**Performance of Quicksort**

We will count the number $C(n)$ of comparisons performed by quicksort in sorting an array of size $n$.

We have seen that `partition()` performs $n$ comparisons (possibly $n-1$ or $n+1$, depending on the implementation).

In fact, $n-1$ is the lower bound on the number of comparisons that any partitioning algorithm can perform.

The reason is that every element other than the pivot must be compared to the pivot; otherwise we have no way of knowing whether it goes left or right of the pivot.

So our recurrence for $C(n)$ is:

$$C(n) = n + C(k-1) + C(n-k), \quad C(0) = C(1) = 0$$  
($k = \text{final position of pivot element}$)

**A bad case (actually the worst case):** At every step, `partition()` splits the array as unequally as possible ($k = 1$ or $k = n$).

Then our recurrence becomes

$$C(n) = n + C(n-1), \quad C(0) = C(1) = 0$$

This is easy to solve.

This is terrible. It is no better than simple quadratic time algorithms like straight insertion sort.

**A good case (actually the best case):** At every step, `partition()` splits the array as equally as possible ($k = (n+1)/2$; the left and right subarrays each have size $(n-1)/2$).

This is possible at every step only if $n = 2^k - 1$ for some $k$.

However, it is always possible to split nearly equally. The recurrence becomes

$$C(n) = n + 2C((n-1)/2), \quad C(0) = C(1) = 0,$$

which we approximate by

$$C(n) = n + 2C(n/2), \quad C(1) = 0$$

This is the same as the recurrence for mergesort, except that the right side has $n$ in place of $n-1$. The solution is essentially the same as for mergesort:

$$C(n) = n \lg(n).$$

This is excellent — essentially as good as mergesort, and essentially as good as any comparison sorting algorithm can be.
The expected case: Here we assume either (i) the array to be partitioned is randomly ordered, or (ii) the pivot element is selected from a random position in the array.

In either case, the pivot element will be a random element of the array to be partitioned. That is, for $k = 1, 2, ..., n$, the probability that the pivot element is the $k$th largest element of the array is $1/n$. (Recall that, if the pivot element is the $k$th largest element of the array, it ends up after partitioning in position $k$.)

In the recurrence

$$C(n) = n + C(k-1) + C(n-k), \quad C(0) = C(1) = 0,$$

all values of $k$ are equally likely. We must average over all $k$.

$$C(n) = (1/n) \sum_{k=1}^{n} (n + C(k-1) + C(n-k)), \quad C(0) = C(1) = 0,$$

$$= n + (1/n) \sum_{k=1}^{n} C(k-1) + (1/n) \sum_{k=1}^{n} C(n-k)$$

Note: $\sum_{k=1}^{n} C(k-1) = \sum_{i=0}^{n-1} C(i)$, by substituting $i = k-1$.

$\sum_{k=1}^{n} C(n-k) = \sum_{i=0}^{n-1} C(i)$, by substituting $i = n-k$.

So our recurrence becomes

$$C(n) = n + (2/n) \sum_{i=0}^{n-1} C(i), \quad \text{or}$$

$$nC(n) = n^2 + 2 \sum_{i=0}^{n-1} C(i)$$

Writing down the same recurrence with $n-1$ replacing $n$, we get

$$(n-1)C(n-1) = (n-1)^2 + 2 \sum_{i=0}^{n-2} C(i).$$

Subtracting this recurrence from the one above it gives

$$nC(n) - (n-1)C(n-1) = n^2 - (n-1)^2 + 2C(n-1), \quad \text{or}$$

$$nC(n) = (n+1)C(n-1) + 2n-1$$

Dividing by $(n+1)$ gives

$$C(n)/(n+1) = C(n-1)/n + (2n-1)/(n(n+1)).$$

To a very good approximation,

$$C(n)/(n+1) = C(n-1)/n + 2/n.$$

Now if let $D(n) = C(n)/(n+1)$, then the recurrence becomes

$$D(n) = D(n-1) + 2/n, \quad D(1) = 0.$$  

This is easy to solve:

$$D(n) = D(n-1) + 2/n$$

$$= D(n-2) + 2/(n-1) + 2/n$$

$$= D(n-3) + 2/(n-2) + 2/(n-1) + 2/n$$

$$= D(1) + 2/2 + 2/3 + ... + 2/(n-2) + 2/(n-1) + 2/n$$

$$= 2 \ln(n) - 2$$

$$\approx 2 \ln(n)$$

$$= 2 \ln(2) \lg(n)$$

$$\approx 1.39 \lg(n)$$

So $C(n) = (n+1)D(n) \approx 1.39 (n+1) \lg(n)$, or $C(n) \approx 1.39 n \lg(n)$

The expected case for quicksort is fairly close to the best case (only 39% more comparisons) and nothing like the worst case.

In most (not all) tests, quicksort turns out to be a bit faster than mergesort.

Quicksort performs 39% more comparisons than mergesort, but much less movement (copying) of array elements.
We saw that, in the expected case, quicksort performs one exchange for every six comparisons, or about $1.39 \frac{n \lg(n)}{6} \approx 0.23 n \lg(n)$ exchanges.

A slightly different partitioning algorithm performs one move (copy) for each three comparisons, or about $0.46 n \lg(n)$ moves.

By contrast, the version of mergesort given in class performs $2n \lg(n)$ moves, although this can be reduced to $n \lg(n)$ moves — still more than twice as many as quicksort is likely to perform.

With a randomized version of quicksort (pivot element chosen randomly), the standard deviation in the number of comparisons is also small.

The probability of performing substantially more than $1.39 n \lg(n)$ comparisons is extremely low.

Quicksort is not stable, since it exchanges nonadjacent elements.

If stability is not required, quicksort provides a very attractive alternative to mergesort.

Quicksort is likely to run a bit faster than mergesort — perhaps 1.2 to 1.4 times as fast.

Quicksort requires less memory than mergesort.

A good implementation of quicksort is probably easier to code than a good implementation of mergesort.