Cella Ant x15 Big-O Analysis

Jared Dyreson
California State University, Fullerton

Mason Godfrey, California State University, Fullerton
Brian Lucero, California State University, Fullerton

2020
September

Contents

1 Big-O Analysis

1.1 General Program Flow

Our implementation of Cella Ant runs in at $O(n^2)$ time for the initial setup, as we need to create a grid on which the Ant resides on. After the first draw, the program will have a time efficiency of $O(1)$, as we're only accessing a specific element in a 2 x 2 matrix. This however does not imply that it only takes on instruction. The evaluation depends on the notion that the algorithm depends on a fixed number of instructions and the most general case is in fact $O(1)$. Our program also has a fixed input size of 41 and 41, determining the size of the matrix. This program could be expanded to further support other dimensions, however it needs to take a factor λ that will properly account for resizing.

1.2 Ant Direction

One of the critical points we needed to address was the direction in which the ant was going to take. Thankfully there was a quick solution from bit manipulation, which has an efficiency of $O(1)$. The code snippet is seen below:

```
var antDir = (this.hex % (32 >> cellState)) >> (4 - cellState);
```

The time efficiency is only $O(1)$ because it simply relies on a constant "cellState", which helps go to the specific bit index of 0x15.

1.3 Memory Usage

Since we need to keep track of the cells present on the screen, we need to save these cell instances. In design, we wanted to use the least amount of space possible and as a result, we mostly used small footprint classes. Each attribute could be traced to either an integer or a simple enumeration class. The main goal was to have the fastest runtime while not exceeding a predefined amount of space, directly in relation with the time complexity stated earlier.

1.4 Integer Handling

Since we mostly used integers ranging from 0 to 3, we could take advantage of switch statements. By design, these have a time complexity of $O(1)$ as each number has a specific outcome without any extra processing. If we did not use this, the other option would have been to rely on either relational arrays or several if statements. These would have eaten away at our goal of time and space complexity respectively.