Team: ARS

BigO Analysis

Both the TP ants have a Big O of O(n^1), which means that they are linear. Because the sequence of steps must be shown sequentially, the algorithm which produces such a pattern must be linear. In other words, it can not be sub linear.

Our algorithm are both linear and fundamentally similar enough to discuss together so we are going to show the Big O analysis of both of them and explain the differences between the algorithms and what impact that has on resources used by it.

The TP ant algorithms both can be considered to have linear time complexity only if the input size is known when the algorithm is initialized even if it is arbitrarily large. This is because the algorithm uses a two dimensional bitmap array to track the state of potions the ant has visited before. The algorithm essentially modifies this array until it hits an edge of the array at which point it breaks. Fortunately, because this was written in scripting languages rather than a compiled language, the code breaking has not affected what is already displayed. Though the size of the array is set though experimination, it can be of any arbitrary size. Its size directly affects how many steps can be shown. When the number of steps needed to be shown, N, goes to infinity the consumption of space needed by the algorithm grows at a rate of N^2. This is because for every step that ‘leaves’ the grid, the number of ‘rows’ and ‘columns’ that needed to be shown increases by one, resulting in the defined relationship. If the algorithm needs to handle an infinite input size then for each ‘step’ taken beyond the array, a new array would have to be created and copied over, which has a time complexity of N^2 since the array size is N^2. However, because we only have to show the ant taking an arbitrary amount of steps, the array can be initialized before the algorithm starts to run. Because we set the array size initially and don’t grow it, it can be considered overhead for our algorithm and have a time complexity of O(1).

Assuming the necessary resources are allocated for the algorithm to run, it runs as follows. First the current indices of the two dimensional array, which represent the current position of the ant, are used to determine the integer value of that position. This integer value can be between zero and one for the TP ant and zero to three for the TP ant 12. The direction of the ant, which is a variable, is then updated by logic, which accounts for both the current value of the direction and the value of that position. Then, the value of the position is incremented and if it exceeds the range, it is wrapped back to zero. Then, a cube is drawn on the canvas. The cube’s position directly corresponds to the cube’s indexes and the color corresponds to the value of that position. Then, using the current direction’s value, a corresponding increment or decrement of one of the indices is made. A new ant is then drawn on the canvas whose position directly corresponds to the updated indices. In the case of ant 12, the current direction is also used to determine the shape of the ant drawn. After determining the shape, the cycle continues until the function finally stops when the contents that exceed the boundaries of the array are called. All of the components of the function described are ordered and sequential, so they can be said to have a value of +1 when describing them in BigO. This means that they are negligible, which means that this part of the algorithm has a BigO of (N). All of the differences mentioned between the two algorithms have no

impact on the resulting BigO because they essentially just add +1 to the run time, which are considered negligible and thrown away.