MATH 282 Analysis of Algorithm’s Complexity

**Algorithm:** Gauss-Jordan elimination

**Factor to be analyzed:** Time (number of steps, speed)

**Situation to be analyzed:** Worst case

**Explanation of situation:** Every pivot row has to be swapped

**Key step (reflects work done):** Matrix access

**Parameter for analysis:** *n* is the number of equations/unknowns

*(what determines how the algorithm’s efficiency changes as the size of data increases?)*

**Questions/Process:**

* Are there any steps which are not simple steps (comparable to the key step)?
  + If so, what is the complexity of those steps (relative to the key step)? Factor into the analysis.
* If desired, count the number of times each step is carried out (or just the key step).
* Identify each loop and determine how many times the loop is carried out (in relation to *n*).
* How are the loops related?
  + If nested, multiply the steps.
  + If separate, add the steps.
* Eliminate any constants and any lower-level terms.

Creating a new matrix involves *n*2 steps.

The loop to go through every pivot row in the matrix is carried out *n* times. For each pivot row, we must:

* Go through SystemSolveable
  + One matrix access at the start, then it loops through every remaining row (so *n* – 1 loops the first time, then *n* – 2, then *n* – 3, and so on down to 0) – treat this as if the loop was done *n* times.
  + One matrix access inside the loop – is done *n*2 times for all rows in total
  + Then we do a loop (*n* + 1) times to swap rows, but it is NOT nested inside the previous loop – done once for every pivot row, so 4 \* (*n* + 1) \* *n* matrix accesses necessary to switch rows for all rows in total
  + So SystemSolveable involves *n*2 + 4*n*2 + 4*n* + *n* matrix accesses, so it is O(*n*2)
  + SystemSolveable is done now (not nested inside any other loop)
* We then get the pivot element (1 matrix access)
  + Go through the pivot row and access every element twice for every column, so there are 2 \* (*n* + 1) matrix accesses for all *n* of the pivot rows, so *n* \* 2 \* (*n* + 1) matrix accesses to divide each pivot row element for all rows in total, so 2*n*2 + 2*n* matrix accesses, so it is O(*n*2)
* Then for the remaining rows or *n* – 1 loops, we get the factor (1 matrix access), so that’s carried out *n* \* (*n* – 1) times
  + Then there are 3 matrix accesses in for every column in every remaining row for every pivot row, so that’s *n* \* (*n* – 1) \* (*n* + 1) columns
  + So getting 0’s in every column other than pivot row involves  
    *n* \* (*n* – 1) + 3 \* *n* \* (*n* – 1) \* (*n* + 1) steps, which is roughly *n*2 + 3*n*3 steps, so it is O(*n*3)

So the entire process takes O(*n*2) + O(*n*2) + O(*n*2) + O(*n*3) steps, and thus the entire process is O(*n*3)

Shortcut analysis: There are two nested loops (O(*n*2) for SystemSolveable, 2 nested loops (O(*n*2)) for the pivot row, and 3 nested loops with each loop being done roughly *n* times for getting the 0’s (O(*n*3)).

**Result:** Algorithm is O( *n*3 )