms of k, where $k = 2^n$

This gives us $T(n) = 4T(\frac{k}{2}) + O(k^2)$

To derive the time complexity for this, we can use the master theorem. Let $f(k) = k^2$, $a = 4$, $b = 2$ This recurrence satisfies case 2 of the master theorem, such that $f(k) = \theta(k^{log_b a}) \rightarrow k^2 = \theta(k^2)$, which is true, thus we get $T(n) = \theta(k^2)$ as our time complexity in terms of k.

Substituting 2^n back in for k, we have $T(n) = \theta(2^{2n})$.

□