**Complexity of Functions in Project 2**

In all cases we consider n to be the largest row or column size:

0: Insert 🡺 O(n)

**84** while (n.nextInRow != null)}

Here, after checking the row to see if our house has a value or not, we start adding the number to house 0. Then we go to check the columns:

**106** while (n.nextInColumn != null) {

In worst Case in both While loops are executed n times, so we can understand that the complexity is O(n).

1: Delete 🡺 O(n)

In this function, after checking that we do not delete 0, we consider the worst case, which is the same as the Insert function, two loops are executed n times, so complexity will be O(n).

2: Search 🡺 O(n^2)

**158** for (Node matrix : rowMatrix)

**161** while (n != null)

Here, we have a for loop and a while that go through rows and columns, respectively, which are again executed n times in the worst case, which complexity becomes O (n ^ 2) due to nesting.

3: Update 🡺 Mine: O(n^2) – Correct Way: O(n)

Here my code has an error that we still go through the line despite having the line number, however same as search function one for and one while are executed n times and the complexity of the function is of order O (n ^ 2). But in the optimal method, we go through columns only with while, without going through rows, which is executed n times (number of columns), so the complexity of the function is of order

O (n).

4-0: Print 🡺 O(n^2)

**23**for (Node n : rowMatrix) {  
 int last = 0;  
 if (n == null) {  
 for (int i = 0; i < columnMatrix.length; i++) {  
 System.*out*.printf("%4s", 0);  
 }  
 }  
 while (n != null) {  
 for (int i = last; i < n.columnIndex; i++) {  
 System.*out*.printf("%4s", 0);  
 }  
 last = n.columnIndex + 1;  
 System.*out*.printf("%4s", n.data);  
 n = n.nextInRow;  
 if (last < columnMatrix.length && n == null) {  
 while (last < columnMatrix.length) {  
 System.*out*.printf("%4s", 0);  
 last++;  
 }  
 }  
 }  
 System.*out*.println();  
}

In this function, we go through the rows once with forEach and in each row we move through the columns and print zeros between two non-zero elements, which is executed n times in both cases, which becomes n ^ 2. So the complexity of the function is O (n ^ 2).

4-1: Print 🡺 O(n^2)

**14**

for (Node n : rowMatrix) {  
 while (n != null) {  
 System.*out*.printf("%5s %5s %5s%n", n.rowIndex, n.columnIndex, n.data);  
 n = n.nextInRow;  
 }  
}

In this function, the while loop, which itself is repeated n times (number of columns) inside the forEach loop, which is the number of rows, is repeated n times, which is a total of n \* n, which can be said that the complexity of the function is O (n ^ 2).

5: save file 🡺 O(n^2)

This function, like printing: code Zero, creates the same matrix in the Output file, which does not differ in time complexity, and again it becomes O (n ^ 2).