8.8.3 Analysis of Quicksort

It is now time to examine the quicksort algorithm carefully, to determine when it works well, when it does not, and how much computation it performs.

276

✓ 1. Choice of Pivot

Our choice of a key at the center of the list to be the pivot is arbitrary. This choice may succeed in dividing the list nicely in half, or we may be unlucky and find that one sublist is much larger than the other. Some other methods for choosing the pivot are considered in the exercises. An extreme case for our method occurs for the following list, where every one of the pivots selected turns out to be the largest key in its sublist:

worst case

$$2 \quad 4 \quad 6 \quad 7 \quad 3 \quad 1 \quad 5$$

Check it out, using the partition function in the text. When quicksort is applied to this list, its label will appear to be quite a misnomer, since at the first recursion the nonempty sublist will have length 6, at the second 5, and so on.

If we were to choose the pivot as the first key or the last key in each sublist, then the extreme case would occur when the keys are in their natural order or in their reverse order. These orders are more likely to happen than some random order, and therefore choosing the first or last key as pivot is likely to cause problems.

✓ 2. Count of Comparisons and Swaps

Let us determine the number of comparisons and swaps that contiguous quicksort makes. Let $C(n)$ be the number of comparisons of keys made by quicksort when applied to a list of length n, and let $S(n)$ be the number of swaps of entries. We have $C(1) = C(0) = 0$. The partition function compares the pivot with every other key in the list exactly once, and thus the function partition accounts for exactly $n - 1$ key comparisons. If one of the two sublists it creates has length r, then the other sublist will have length exactly $n - r - 1$. The number of comparisons done in the two recursive calls will then be $C(r)$ and $C(n - r - 1)$. Thus we have

total number of comparisons

$$C(n) = n - 1 + C(r) + C(n - r - 1).$$

To solve this equation we need to know r. In fact, our notation is somewhat deceptive, since the values of $C(\ )$ depend not only on the length of the list but also on the exact ordering of the entries in it. Thus we shall obtain different answers in different cases, depending on the ordering.

✓ 3. Comparison Count, Worst Case

First, consider the worst case for comparisons. We have already seen that this occurs when the pivot fails to split the list at all, so that one sublist has $n-1$ entries and the other is empty. In this case, since $C(0) = 0$, we obtain $C(n) = n - 1 + C(n - 1)$. An expression of this form is called a ***recurrence relation*** because it expresses its answer in terms of earlier cases of the same result. We wish to solve the recurrence, which means to find an equation for $C(n)$ that does not involve $C(\ )$ on the other side.

recurrence relation

Various (sometimes difficult) methods are needed to solve recurrence relations, but in this case we can do it easily by starting at the bottom instead of the top:

$$\begin{aligned}
C(1) &= 0. \\
C(2) &= 1 + C(1) &&= 1. \\
C(3) &= 2 + C(2) &&= 2 + 1. \\
C(4) &= 3 + C(3) &&= 3 + 2 + 1. \\
&\vdots &&\vdots \\
C(n) &= n - 1 + C(n-1) &&= (n-1)+(n-2)+\cdots+2+1 \\
& &&= \tfrac{1}{2}(n-1)n = \tfrac{1}{2}n^2 - \tfrac{1}{2}n.
\end{aligned}$$

In this calculation we have applied Theorem A.1 on page 647 to obtain the sum of the integers from 1 to $n-1$.

selection sort

Recall that selection sort makes about $\frac{1}{2}n^2 - \frac{1}{2}n$ key comparisons, and making too many comparisons was the weak point of selection sort (as compared with insertion sort). Hence in its worst case, quicksort is as bad as the worst case of selection sort.

4. Swap Count, Worst Case

Next let us determine how many times quicksort will swap entries, again in its worst case. The partition function does one swap inside its loop for each key less than the pivot and two swaps outside its loop. In its worst case, the pivot is the largest key in the list, so the partition function will then make $n+1$ swaps. With $S(n)$ the total number of swaps on a list of length n, we then have the recurrence

$$S(n) = n + 1 + S(n-1)$$

in the worst case. The partition function is called only when $n \ge 2$, and $S(2) = 3$ in the worst case. Hence, as in counting comparisons, we can solve the recurrence by working downward, and we obtain

answer

$$S(n) = (n+1) + n + \cdots + 3 = \tfrac{1}{2}(n+1)(n+2) - 3 = 0.5n^2 + 1.5n - 1$$

swaps in the worst case.

5. Comparison with Insertion Sort and Selection Sort

In its worst case, contiguous insertion sort must make about twice as many comparisons and assignments of entries as it does in its average case, giving a total of $0.5n^2 + O(n)$ for each operation. Each swap in quicksort requires three assignments of entries, so quicksort in its worst case does $1.5n^2 + O(n)$ assignments, or, for large n, about three times as many as insertion sort. But moving entries was the weak point of insertion sort in comparison to selection sort. Hence, in its worst case, quicksort (so-called) is worse than the poor aspect of insertion sort, and, in regard to key comparisons, it is also as bad as the poor aspect of selection sort. Indeed, in the worst-case analysis, quicksort is a disaster, and its name is nothing less than false advertising.

poor worst-case behavior

excellent average-case behavior

It must be for some other reason that quicksort was not long ago consigned to the scrap heap of programs that never worked. The reason is the *average* behavior of quicksort when applied to lists in random order, which turns out to be one of the best of any sorting methods (using key comparisons and applied to contiguous lists) yet known!

8.8.4 Average-Case Analysis of Quicksort

To do the average-case analysis, we shall assume that all possible orderings of the list are equally likely, and for simplicity, we take the keys to be just the integers from 1 to n.

277

✓ 1. Counting Swaps

When we select the pivot in the function partition, it is equally likely to be any one of the keys. Denote by p whatever key is selected as pivot. Then after the partition, key p is guaranteed to be in position p, since the keys $1, \dots, p-1$ are all to its left and $p+1, \dots, n$ are to its right.

The number of swaps that will have been made in one call to partition is $p+1$, consisting of one swap in the loop for each of the $p-1$ keys less than p and two swaps outside the loop. Let us denote by $S(n)$ the average number of swaps done by quicksort on a list of length n and by $S(n,p)$ the average number of swaps on a list of length n where the pivot for the first partition is p. We have now shown that, for $n \ge 2$,

$$S(n,p) = (p+1) + S(p-1) + S(n-p).$$

We must now take the average of these expressions, since p is random, by adding them from $p=1$ to $p=n$ and dividing by n. The calculation uses the formula for the sum of the integers (Theorem A.1), and the result is

$$S(n) = \frac{n}{2} + \frac{3}{2} + \frac{2}{n}\Big(S(0) + S(1) + \cdots + S(n-1)\Big).$$

2. Solving the Recurrence Relation

The first step toward solving this recurrence relation is to note that, if we were sorting a list of length $n-1$, we would obtain the same expression with n replaced by $n-1$, provided that $n \ge 2$:

$$S(n-1) = \frac{n-1}{2} + \frac{3}{2} + \frac{2}{n-1}\Big(S(0) + S(1) + \cdots + S(n-2)\Big).$$

Multiplying the first expression by n, the second by $n-1$, and subtracting, we obtain

$$nS(n) - (n-1)S(n-1) = n + 1 + 2S(n-1),$$

or

$$\frac{S(n)}{n+1} = \frac{S(n-1)}{n} + \frac{1}{n}.$$

We can solve this recurrence relation as we did a previous one by starting at the bottom. The result is

$$\frac{S(n)}{n+1} = \frac{S(2)}{3} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

The sum of the reciprocals of integers is studied in Section A.2.8, where it is shown that

$$1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n + O(1).$$

The difference between this sum and the one we want is bounded by a constant, so we obtain

$$S(n)/(n+1) = \ln n + O(1),$$

or, finally,

$$S(n) = n \ln n + O(n).$$

To compare this result with those for other sorting methods, we note that

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

and $\ln 2 \approx 0.69$, so that

$$S(n) \approx 0.69(n \lg n) + O(n).$$

3. Counting Comparisons

Since a call to the partition function for a list of length n makes exactly $n-1$ comparisons, the recurrence relation for the number of comparisons made in the average case will differ from that for swaps in only one way: Instead of $p+1$ swaps in the partition function, there are $n-1$ comparisons. Hence the first recurrence for the number $C(n,p)$ of comparisons for a list of length n with pivot p is

$$C(n,p) = n - 1 + C(p-1) + C(n-p).$$

When we average these expressions for $p = 1$ to $p = n$, we obtain

$$C(n) = n + \frac{2}{n}\Big(C(0) + C(1) + \cdots + C(n-1)\Big).$$

276

Since this recurrence for the number $C(n)$ of key comparisons differs from that for $S(n)$ only by the factor of $\frac{1}{2}$ in the latter, the same steps used to solve for $S(n)$ will yield

$$C(n) \approx 2n \ln n + O(n) \approx 1.39 n \lg n + O(n).$$