Lecture Notes, Fundamental Algorithms: Quicksort

The quicksort algorithm is based on a Partition procedure. Partition(A, i, j, k), given as input an array $A[i:k]$ reorders it about a pivot, typically the item initially in location i, i.e. item $e = A[i]$ in the input, so that after the partition:

- $A[j] = e$
- $A[h] \leq A[j]$, for $i \leq h < j$
- $A[\ell] > A[j]$, for $j \leq \ell \leq k$

Sometimes, the partitioning is defined to create three sets of items, the middle set comprising all the items with value e.

Given the Partition procedure, it is easy to give an algorithm Quicksort(A, i, k) which sorts an array $A[i:k]$. If there are one or zero items in A, i.e. $i \geq k$, there is nothing to do; otherwise:

- First partition, i.e. run Partition(A, i, j, k)
- Then recursively sort $A[i:j-1]$ and $A[j+1:k]$

Partition is readily implemented to run in $O(n)$ time in-place, i.e. with only $O(1)$ additional variables, and in particular with no additional array (in contrast to merge sort), though some care is needed to get it right.

A little terminology: $A[j]$ is called the pivot item.

Now we focus on quicksort's running time. If we are unlucky in our choice of pivot, we could end up with $j = i$ or $j = k$, and we then have a recursive problem of size $n-1$ to sort. If this repeats, the total number of instructions executed (the work) is $\Theta(n + (n-1) + (n-2) + \ldots + 1) = \Theta(\frac{1}{2}n(n+1)) = \Theta(n^2)$, which is not very efficient for sorting.

However, the average running time is much better, assuming all the input items are distinct and all input orderings are equally likely. Then, with probability $1/n$, the pivot item is going to be the i-th item in the sorted order. In this case, the recursive problems have sizes $i-1$ and $n-i$. This leads to the following recurrence for the running time, where c' is a suitable constant.

$$
\begin{aligned}
T(n) &\leq c'n + \frac{1}{n}\sum_{i=1}^{n}[T(i-1) + T(n-i)] \quad n > 1 \\
T(1) &\leq c' \\
T(0) &\leq c'
\end{aligned}
$$

Notice that $\sum_{i=1}^{n} T(i-1) = T(0) + \sum_{j=1}^{n-1} T(j) \leq c' + \sum_{i=1}^{n-1} T(i)$; also, on setting $j = n-i$ we obtain $\sum_{i=1}^{n} T(n-i) = \sum_{j=n-1}^{0} T(j) = \sum_{j=0}^{n-1} T(j) = c' + \sum_{i=1}^{n-1} T(i)$. Thus, setting $c \geq c' + 2c'/n$ ($c = 2c'$ suffices as $n \geq 2$ in this case), we can rewrite this recurrence equation as:

$$
\begin{aligned}
T(n) &\leq cn + \frac{2}{n}\sum_{i=1}^{n-1} T(i) \quad n > 1 \\
T(1) &\leq c \\
T(0) &\leq c
\end{aligned}
$$

Rather than solve this recurrence equation directly (which can be done) we are going to obtain an upper bound by a simple argument.

Incidentally, the recurrence equation for the expected number of comparisons, $C(n)$, performed by quicksort is as follows:

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

To the reader: justify this.

The analysis

There are many, many different recursion trees that arise, one for each possible sequence of pivot choices. Each recursion tree has its own runtime. We seek to average the runtime over all these recursion trees, weighting each tree by the probability it occurs. This is then the average runtime. Fortunately, we do not need to explore the different recursion trees in detail.

For each recursion tree, we group the recursive calls into phases, for $i = 0, 1, 2, \ldots$. Phase i will include all subproblems with sizes m satisfying $\left(\frac{3}{4}\right)^{i+1} n < m \leq \left(\frac{3}{4}\right)^{i} n$. So the initial call will be in Phase 0.

Depending on the split produced by the initial partition, one of the two resulting recursive subproblems may also be in Phase 0. It is helpful to define the good and bad cases: in the *good* case, the two recursive subproblems for the initial call each have size at most $\frac{3}{4}n$; otherwise we are in the bad case with one problem having size more than $\frac{3}{4}n$ (and the other problem having size less than $\frac{1}{4}n < \frac{3}{4}n$). More generally, if a recursive call is in Phase i, we are in the good case if both its children have size at most $\left(\frac{3}{4}\right)^{i+1} n$, and in the bad case otherwise, with one problem having size more than $\left(\frac{3}{4}\right)^{i+1} n$ (and the other problem having size less than $\frac{1}{4}\left(\frac{3}{4}\right)^{i} n < \left(\frac{3}{4}\right)^{i+1} n$).

Thus in the bad case, one of the two resulting subproblems will be in Phase j for some $j > i$, for this subproblem will have size less than $\left(\frac{3}{4}\right)^{i+1} n$. So in each recursion tree, starting with a subproblem in Phase i, there will be a path of Phase i subproblems, with all off-path subproblems being in later phases. Depending on the random choices of the pivots, this path could be of length 1, or it could be longer.

Now we focus on a particular subproblem of size m and the recursion trees in which it occurs. Suppose this problem is in Phase i. We wish to determine how much work is performed in Phase i in the portion of the recursion tree rooted at this problem of size m. We will average over all the recursion trees in which this subproblem occurs.

When $m = n$, we are in the bad case if the pivot lies in the first or last $\lceil n/4 \rceil - 1$ items. So there are at most $2(\lceil n/4 \rceil - 1) \leq 2\lfloor n/4 \rfloor \leq n/2$ bad choices of the pivot, and hence the probability of the good case is at least $\frac{1}{2}$, and the probability of the bad case is at most $\frac{1}{2}$. More generally, in Phase 0, we are in the bad case if the pivot lies in the first or last $\lceil n/4 \rceil - 1$ items in the whole n item input. This leaves at least $n/2$ good pivot choices in the subproblem, and hence the probability of the good case is at least $(\frac{1}{2}n)/m \geq \frac{1}{2}$.

We bound the average non-recursive work on this path as follows. Each partition in Phase 0 performs at most cn operations for some constant c. The first partition must happen. A second partition happens with probability at most $\frac{1}{2}$. A third partition happens with probability at most $1/2^2$. A jth partition happens with probability at most $1/2^{j-1}$. So the total work is at most $cn(1 + 1/2 + 1/2^2 + \ldots) = 2cn$ operations.

Similarly, in Phase i, averaging the cost of the work for performing the partitioning over the paths of Phase i subproblems starting from a given subproblem of size m will yield an average cost of $2cm$ operations for the partitioning.

The key observation is that in any one recursion tree the following subproblems are disjoint: the topmost problems in Phase i (problems whose parents are in an earlier phase). Therefore these subproblems have total size at most n. This means that the subproblems on the paths starting at these subproblems perform at most $2cn$ operations on the average. (To see this, suppose these subproblems have sizes $n_1, n_2, \ldots, n_k$ for some k; then on the paths descending from these subproblems, averaging over all the recursion trees they induce, the partitioning work they perform is at most $2c(n_1 + n_2 + \ldots + n_k) \leq 2cn$ as $n_1 + n_2 + \ldots + n_k \leq n$.)

If we imagine unfolding the recursion trees down to the initial problems in Phase i, we have just shown that for each such recursion tree, the average cost of expanding down to the initial problems in Phase $i+1$ is at most $2cn$. Therefore, over all the recursion trees, the average cost of partitioning in Phase i is at most $2cn$.

Finally, the leaf level problems, problems of size 1, will each require at most c work, for cn work in total.

How many phases are there in the recursion tree? Remember the successive size bounds on the problem sizes are $n, (3/4)n, (3/4)^2 n, \ldots, (3/4)^i n, \ldots 1$. Thus there are at most $1 + \log_{4/3} n$ phases. We conclude that the overall work is at most $cn(1 + \log_{4/3} n) = O(n \log n)$.

In practice, Quicksort is highly efficient, and is the preferred algorithm for sorting data that fits in main memory.