Although this exercise isn't worth any points, it gives you valuable experience doing algorithm analysis. You're almost definitely going to have to complete the exercises to succeed in the course.

Problem 1 – Wacky Analysis, Take 1

Download the zip file below and extract it somewhere on your computer. The code is an excerpt from the `LevelBuilder` script from the Wacky Breakout game from the previous course in the specialization.

```
1  // add rows of blocks
2  Vector2 currentPosition = new Vector2(leftBlockOffset, topRowOffset);
3  for (int row = 0; row < 3; row++)
4  {
5      for (int column = 0; column < blocksPerRow; column++)
6      {
7          PlaceBlock(currentPosition);
8          currentPosition.x += blockWidth;
9      }
10
11     // move to next row
12     currentPosition.x = leftBlockOffset;
13     currentPosition.y += blockHeight;
14 }
```

Analyze the algorithmic complexity of the code where n is the number of blocks per row. The PlaceBlock operation is O(1).

<u>**Solution**</u>

The best way to approach your analysis is to figure out the complexity of each line of code. I've added line numbers to the code above for easy reference.

Comments, blank lines, and curly braces (Lines 1, 4, 6, 9, 10, 11, and 14) don't actually lead to executable code, so they don't contribute anything to the algorithm complexity.

Assignments statements and calculations (like additions) take constant time, so Lines 2, 8, 12, and 13 are all O(1). The problem statement said Line 7 is also O(1).

That leaves the loops on Lines 3 and 5. The loop on Line 3 executes 3 times (with row equal to 0, 1, and 2). The loop on Line 5 executes n times because the problem statement said that n is the number of blocks per row.

When we're analyzing loops, we're trying to determine (or approximate) the number of iterations for the loop relative to n. Because a nested loop executes for each iteration of the outer loop, we need to multiply the number of iterations for the nested loop by the number of iterations for the outer loop to calculate the total number of iterations. When we do that for this example, we get a total of 3n iterations for the loops. Because we throw away the 3 for our complexity analysis, the loops have O(n) complexity.

Finally, the O(n) complexity for the loops dominates all the O(1) operations in the code. That means this code excerpt is O(n) where n is the number of blocks per row.

Problem 2 – Wacky Analysis, Take 2

Analyze the algorithmic complexity of the code where n is the total number of blocks in the level.

Solution

The only difference for this problem is how we define n. That means the only analysis we need to re-do is for the loop on Line 5; the loop on Line 3 still executes 3 times and all the other operations are still O(1).

Because we've redefined n to be the total number of blocks in the level, the loop on Line 5 now executes n / 3 times. Multiplying the iterations for the loop on Line 3 by the iterations for the loop on Line 5 yields a total of n iterations for the loops.

That means this code excerpt is still O(n) where n is the total number of blocks in the level.