Algorithm 28.1. "Pure" QR Algorithm

$A^{(0)} = A$

for $k = 1, 2, \ldots$

$Q^{(k)}R^{(k)} = A^{(k-1)}$ QR factorization of $A^{(k-1)}$

$A^{(k)} = R^{(k)}Q^{(k)}$ Recombine factors in reverse order

```
Ak=Matrix(A)
for k=1:100
    Q,R=qr(Ak)
    global Ak=R*Q
end
```

in Julia the algorithm looks like this

$A^{(0)} = A$

$Q^{(1)}R^{(1)} = A^{(0)}$

$A^{(1)} = R^{(1)}Q^{(1)}$

$Q^{(2)}R^{(2)} = A^{(1)}$

$A^{(2)} = R^{(2)}Q^{(2)}$

$Q^{(1)}R^{(1)} = A^{(0)}$ $\quad R^{(1)} = (Q^{(1)})^T A^{(0)}$

$A^{(1)} = R^{(1)}Q^{(1)} = (Q^{(1)})^T A^{(0)} Q^{(1)}$

Thus $A^{(1)}$ is related to $A^{(0)}$ through a similarity transformation involving $(Q^{(1)})^T$ and therefore $A^{(1)}$ has the same eigenvalues as $A^{(0)}$.

$A^{(2)} = R^{(2)}Q^{(2)} = (Q^{(2)})^T A^{(1)} Q^{(2)}$

Thus $A^{(2)}$ has the same eigenvalues and eigenvectors as $A^{(1)}$ and so as $A^{(0)}$

The amazing thing, as you iterate this, is that $A^{(n)} \to$ a diagonal matrix (If lucky).

Although the algorithm works for general matrices, there is a problem with complex eigenvalues coming in conjugate pairs that have the same magnitude. In that case one needs to break the conjugate symmetry by adding a shift in the imaginary direction...

Simpler assume $A^T = A$ for now to avoid complex.

```
julia> A=rand(4,4)
4×4 Matrix{Float64}:
 0.0424851  0.385023   0.364093  0.511945
 0.835597   0.0134645  0.263444  0.602239
 0.15474    0.122023   0.640464  0.913725
 0.336913   0.771946   0.651124  0.818216

julia> A=A+A'
4×4 Matrix{Float64}:
 0.0849703  1.22062   0.518834  0.848859
 1.22062    0.026929  0.385468  1.37419
 0.518834   0.385468  1.28093   1.56485
 0.848859   1.37419   1.56485   1.63643
```

```
julia> Ak=Matrix(A)
       for k=1:100
           Q,R=qr(Ak)
           global Ak=R*Q
       end
```

```
julia> Ak
4×4 Matrix{Float64}:
 4.00688        -2.0782e-16     8.10313e-16    3.36335e-16
 1.16691e-48   -1.3003          5.02323e-17   -3.38295e-16
 1.08858e-79    8.6064e-33      0.640657       6.39867e-17
 3.4195e-110    2.86896e-62   -4.41647e-33   -0.31798
```

The eigenvalues of a diagonal matrix are just on the diagonal...

```
julia> lambdas100=diag(Ak)
4-element Vector{Float64}:
  4.0068787252906
 -1.3002967911550671
  0.6406566068270437
 -0.3179793105343564
```

```
julia> Ak=Matrix(A)
       for k=1:10
           Q,R=qr(Ak)
           global Ak=R*Q
       end

julia> Ak
4×4 Matrix{Float64}:
 4.00688       0.000113702   4.93334e-8    3.72006e-11
 0.000113702  -1.3003        4.00284e-5    3.20313e-7
 4.93334e-8    4.00284e-5    0.640657     -1.06018e-5
 3.72003e-11   3.20313e-7   -1.06018e-5   -0.31798

julia> lambdas10=diag(Ak)
4-element Vector{Float64}:
  4.0068787228546086
 -1.3002967878934637
  0.6406566058842909
 -0.31797993093629273
```

```
julia> sort(eigvals(A))
4-element Vector{Float64}:
 -1.300296791155067 8
 -0.317979931053436
  0.6406566068270442
  4.006878725290598

julia> sort(lambdas100)
4-element Vector{Float64}:
 -1.300296791155067 1
 -0.31797993105343564
  0.6406566068270437
  4.0068787252906

julia> norm(sort(eigvals(A))-sort(lambdas100))
2.809215289030219e-15
```

Doing pretty well. Try to improve using shifts...

```
julia> Ak=Matrix(A)
       for k=1:100
           mu=1
           Q,R=qr(Ak-mu*I)
           global Ak=R*Q+mu*I
       end

julia> Ak
4×4 Matrix{Float64}:
  4.00688       -2.04307e-11   5.58876e-17  5.61784e-16
 -2.04269e-11  -1.3003         1.45298e-16  5.56745e-17
  5.75735e-36  -2.72329e-25   -0.31798      7.28655e-16
 -1.69046e-90   8.10216e-80    2.85601e-55  0.640657

julia> sort(diag(Ak))
4-element Vector{Float64}:
 -1.3002967911550614
 -0.31797993105343547
  0.6406566068270441
  4.006878725290595
```

choose shift ≈ 0.64...

At least it still works...

Trick how to choose shifts to speed convergence.

```
julia> Ak=Matrix(A)
       for k=1:100
           mu=0.64
           Q,R=qr(Ak-mu*I)
           global Ak=R*Q+mu*I
       end

julia> Ak
4×4 Matrix{Float64}:
 4.00688       1.32202e-15   8.75357e-16   -9.14195e-17
 1.01736e-23  -1.3003       -4.1562e-16     1.56173e-16
 9.85888e-55   9.36325e-32  -0.31798       -1.19921e-16
 0.0           0.0          -3.09436e-315   0.640657
```

That seemed to help..

smaller

But .64 is not known ahead of time... idea
iteratively improve the shift..

```
julia> Ak=Matrix(A)
       for k=1:100
           mu=Ak[4,4]
           Q,R=qr(Ak-mu*I)
           global Ak=R*Q+mu*I
       end

julia> Ak
4×4 Matrix{Float64}:
  4.00688       3.92591e-15   1.24709e-17  -5.82881e-16
 -2.1974e-23   -1.3003        1.29866e-16   9.63904e-16
  2.30397e-54  -1.01308e-31  -0.31798      -1.90108e-16
  0.0           0.0           0.0           0.640657
```

even better...

Note that the even better converges so fast that the bottom row will be all zeros after only about 12 iterations.

I forgot to show this in class.

```
julia> Ak=Matrix(A)
       for k=1:12
           mu=Ak[4,4]
           Q,R=qr(Ak-mu*I)
           global Ak=R*Q+mu*I
       end
```

even after only 12, the entire lower row is zeros

```
julia> Ak
4×4 Matrix{Float64}:
  4.00677     -0.0242769    2.32056e-6   -7.40061e-17
 -0.0242769   -1.30019     -9.23687e-5   -3.34035e-16
  2.32056e-6  -9.23687e-5  -0.31798       1.79152e-16
  0.0          0.0          0.0           0.640657
```

already small...