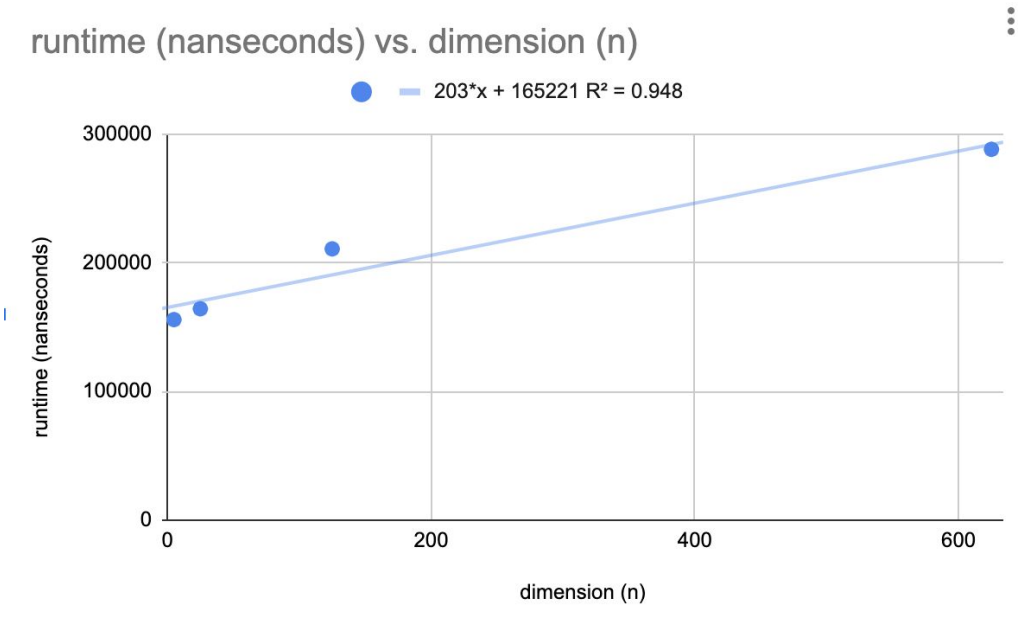
Theoretical analysis:

The growth of the program is O(n). In the boolean inMatrix method, the run time is O(n) because I used while loop based on the size of the matrix. The time complexity of the method is based on the size of the matrix. If n is 4, it takes 7 steps to get to 9 which is the worst-case scenario; if n is 5, then it takes 9 steps to get to the worst-case scenario. Thus, we know the iteration of the function would be 2n-1 which we can just say is O(n).

Empirical Analysis:

The class named sortedMatrix was designed for empirical analysis. Please find the code in all the files. I generated 4 matrices that are 5 by 5, 25 by 25, 125 by 125, and 625 by 625 to graph the corresponding runtime. I maximized runtime for each matrix by picking the worst-case scenario which is the value on the bottom left of the matrix.

Dimensions * Dimensions	Runtime difference in nanoseconds
5 by 5	155805.0
25 by 25	164220.0
125 by 125	210750.0
625 by 625	288098.0



Based on the graph, we can tell that the data is pretty linear. I picked the values 5, 25, 125, 625, which increase the previous value by 5 times to test the efficiency. The time complexity increases proportionally as the dimension increases by 5 times. This is very efficient that as n gets larger, the runtime gets larger proportionally instead of drastically increasing out of proportions. This is also very much more efficient than going through each column and each row by creating nested loops, which have a time complexity of O(n^2).
Thus, based on both theoretical analysis and empirical analysis, we can conclude that the time complexity for this program is O(n).