### Quicksort ([Quicksort pseudo code](http://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CFkQFjAE&url=http%3A%2F%2Fwww.cc.gatech.edu%2Fclasses%2Fcs3158_98_fall%2Fquicksort.html&ei=JSQ-UZvgNcbprAexnIGIBw&usg=AFQjCNENgVCzy3vR35w-cYGBouKdhgMi2g&sig2=etwXbJfsZ99CSceW-CJA3w&bvm=bv.43287494,d.bmk))

Quicksort(A,p,r) {

if (p < r) {

q <- Partition(A,p,r)

Quicksort(A,p,q)

Quicksort(A,q+1,r)

}

}

Partition(A,p,r)

x <- A[p]

i <- p-1

j <- r+1

while (True) {

repeat

j <- j-1

until (A[j] <= x)

repeat

i <- i+1

until (A[i] >= x)

if (i<-=""> A[j]

else

return(j)

}

}

Here is pseudo-Python-code for quicksort. It is based on the algorithm as it is presented in *Introduction to Algorithms* by Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, Cambridge, Mass.: MIT Press, 1990. (The code may look a lot like real Python code, but it isn't.)

The algorithm is implemented by the Quicksort class. After each chunk of code, I list the name of the Quicksort method that implements it. The mapping is a little rough, but I think you'll see the connection.

def Quicksort(A, p, r):

"""Sort array A in place.

Initial call should be Quicksort(A, 0, len(A))

"""

if p < r:

q = Partition(A, p, r)

Quicksort(A, p, q)

Quicksort(A, q+1, r)

Implemented by Quicksort.done

def Partition(A, p, r):

x = A[p] # the pivot element

i = p - 1

j = r + 1

Implemented by Quicksort.start

while 1:

j = j - 1

while A[j] <= x:

j = j - 1

Implemented by Quicksort.slide\_left

i = i + 1

while A[i] >= x:

j = j - 1

Implemented by Quicksort.slide\_right

if i < j: # elements being compared

swap(A[i], A[j])

else:

return j

Implemented by Quicksort.exchange\_or\_done