```
julia> A=[2 0 4 3 ; -4 5 -7 -10 ;
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -4.0   5.0  -7.0  -10.0
  1.0  15.0   2.0   -4.5
 -2.0   0.0   2.0  -13.0
```

```
julia> Atrouble=[2 0 4 3 ; -2
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0
```

same matrix

```
julia> p=[1,4,3,2]
4-element Vector{Int64}:
 1
 4
 3
 2
```

permutation

```
julia> Atrouble[p,:]
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -4.0   5.0  -7.0  -10.0
  1.0  15.0   2.0   -4.5
 -2.0   0.0   2.0  -13.0
```

permutation matrix: Find the matrix representing the linear operation by performing that op. on the identity matrix

```
julia> using LinearAlgebra

julia> diagm(ones(4))
4×4 Matrix{Float64}:
 1.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0
 0.0  0.0  1.0  0.0
 0.0  0.0  0.0  1.0
```

← Identity matrix

This is more efficient that writing PA.

permutation matrix

```
julia> P=diagm(ones(4))[p,:]
4×4 Matrix{Float64}:
 1.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0
 0.0  0.0  1.0  0.0
 0.0  1.0  0.0  0.0
```

```
julia> P*A
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0
```

```
julia> A=[2 0 4 3 ; -4 5 -7 -10 ;
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -4.0   5.0  -7.0  -10.0
  1.0  15.0   2.0   -4.5
 -2.0   0.0   2.0  -13.0
```

```
julia> p=[2,3,4,1]
4-element Vector{Int64}:
 2
 3
 4
 1

julia> A[p,:]
4×4 Matrix{Float64}:
 -4.0   5.0  -7.0  -10.0
  1.0  15.0   2.0   -4.5
 -2.0   0.0   2.0  -13.0
  2.0   0.0   4.0    3.0
```

$p = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}$

row 2 is called row 1

row 3 is called row 2

still the same..

```
julia> P=diagm(ones(4))[p,:]
4×4 Matrix{Float64}:
 0.0  1.0  0.0  0.0
 0.0  0.0  1.0  0.0
 0.0  0.0  0.0  1.0
 1.0  0.0  0.0  0.0
```

PA

```
julia> P*A
4×4 Matrix{Float64}:
 -4.0   5.0  -7.0  -10.0
  1.0  15.0   2.0   -4.5
 -2.0   0.0   2.0  -13.0
  2.0   0.0   4.0    3.0
```

```
function mylu(A)
        m,n=size(A)
        if m!=n
                println("Need a square matrix!")
                throw(exit())
        end
        U=zeros(size(A))
        L=zeros(size(A))
        Ak=copy(A)
        for k=1:n
                U[k,:]=Ak[k,:]
                L[:,k]=Ak[:,k]/U[k,k]
                Ak=Ak-L[:,k]*U[k,:]'
        end
        return L,U
end
```

```
julia> include("mylu.jl")
mylu (generic function with 1 method)

julia> L,U=mylu(A)
([1.0 0.0 -0.0 0.0; -2.0 1.0 -0.0 0.0; 0.5 3.0 1.0 0.0; -1.0 0.0 -2.0 1.0], [2.0
 0.0 4.0 3.0; 0.0 5.0 1.0 -4.0; 0.0 0.0 -3.0 6.0; 0.0 0.0 0.0 2.0])
```

```
julia> L
4×4 Matrix{Float64}:
  1.0   0.0   -0.0   0.0
 -2.0   1.0   -0.0   0.0
  0.5   3.0    1.0   0.0
 -1.0   0.0   -2.0   1.0

julia> U
4×4 Matrix{Float64}:
 2.0   0.0   4.0   3.0
 0.0   5.0   1.0  -4.0
 0.0   0.0  -3.0   6.0
 0.0   0.0   0.0   2.0
```

```
julia> norm(L*U-A)
0.0
```

verification that A=LU

```
julia> L,U=mylu(Atrouble)
([1.0 NaN NaN NaN; -1.0 NaN NaN NaN; 0.5 Inf NaN NaN; -2.0 Inf NaN NaN], [2.0 0.
0 4.0 3.0; 0.0 0.0 6.0 -10.0; NaN NaN -Inf Inf; NaN NaN NaN NaN])

julia> L
4×4 Matrix{Float64}:
  1.0  NaN   NaN  NaN
 -1.0  NaN   NaN  NaN
  0.5   Inf  NaN  NaN
 -2.0   Inf  NaN  NaN

julia> U
4×4 Matrix{Float64}:
   2.0    0.0    4.0    3.0
   0.0    0.0    6.0  -10.0
 NaN    NaN    -Inf    Inf
 NaN    NaN    NaN    NaN
```

Problem: Not all matrices have an LU factorization.

In Gaussian elimination solve the problem with row swaps... Not surprising that caused the problem in this example

```
function mylu(A)
        m,n=size(A)
        if m!=n
                println("Need a square matrix!")
                throw(exit())
        end
        U=zeros(size(A))
        L=zeros(size(A))
        Ak=copy(A)
        for k=1:n
                U[k,:]=Ak[k,:]
                L[:,k]=Ak[:,k]/U[k,k]
                Ak=Ak-L[:,k]*U[k,:]'
        end
        return L,U
end
```

denominator is zero...

```
println("About to divide by ",U[k,k])
```

add diagnostic...

```
julia> L,U=mylu(Atrouble)
About to divide by 2.0
About to divide by 0.0
About to divide by -Inf
About to divide by NaN
([1.0 NaN NaN NaN; -1.0 NaN NaN NaN; 0.5 Inf NaN NaN; -2.0 Inf NaN NaN], [2.0 0.
0 4.0 3.0; 0.0 0.0 6.0 -10.0; NaN NaN -Inf Inf; NaN NaN NaN NaN])
```

trouble happened here...

consequences of that trouble...

Idea: swap the rows so don't divide by zero

Better Idea: swap the rows so we divide by the thing that is farthest from zero...

```
julia> Atrouble
4×4 Matrix{Float64}:
  2.0    0.0    4.0    3.0
 -2.0    0.0    2.0  -13.0
  1.0   15.0    2.0   -4.5
 -4.0    5.0   -7.0  -10.0
```

swap 1st row with last row so the first division is as far from zero as possible..

way to search for that row

```
i = argmax( abs.(A₁[:,1]) )
```

apply abs pointwise

```
julia> Atrouble[:,1]
4-element Vector{Float64}:
  2.0
 -2.0
  1.0
 -4.0
```

```
julia> abs.(Atrouble[:,1])
4-element Vector{Float64}:
 2.0
 2.0
 1.0
 4.0
```

searches the vector and return the index of the largest element..

```
julia> argmax(abs.(Atrouble[:,1]))
4
```

```
function myplu(A)
    m,n=size(A)
    if m!=n
        println("Need a square matrix!")
        throw(exit())
    end
    U=zeros(size(A))
    L=zeros(size(A))
    Ak=copy(A)
    for k=1:n
        i=argmax(abs.(Ak[:,k]))
        U[k,:]=Ak[i,:]
        println("About to divide by ",U[k,k])
        L[:,k]=Ak[:,k]/U[k,k]
        Ak=Ak-L[:,k]*U[k,:]'
    end
    return L,U
end
```

display(Ak) add.

i is row of column k with largest magnitude.

a new A matrix

```
julia> L,U=myplu(Atrouble)
About to divide by -4.0
About to divide by 16.25
About to divide by 5.538461538461538
About to divide by -0.1666666666666664
([-0.5 0.15384615384615385 0.08333333333333
 -0.0; -0.25 1.0 0.0 -0.0; 1.0 0.0 0.0 -0.
25 -7.0; 0.0 0.0 5.538461538461538 -9.0769
666664])
```

```
julia> Atrouble
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0
```

```
julia> include("mylu.jl")
myplu (generic function with 1 method)

julia> L,U=myplu(Atrouble)
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0
About to divide by -4.0
4×4 Matrix{Float64}:
 0.0   2.5   0.5   -2.0
 0.0  -2.5   5.5   -8.0
 0.0  16.25  0.25  -7.0
 0.0   0.0   0.0    0.0
About to divide by 16.25
4×4 Matrix{Float64}:
 0.0  0.0  0.461538  -0.923077
 0.0  0.0  5.53846   -9.07692
 0.0  0.0  0.0        0.0
```

3 is index of the largest entry is the second column.

```
julia> L
4×4 Matrix{Float64}:
 -0.5     0.153846  0.0833333   1.0
  0.5    -0.153846  1.0        -0.0
 -0.25    1.0       0.0        -0.0
  1.0     0.0       0.0        -0.0

julia> U
4×4 Matrix{Float64}:
 -4.0    5.0    -7.0       -10.0
  0.0   16.25    0.25       -7.0
  0.0    0.0     5.53846    -9.07692
  0.0    0.0     0.0        -0.166667
```

```
julia> L*U
4×4 Matrix{Float64}:
  2.0   1.38778e-16   4.0    3.0
 -2.0  -1.38778e-16   2.0  -13.0
  1.0  15.0           2.0   -4.5
 -4.0   5.0          -7.0  -10.0

julia> Atrouble
4×4 Matrix{Float64}:
  2.0   0.0   4.0   3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0
```

no longer the correct shape of matrix

```
function myplu(A)
    m,n=size(A)
    if m!=n
        println("Need a square matrix!")
        throw(exit())
    end
    U=zeros(size(A))
    L=zeros(size(A))
    p=zeros(Int,n)
    Ak=copy(A)
    for k=1:n
#       display(Ak)
        i=argmax(abs.(Ak[:,k]))
        p[k]=i
        U[k,:]=Ak[i,:]
        println("About to divide by ",U[k,k])
        L[:,k]=Ak[:,k]/U[k,k]
        Ak=Ak-L[:,k]*U[k,:]'
    end
    return L[p,:],U
end
```

```
julia> L
4×4 Matrix{Float64}:
  1.0     0.0        0.0         -0.0
 -0.25    1.0        0.0         -0.0
  0.5    -0.153846   1.0         -0.0
 -0.5     0.153846   0.0833333    1.0

julia> L*U
4×4 Matrix{Float64}:
 -4.0   5.0          -7.0  -10.0
  1.0  15.0           2.0   -4.5
 -2.0  -1.38778e-16   2.0  -13.0
  2.0   1.38778e-16   4.0    3.0
```

now L has the right shape, but LU gives the matrix with rows in wrong order...