18. Sorting Algorithms :: Quick Sort :: Average Case Time Complexity

How efficient is quick sort?

It turns out that the worst case behavior of quick sort occurs when the list is already sorted, and in this case, the time complexity is $O(n^2)$ which is no better than insertion or selection sort.

However, analyzing an algorithm to determine the worst case time complexity only tells us that the time for the algorithm will never exceed some upper bound, i.e., since quick sort has worst case time complexity of $O(n^2)$ we know it will never take more than $C \cdot n^2$ list accesses to sort a list.

But algorithms do not always exhibit their worst case behavior, so another important measure of time complexity is the **average case** time complexity.

To rigorously prove the average case time complexity of quick sort—and in general, for most algorithms—is not as easy as proving the worst case time complexity (which is why we often start by proving the worst case complexity). Therefore, rather than trying to rigorously prove the average case time complexity of quick sort, let's make some simplifying assumptions and look at an informal proof.

If the size of the list is $n = 2^p$ and the elements of the list are truly in random order, then on average we would expect the partitioning operation to partition the list into two sublists each of size n / 2. Of course, this will not happen every time. Sometimes the ratio of the sublist sizes may be 25% to 75%, other times 33% to 67%, other times 10% to 90%, but on average, we would expect a 50% to 50% size ratio after partitioning.

If that is the case, then the number of recursive calls will be proportional to $lg\ n$ because during each recursive call the list size will be reduced by one-half.

18. Sorting Algorithms :: Quick Sort :: Average Case Time Complexity (continued)

Our key operations we wish to count—as with merge sort—are the total number of list accesses that will occur during quick sort.

Since all list accesses are performed in the partitioning procedure, the total number of list accesses made during quick sort will depend on the number of list accesses that the partitioning procedure makes on each randomly-ordered sublist.

Let's define $p(n)$ as the number of list accesses that the partitioning procedure makes on a list of size n. Then we have:

Level	**Number of list accesses**
0	$p(n)$
1	$p(n/2) + p(n/2) = 2p(n/2)$
2	$p(n/4) + p(n/4) = 2p(n/2^2)$
3	$p(n/8) + p(n/8) = 2p(n/2^3)$
...	
$lg\ n$	$2p(n/2^{lg\ n}) = 2p(1) = 0$

Consequently, the total number of list accesses will be:

$$a(n) \approx 2\sum_{i=0}^{lg\ n} p\left(\frac{n}{2^i}\right)$$

18. Sorting Algorithms :: Quick Sort :: Average Case Time Complexity (continued)

Therefore, we need to determine what $p(n)$ is for a randomly-ordered list.

This is not very easy to prove, so in the interest of time, I will just tell you that it is no more than Cn for some positive constant C—which means that *partition*() is $O(n)$ Thus:

Level	**Number of list accesses**
0	No more than Cn
1	No more than Cn
2	No more than Cn
3	No more than Cn
...	
$lg\ n$	No more than Cn

Consequently:

$$a(n) \approx Cn + Cn + \cdots + Cn$$

where there are roughly $lg\ n$ additions, or in other words:

$$a(n) = (lg\ n) \cdot Cn = Cn\ lg\ n$$

which is O($n\ lg\ n$). Thus, the average case time complexity of quick sort is $O(n\ lg\ n)$.