CIS 575. Introduction to Algorithm Analysis
Material for February 23, 2024

Quicksort: an Analysis

The topic of this note is covered in *Cormen*'s Section 7.2.

1 Quicksort: Pivot Selection and Time Analysis

It should be obvious that the performance of QuickSort will depend on which pivot is chosen.

Best Case is when we pick a pivot close to the median. If we could ensure that this is always the case, the running time would be described (since Dutch National Flag runs in time $\Theta(n)$) by the recurrence

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

which has solution $T(n) \in \Theta(n \lg(n))$.

Worst Case is when we pick the smallest key or the largest key as the pivot. If we are so unlucky that this is always the case, the running time would be described by the recurrence

$$T(n) = T(n-1) + \Theta(n)$$

and thus

$$T(n) = \sum_{i=1}^{n} \Theta(i) \in \Theta(n^2).$$

Almost Worst Case is when we pick a pivot such that around 90 % of all records have smaller keys (or 90 % have larger keys). In that case, we get the recurrence

$$T(n) = T(\frac{9}{10}n) + T(\frac{1}{10}n) + \Theta(n)$$

and it is possible (you may try!) to use the substitution method to prove that then

$$T(n) \leq cn \lg(n) \text{ for } n \geq 2$$

will hold for some c. Thus even in this case, which appears to be quite different from the "best case", we have $T(n) \in \Theta(n \lg(n))$ (though the constant c will be greater than for the "best case").

"Average" Case Assume that the pivot is chosen *randomly*. Then one can prove by probabilistic analysis (as we do in CIS775, the graduate version of this course) that the *expected* running time is in $\Theta(n \lg(n))$.

1.1 Space Use

Since the Dutch National Flag algorithm is in-place (uses space that is "substantially less" than what the input occupies), it may be tempting to believe that also QuickSort is in-place. But one must keep in mind that its recursion is not tail-recursive: when doing a recursive call to sort the first of the two partitions, the stack must contain the information needed to later sort the second partition. That is, the stack must contain information about all the recursive calls that have been postponed, and there could be almost n such calls. However, if we always process the *shortest* partition first there will be at most $\lg(n)$ such calls; in that case, the space use is in $O(\lg n)$ which is usually considered "in-place".

1.2 Concluding Remarks

We have seen that the Divide & Conquer paradigm can be used to derive a sorting algorithm QuickSort that unless we are very unlucky runs in time $\Theta(n \lg(n))$. Recall that Merge Sort *always* runs in time $\Theta(n \lg(n))$, as does Heapsort which we shall encounter later. But QuickSort merits its name in that if it is implemented efficiently, it is typically significantly faster than those algorithms.

One may modify QuickSort, and also Merge Sort, as follows: when the number of elements to be sorted is below a certain threshold, use Insertion Sort to sort them. This will *not* affect the asympotic complexity (no risk of inheriting the quadratic running time of Insertion Sort), but *may* result in a faster algorithm.