Alexander Rodriguez

Turn in a one-paragraph commentary with your code. Describe your matrix implementation. Why did you choose the implementation you did? What is the computational complexity of the operations in your matrix implementation?

To start I created my constructors, getters, and setters. After that I began with my addElement method which also organizes the row and column values, focusing on row values first and then the column values. All of these values were stored in the LinkedList class that I imported. This list contained the object Node, which was a class I created to hold a row, column, and data value. The time complexities of the algorithms in this method were O(n), usually in the case where a value is not found. I used a stack to organize my list, but since the amount of times a value was placed into the stack depended on an “index” or starting point, the time complexity was O(n). I chose this implementation so other functions, such as the toString function, would be better able to go through the list and find values. Every other method also had a time complexity of O(n) except the determinant function. The determinant function had a time complexity of O(n!). This is because every time “minor(row, col).determinant” is called, it would have another (n-1)! steps to do, where n is the size of the original size of the matrix. I chose this implementation on the determinant function because it allowed me to make a base case of a 1x1 matrix and return the remaining value. I did not see any other way to implement a recursive function without going through the length of the matrix’s size.