**The Dual Simplex Problem**

Almost done with Simplex. But need to solve *minimization* problem.

Example: Minimize 2x + 3y subject to

x + y >= 2  
3x + y >= 3  
x + 4y >= 4

Side note: What if I have a problem with an inequality that goes the wrong way? Multiply by -1:

Max x + y subject to x + y <= 5 and x – y >= 1, same as –x + y <= -1

Setup coefficient matrix A:

1 1 2  
3 1 3  
1 4 4  
2 3 1

Transpose A:

1 3 1 2  
 1 1 4 3  
 2 3 4 **1**

Call these columns y1, y2, y3 and insert 2 by 2 identity matrix before the last column, fill in zeros in the last row:

1 3 1 1 0 2  
 1 1 4 0 1 3  
 2 3 4 0 0 1

This matrix is the \*dual\* matrix to the original problem and you can solve it with the usual Simplex method. The first cols are labeled y1, y2, y3, but the next two columns are labeled x and y. When the simplex process is done, the value of x is in the last row of the “x” column, the value for y is in the last row of the “y” column!!!

**How good is an Algorithm?**

How do you measure “goodness”? Define “Big-O” notation:

**Def**.: A function f is Big-O of another functiong – we say that f = O(g) – if  
 lim as x -> infinity f/g = r and 0 < r < infinity

**Examples**: f(x) = x^2 + 2x -9, g(x) = 7 x^2 -9x +115. Then: is f = O(g)??? lim x->infinity f(x) / g(x) = 1/7 YES

Is f = O(x^3) ? NO. Is f = O(x)? NO. Is f = O(x^2) YES!!!

Ex.: f(x) = 7 x^4 – 9x^2 + 15. Then f = O(x^4)

So Big-O measures how fast a function goes to infinity as x goes to infinity!!!

Ex: f(x) = 2^x + x^200,000. Then f = O(2^x)

Computer algorithm are measured in Big-O notation, where the x is often replaced by and n, and the n stands for “size” of the problem.

Example: Vector addition is big-O of what? Vector addition requires n operations, thus it is O(n)

Matrix addition is big-O of what? Requires n \* n operations, thus it is O(n^2)

Matrix multiplication is big-O if what? O(n^3)

Big-O notation for algorithms is also called the “order of complexity” of an algorithm!!!

Order of complexity for finding det of a matrix A? Our standard algorithm is O(n!).

Goal is: find the best algorithm to solve a problem, i.e. the algorithm with the smallest order of complexity!!e to O(n).

**Simple Search Algorithm:**

Find a number in an array: always assume the worst case, i.e. the number you are searching for is not in the list. Then searching is O(n).

**Better Algorithm:**

Assume the array is sorted. What is the order of complexity now: O(n). Can you devise a better algorithm?

1, 3, 5, 7,11, 12, 12, 12, 13, 19, 21, 25, 33, 35, 36, 37

Find 8: check the middle number. If it is 8, done. If the middle number is greater than 8, then 8 can only be on the left, else could be only on the right. Repeat! Thus, at each step you cut the number of remaining searches in half. What is the order of complexity of this *binary search* algorithm?

**Homework:**

1. Find order of complexity of the binary search algorithm