Tutorial – Algorithm Efficiency and Big-O Notation

Introduction:

In this tutorial we will measure the efficiency of the *Connect Four checkForWin* algorithm we wrote in the previous tutorial.

Requirements:

It is recommended you complete the previous tutorial on *Algorithms* before attempting this tutorial.

If you have not created your own algorithm for the *checkForWin()* function, you can use the one described in the *Algorithms* tutorial. Since we will only be analysing the efficiency of this algorithm, there is no requirement for you to have completed the code implementation of this function.

How Efficient is Our Algorithm?:

The following was the pseudo-code for the *checkForWin()* function, as described in the previous tutorial:

```
Function: checkForWin()
      For each tile on the x axis
            For each tile on the y axis
                  Get the tile at location x,y
                  If the tile is empty, continue to the next tile
                  If the next 3 tiles to the right match the current
                        tile, return true
                  If the previous 3 tiles to the left match the current
                        tile, return true
                  If the 3 tiles above the current tile match the
                        current tile, return true
                  If the 3 tiles below the current tile match the
                        current tile, return true
                  If the 3 tiles diagonally to the right and down match
                        the current tile, return true
                  If the 3 tiles diagonally to the left and down match
                        the current tile, return true
                  If the 3 tiles diagonally to the right and up match
                        the current tile, return true
                  If the 3 tiles diagonally to the left and up match
                        the current tile, return true
      return false
```

This function executes whenever a new tile is placed on the board. It will look at every tile on the board, looking for the first instance of a series that matches the win condition (four matching tiles in a row).

When we consider the efficiency of this algorithm, we do so in relation to the data set it executes on.

In our case the data set is the game board, with each tile on that board representing a piece of data.

We can simplify our algorithm by grouping the series of '*if*' statements into a single unit of processing and calling this a 'check for a match'. Our algorithm can then be described simply as "for every tile, check for a match".

The nested *for* loop may make you think that we are looping through our data more than we actually are. The outside loop iterates over the columns, while the inside loop iterates over the rows. The end result is that (in the worst case) we look at every tile on our board when checking for a match.

If, in the worst case, we need to check each element of data (i.e., every tile on the board) when searching for a match, then we can write our algorithm's efficiency as:

O(n)

Consider for a moment that we wanted to increase the size of our game board by one extra column. Our board grows from its current size of 42 (6*7) to 48 (6*8). According to our algorithm's efficiency O(n), what should we expect when we run the algorithm using the new board size. Do the number of additional data items processed match the formula?

> Answer: yes, it does. Increasing the board by 6 additional tiles means we need to check 6 more tiles when searching for a match.

Challenge:

If you came up with your own algorithm during the last tutorial, measure its efficiency and write it in big-O notation. Is it more or less than O(n)?

Can you come up with an algorithm that is more efficient? Remember, the *if* statements within the inside loop are counted as one operation. Decreasing the number of *if* statements won't increase the algorithms efficiency. Only reducing the number of tiles we need to check will change the algorithm's efficiency.

Can you think of how we can modify our program so that the *checkForWin()* function has O(1) efficiency?