T: he following is the code for pivoting *LU* factorization

Code:

```
function [L,A] = LUPivot(A)
%This funciton takes an NxN matrix A and returns an LU factorization
% without pivoting the rows
n = size(A,2);
L = zeros(n);


for k = 1:n %Initializes the diagonal of L
    L(k,k) = 1;
end


for i = 1:n-1 % Iterates through columns of A


    %U pivot
     [M, I] = max(A(i,:));
     tmp = A(i,:);
     A(i,:) = A(I,:);
     A(I,:) = tmp;

     %L Pivot
     tmp = L(i, 1:i-1);
     L(i, 1:i-1) = L(I, 1:i-1);
     L(I, 1:i-1) = tmp;


    for j = i+1:n % Iterates through Rows of A



        x = A(j,i)/A(i,i); %Calculates factor for Gauss Elim

        L(j,i) = x; %Stores factor in L

        for k = 1:n % Iterates through current row and performs Gaussian
            A(j,k) = A(j,k) - (A(i,k)*x);
        end

    end
```

```
end
```

Supplemental 1: Write a function to compute the inverse of a *nxn* matrix *A*.

1. Let b_1 be column i of A^{-1}. What are the entries of Ab_i? Hint: most of them are zero! Use the column perspective of matrix multiplication.

   Solution:
   Given that when we multiply a matrix by its inverse we get the identity matrix, let's consider the column perspective of the following equation,

$$AA^{1} = I$$
$$[Ab_1, Ab_2, \ldots, Ab_n] = [i_1, i_2, \ldots 1_n]$$

   Therefore it must be the case that $Ab_1 = i_1$ which is a one-hot column vector.

2. The following code addresses the part 2 and 3,
   Code:

```
function [B] = inverse(A)

n = size(A,2); %Grabs dimension
B = zeros(n); %Initilises B
[L,U] = LUPivot(A); %Calls LU Factor


for j = 1:n

    %Makes I_n vector
    i = zeros(n, 1);
    i(j) = 1;

    %Calls U and L Solve to get column of A^{-1}
    y = lsolve(L,i);
    x = usolve(U,y);
    %Saves each column
    B(:, j) = x;


end
```

```
end
```

Supplemental 2: Determine, with justification, the number of floating point operations required to compute the inverse of a matrix using the strategy of the previous problem. A complete answer will be of the form,

$$cn^j + O(n^k)$$

where c is an explicit number, and where j, k are explicit integers with $j > k$.

Solution:
Our method for computing the inverse of A required and LU factorization, a L solve and a U solve. We showed in the Counting FLOPS that an L solve takes $n^2 - n$ operations and that, a U solve takes n^2 operation. As a reminder we showed that for each x_n in the L solve we said that there were $n - 1$ multiplications and $n - 1$ subtractions, then we used the gauss formula to get,

$$\sum_{i=1}^{n} 2i - 2 = n^2 - n.$$

Similarly we can count the FLOPS for a U solve, but a faster way is noticing that for each x_n there are $2n - 1$ operations because of the extra division operation, and when summed over n there are n more operation thus n^2. Counting the operations for the LU factorization we get that to clear the first column we first find our multiple, then subtract a multiple of row 1 from row 2. Since the first entry will always be 0 we now there are $n - 1$ multiplications and subtractions. Doing thus for all the $n - 1$ rows to clear the first column leaves us with,

$$(n - 1)(2(n - 1) + 1) = 2(n - 1)^2 + (n - 1).$$

Now we sum over the n columns. Recall that this sum was calculated in class and it is of the form,

$$\sum_{j=1}^{n} 2(j - 1)^2 + (j - 1) = \frac{2}{3}n^3 + a_2n^2 + a_1n.$$

Where a_2, a_1 are constants. Note that our method requires one LU factorization and n U solves and n L solves. Summing over our operations,

$$\frac{2}{3}n^3 + a_2n^2 + a_1n + n(n^2) + n(n^2 - n) = \frac{2}{3}n^3 + a_2n^2 + a_1n + 2n^3 - n^2 = \frac{8}{3}n^3 + O(n^2).$$

Supplemental 3: How many $6x6$ permutations exist.

Solution:
By definition a permutation matrix has only one $'1'$ in each row and column. Adding a $'1'$ to each row a row at a time. the first row there are 6 columns to choose from. When we move to the second row there is one less column to choose from so there are only 5 spots.Thus there are only 6! = 720 permutation matrices.

Supplemental 4: A permutation matrix can be represented by a vector $[p_1, \ldots p_n]$ where p_i records which column contains the $'1'$ in row i. Modify *lsolve* to make *plsolve* so it returns,

$$Lc = Pb.$$

Code:

```
function y = lpsolve(L,P, b)
% Given a lower triangular matrix L with unit diagonal
% and a vector b,
% this routine solves Ly = b and returns the solution y.



n = size(b,2);        % Determine size of b.
bp = zeros(n,1);

for i = 1:n           % Compute permutatino of
    pb(i) = b(P(i));
end




for i=1:n      % Loop over equations.
    y(i) = pb(i);    % Solve for y(i) using

    for j=1:i-1        % previously computed y(j),
                         % j=1, . . . ,i-1.
        y(i) = y(i) - L(i,j)*y(j);
    end
end                    % j=1, . . . ,i-1.
```