Illinois Institute of Technology
Department of Computer Science

Lecture 4: January 25, 2016

CS 430 Introduction to Algorithms
Spring Semester, 2016

Quicksort

Quicksort, given in greater detail in section 7.1 of CLRS3, is among the most commonly used algorithms for sorting internally. Its analysis is also of particular interest because its performance on average is asymptotically faster than its worst-case performance.

Although there are many variants, the basic form of the algorithm selects the first element of the array and partitions the array around the value of that element: that is, it divides the array into two sections, one consisting of all elements smaller than the first element and the second consisting of all elements larger than the first element. It then sorts each partition recursively.

To analyze the quicksort algorithm, we must ask how it behaves in the best case, in the average case, and in the worst case. The best case occurs when all of the partitions are perfectly balanced—that is, when the array is partitioned evenly in each recursive call. The worst case occurs when the input array is already sorted, as in that case the partitions are all unbalanced.

What are we counting in our analyses? In most sorting algorithms, including quicksort, the running time is proportional to the number of comparisons of pairs of elements. To simplify matters, then, we only count comparisons of pairs of elements.

In general, what is the form of the recurrence we will be using? In an array of size n, partitioning can be accomplished in $n-1$ comparisons. If partitioning yields one subarray of size k and one subarray of size $n-1-k$, the general recurrence is of the form

$$c(n) = n - 1 + c(k) + c(n-1-k)$$

Best-case analysis

In the best case, the partitions are balanced at all levels of the recursion. Thus, in the general form above, $k = \lceil \frac{n-1}{2} \rceil$. This yields the recurrence

$$c(n) = n - 1 + c\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + c\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right)$$

It turns out that this recurrence is quite difficult to solve—in fact, the solution depends on the binary representation of n. Fortunately, it is quite similar to the following much simpler recurrence, which can be solved by the master method:

$$\begin{aligned} c(n) &= n - 1 + 2c\left(\frac{n}{2}\right) \\ &= \Theta(n \lg n) \end{aligned}$$

Before we continue, how do we know that balanced partitioning actually yields the best case? The best case

occurs when the value of the recurrence is minimized:

$$c(n) = n - 1 + \min_{0 \le k \le n-1} (c(k) + c(n-1-k))$$

Techniques of calculus tell us that, indeed, this function is minimized when $k = n - 1 - k = \frac{n-1}{2}$.

Worst-case analysis

We can use a similar technique to analyze the worst case. Now we want to solve

$$c(n) = n - 1 + \max_{0 \le k \le n-1} (c(k) + c(n-1-k))$$

We first guess that $c(n) \le \frac{n^2}{2}$. Unfortunately, this approximation is not accurate enough to complete the calculation. If we tighten our approximation to $c(n) \le \frac{n^2}{2} + \frac{3n}{2}$, the computation succeeds:

$$\begin{aligned} c(n) &\le n - 1 + \max_{0 \le k \le n-1} (c(k) + c(n-1-k)) \\ &\le n - 1 + \max_{0 \le k \le n-1} \left(\frac{k^2}{2} + \frac{3k}{2} + \frac{(n-1-k)^2}{2} + \frac{3(n-1-k)}{2} \right) \end{aligned}$$

The function is maximized when $k = 0$ or $k = n - 1$, so

$$\begin{aligned} c(n) &\le n - 1 + \frac{(n-1)^2}{2} + \frac{3(n-1)}{2} \\ &\le \frac{n^2}{2} + \frac{3n}{2} \end{aligned}$$

Average-case analysis

The analysis given in the section 7.4.2 of CLRS3 is based on the clever observation that over the course of the algorithm, the probability that element i is ever compared to element j, $j \ge i$, is $2/(j - i + 1)$. Summing this probability for all values $1 \le i < j \le n$ gives an expected number of comparisons of $2n \ln n$. But this analysis is difficult to extend to variants of quicksort such as the "median-of-three partition" (exercise 7-5 on pages 188-189 of CLRS3); it also conceals some of the intuition of what is happening, so we give the alternative (actually, the classical!) analysis outlined in exercise 7-3 on page 187 of CLRS3.

In the average case all possible partitionings of the array are equally likely at each level of recursion. To force this behavior, we modify the basic algorithm to select an element at random, rather than the initial array element, and partition around it.

The recurrence now is

$$\begin{aligned} c(n) &= n - 1 + \sum_{k=0}^{n-1} (c(k) + c(n-1-k)) \cdot \mathbf{Pr}\{k \text{ elements are less than the partition element}\} \\ &= n - 1 + \sum_{k=0}^{n-1} \frac{1}{n} (c(k) + c(n-1-k)) \\ &= n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} c(k) \end{aligned}$$

Given the initial values $t_0, t_1, \ldots, t_{n_0}$, we can rewrite this as

$$t_n = an + b + \frac{2}{n}\sum_{i=0}^{n-1} t_i, \text{ where } n > n_0$$

Then multiplying both sides by n to clear the fractions:

$$nt_n = an^2 + bn + 2\sum_{i=0}^{n-1} t_i \tag{1}$$

This equation also holds with $n-1$ substituted for n:

$$(n-1)t_{n-1} = a(n-1)^2 + b(n-1) + 2\sum_{i=0}^{n-2} t_i \tag{2}$$

Subtracting (2) from (1) gives[1]

$$nt_n - (n-1)t_{n-1} = a(n^2 - (n-1)^2) + b(n - (n-1)) + 2t_{n-1}$$

We have eliminated the summation. Now, bring the $2t_{n-1}$ to the left hand side and simplify. We get

$$nt_n - (n+1)t_{n-1} = (2n-1)a + b$$

Dividing by $n(n+1)$ and using the method partial fractions,

$$\begin{aligned} \frac{t_n}{n+1} - \frac{t_{n-1}}{n} &= \frac{(2n-1)a+b}{n(n+1)} \\ &= \frac{3a-b}{n+1} + \frac{b-a}{n} \end{aligned}$$

Summing,

$$\sum_{i=n_0+1}^{n}\left(\frac{t_i}{i+1} - \frac{t_{i-1}}{i}\right) = \sum_{i=n_0+1}^{n}\left(\frac{3a-b}{i+1} + \frac{b-a}{i}\right)$$

The left-hand side is the telescoping sum

$$\frac{t_n}{n+1} - \frac{t_{n-1}}{n} + \frac{t_{n-1}}{n} - \frac{t_{n-2}}{n-1} + \cdots - \frac{t_{n_0+1}}{n_0+2} + \frac{t_{n_0+1}}{n_0+2} - \frac{t_{n_0}}{n_0+1} = \frac{t_n}{n+1} - \frac{t_{n_0}}{n_0+1}$$

Combining terms on the right-hand side gives

$$\frac{t_n}{n+1} - \frac{t_{n_0}}{n_0+1} = 2aH_n - 2aH_{n_0} - (3a-b)\left(\frac{1}{n_0+1} - \frac{1}{n+1}\right)$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the nth harmonic number. This gives us

$$t_n = 2anH_n + n\left(\frac{t_{n_0} - 3a + b}{n_0+1} - 2aH_{n_0}\right) + 2aH_n + \frac{t_{n_0} - 3a + b}{n_0+1} + 3a - b - 2aH_{n_0}$$

[1] D. Greene and D. Knuth, *Mathematics for the Analysis of Algorithms*, third edition, section 2.1.2 elaborates on this analysis technique.

where $t_{n_0} = an_0 + b + \frac{2}{n_0}\sum_{i=0}^{n_0-1} t_i$. Since $H_n = \ln n + O(1)$,

$$\begin{aligned} t_n &= 2an \ln n + O(n) \\ &= (2a \ln 2) n \lg n + O(n) \end{aligned}$$

Reverting to our recurrence of interest, c,

$$\begin{aligned} c(n) &= (2 \ln 2) n \lg n + O(n) \\ &\approx 1.38 n \lg n \end{aligned}$$

Since the best case involved $n \lg n$ comparisons, the average case requires approximately 38 percent more comparisons than the best case. Still, both the average case and the best case require $\Theta(n \lg n)$ comparisons, asymptotically fewer than the worst case.

Improving the average case

We can improve on the above by modifying the probability distribution. Rather than selecting one element at random, we select three. We then partition the array on the *median* of the three chosen elements.[2]

As above,

$$c(n) = n - 1 + \sum_{k=0}^{n-1} (c(k) + c(n-1-k)) \cdot \mathbf{Pr}\{k \text{ elements are less than the partition element}\}$$

There are $\binom{n}{3}$ ways to choose the three elements. If k elements are smaller than the median of the three chosen elements, the smallest of the chosen elements must be among those k and the largest must be among the remaining $n-1-k$. Thus the probability that k elements are less than the partition element is $\frac{k(n-1-k)}{\binom{n}{3}}$, leading to the recurrence

$$c(n) = n - 1 + \sum_{k=0}^{n-1} \frac{k(n-1-k)}{\binom{n}{3}} (c(k) + c(n-1-k))$$

It turns out that solving this recurrence yields $c(n) \approx \frac{12}{7} n H_n \approx 1.19 n \lg n$, or only 19 percent slower than the best case.

We could carry this approach further and select five, seven, or more elements in the array, find the median, and proceed from there. This is not as good an idea as it may sound, however, because before long the additional overhead in each recursive step to select the elements and find their median dominates the savings generated by the improved probability distribution.

[2] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, section 3.7 discusses this algorithm in greater detail.