A Simple Expected Running Time Analysis for Randomized "Divide and Conquer" Algorithms

Brian C. Dean
M.I.T., Cambridge, MA 02139, USA

Abstract. There are many randomized "divide and conquer" algorithms, such as randomized Quicksort, whose operation involves partitioning a problem of size n uniformly at random into two subproblems of size k and $n-k$ that are solved recursively. We present a simple combinatorial method for analyzing the expected running time of such algorithms, and prove that under very weak assumptions this expected running time will be asymptotically equivalent to the running time obtained when problems are always split evenly into two subproblems of size $n/2$.

Keywords: quicksort, divide and conquer, probabilistic recurrence

In this paper we study the expected running time analysis of randomized "divide and conquer" algorithms that divide a problem of size n uniformly at random into two subproblems of size k and $n-k$, recursively solve these subproblems, and then somehow combine their solutions to obtain a solution to the original problem. A prototypical example is randomized Quicksort [3]. Letting the random variable T_n denote the running time for a problem of size n, we can compute $\mathbf{E}[T_n]$ by solving the recurrence

$$\begin{aligned} \mathbf{E}[T_n] &= f(n) + \frac{1}{n-1}\sum_{k=1}^{n-1}(\mathbf{E}[T_k] + \mathbf{E}[T_{n-k}]) \\ &= f(n) + \frac{2}{n-1}\sum_{k=1}^{n-1}\mathbf{E}[T_k], \end{aligned} \tag{1}$$

where $f(n)$ is a nondecreasing function giving the running time spent in the partitioning and recombining steps of the algorithm, and where $T_1 = f(1) = \Theta(1)$ as a base case. Upon encountering this complicated recurrence for the first time (usually when studying about randomized Quicksort), students are often taught to "guess" a solution for $\mathbf{E}[T_n]$ by assuming the best case where all subproblems are split evenly in half. This gives a much simpler recurrence,

$$S_n = \begin{cases} f(n) + 2S_{n/2} & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1, \end{cases} \tag{2}$$

whose solution can usually be obtained on inspection, for example by using the "Master Method" from the prominent algorithms textbook of Cormen et al. [1]. We prove that the answer obtained by this technique is actually much more than just a reasonable guess:

Theorem 1. *As long as $f(n)$ changes by at most a constant factor when n is multiplied by a constant factor (we call this the polynomial growth condition), then $\mathbf{E}[T_n] = \Theta(S_n)$.*

To prove the theorem, we develop a simple combinatorial method for solving (1) that is interesting in its own right. Slightly simpler approaches are known for solving the special case of randomized Quicksort where $f(n) = \Theta(n)$ [1, Section 7.4]. However, our technique works with an arbitrary $f(n)$ term and is much less complicated than other more powerful techniques for obtaining bounds on general probabilistic recurrences (see, for example [4]).

The Method. Let x_k be a random variable whose value gives the total number of subproblems of size k generated over the entire course of the algorithm. We will shortly prove that

$$\mathbf{E}[x_k] = \begin{cases} 1 & \text{if } k = n \\ \frac{2n}{k(k+1)} & \text{if } 1 \leq k < n. \end{cases} \tag{3}$$

In terms of the random variables $x_1 \ldots x_n$, we can now express (1) in the following equivalent form:

$$\begin{aligned} \mathbf{E}[T_n] &= \sum_{k=1}^{n} \mathbf{E}[x_k] f(k) \\ &= f(n) + 2n \sum_{k=1}^{n-1} \frac{f(k)}{k(k+1)}. \end{aligned} \tag{4}$$

This transforms the problem of computing $\mathbf{E}[T_n]$ into that of simply summing up an appropriate series. For example, if $f(k) = \Theta(k)$ as with randomized Quicksort, we immediately find that $\mathbf{E}[T_n] = \Theta(n \log n)$. If $f(k)$ happens to a random variable (e.g. if the partitioning or recombining steps of our algorithm are somehow randomized), it suffices to replace $f(k)$ with its expectation when evaluating (4), since $f(k)$ and x_k will be independent: x_k depends only on random choices made when solving subproblems of size larger than k, and $f(k)$ depends on random choices made when solving subproblems of size k.

Theorem 2. $\mathbf{E}[x_k] = \frac{2n}{k(k+1)}$ *for* $k < n$.

Proof. It is easy to see that $\mathbf{E}[x_n] = 1$ and $\mathbf{E}[x_{n-1}] = 2/(n-1)$. For $k < n-1$, consider all of the subproblems of size k that arise during the execution of our algorithm, all of which are created by randomly splitting up subproblems of size larger than k. Write x_k as the sum of two random variables y_k and z_k whose values respectively give the number of subproblems of size k created by splitting subproblems of size $k+1$, and of size strictly larger than $k+1$. We know that $\mathbf{E}[y_k] = \frac{2}{k}\mathbf{E}[x_{k+1}]$, since precisely two of the k possible splitting points in a problem of size $k+1$ will result in a problem of size k (the base case $k = 1$ is an exception, but we still have $\mathbf{E}[y_1] = 2\mathbf{E}[x_2]$). We argue that $\mathbf{E}[z_k] = \mathbf{E}[x_{k+1}]$ as follows: consider any sequence of successive subproblem splits, starting from the initial problem of size n, that results in a subproblem of size $k+1$. There is an analogous sequence of splits that occurs with the same probability and that results in a subproblem of size k — this follows from the fact that on the last split of the sequence, it is equally likely that we obtain a subproblem of size $k+1$ as it is that we obtain a subproblem of size k. In fact, we claim there is a bijection between equal-probability "split sequences" yielding subproblems of size $k+1$, and those yielding subproblems of size k that do not include a subproblem of size $k+1$ as an intermediate step. We therefore have

$$\mathbf{E}[x_k] = \mathbf{E}[y_k] + \mathbf{E}[z_k] = \left(\frac{2}{k}\right)\mathbf{E}[x_{k+1}] + \mathbf{E}[x_{k+1}] = \left(\frac{k+2}{k}\right)\mathbf{E}[x_{k+1}].$$

Hence,

$$\begin{aligned} \mathbf{E}[x_k] &= \mathbf{E}[x_{n-1}]\left(\frac{n}{n-2}\right)\left(\frac{n-1}{n-3}\right)\cdots\left(\frac{k+4}{k+2}\right)\left(\frac{k+3}{k+1}\right)\left(\frac{k+2}{k}\right) \\ &= \left(\frac{2}{n-1}\right)\left(\frac{n}{n-2}\right)\left(\frac{n-1}{n-3}\right)\cdots\left(\frac{k+4}{k+2}\right)\left(\frac{k+3}{k+1}\right)\left(\frac{k+2}{k}\right) \\ &= \frac{2n}{k(k+1)}. \end{aligned}$$

Proof of Theorem 1. Let $\lfloor\lfloor k\rfloor\rfloor$ denote the largest power of two whose value is at most k, and let $\lceil\lceil k\rceil\rceil$ denote the smallest power of two whose value is at least k. Since f satisfies the polynomial growth condtion, $f(k) = \Theta(f(\lfloor\lfloor k\rfloor\rfloor))$. Assuming for the moment that n is a power of 2, we have

$$
\begin{aligned}
\mathbf{E}[T_n] &= f(n) + 2n\sum_{k=1}^{n-1}\frac{f(k)}{k(k+1)} \\
&= f(n) + n\Theta\left(\sum_{k=1}^{n-1}\frac{f(\lfloor\lfloor k\rfloor\rfloor)}{\lfloor\lfloor k\rfloor\rfloor^2}\right) \\
&= f(n) + n\Theta\left(\sum_{j=0}^{\log_2 n-1}\frac{f(2^j)}{(2^j)^2}2^j\right) \\
&= f(n) + \Theta\left(\sum_{j=0}^{\log_2 n-1} f(2^j)\frac{n}{2^j}\right) \\
&= \Theta\left(\sum_{j=0}^{\log_2 n} f(2^j)\frac{n}{2^j}\right) \\
&= \Theta(S_n).
\end{aligned}
$$

The last step follows from the fact that the penultimate expression gives the algebraic expansion of the recurrence (2). If n is not a power of two we bound $\mathbf{E}[T_n]$ between $\mathbf{E}[T_{\lfloor\lfloor n\rfloor\rfloor}]$ and $\mathbf{E}[T_{\lceil\lceil n\rceil\rceil}]$. From (1) we can argue straightforwardly that $\mathbf{E}[T_n]$ is monotonically nondecreasing if f is monotonically nondecreasing. Furthermore, the polynomial growth condition implies that $S_n = \Theta(S_{\lfloor\lfloor n\rfloor\rfloor}) = \Theta(S_{\lceil\lceil n\rceil\rceil})$. Thus,

$$\Theta(S_n) = \Theta(S_{\lfloor\lfloor n\rfloor\rfloor}) = \mathbf{E}[T_{\lfloor\lfloor n\rfloor\rfloor}] \leq \mathbf{E}[T_n] \leq \mathbf{E}[T_{\lceil\lceil n\rceil\rceil}] = \Theta(S_{\lceil\lceil n\rceil\rceil}) = \Theta(S_n).$$

"One-Sided" Divide and Conquer Algorithms. We wish to remark briefly on one other common type of randomized divide and conquer algorithm, in which we partition a size-n problem uniformly at random into two subproblems but recurse on only one of these. Prominent examples of such "one-sided" divide and conquer algorithms include the natural randomized generalization of binary search as well as Hoare's "Quickselect" algorithm [2]. If we select a subproblem on which to recurse at random (with probability proportional to the size of the subproblem), then one can apply analogous methods to those devloped above. In this case, one can show that $\mathbf{E}[x_k] = 2/(k+1)$ for $k < n$, giving us the following analog of (4):

$$\mathbf{E}[T_n] = f(n) + 2\sum_{k=1}^{n-1}\frac{f(k)}{k+1}. \tag{5}$$

This formula gives us a simple proof that it takes $\Theta(\log n)$ expected time to perform a randomized binary search for a random element in an n-element sorted array (here, $f(k) = \Theta(1)$), and that it takes $\Theta(n)$ expected time to select for a random order statistic in an n-element array ($f(k) = \Theta(k)$ in this case). Unfortunately, in practice one typically searches for a particular non-random array element or order statistic. In this situation, the decision of which subproblem to recursively solve is no longer a random event, and in order to obtain a bound on worst-case running time we must conservatively assume that the algorithm always recurses on the larger of its two subproblems. Here our methods break down, since there no longer seems to be simple expression for $\mathbf{E}[x_k]$.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* (MIT Press, 2001).

[2] C. A. R. Hoare. Algorithm 63 (PARTITION) and algorithm 65 (FIND). *Communications of the ACM* 4(7) (1961) 321-322.

[3] C. A. R. Hoare. Quicksort. *Computer Journal* 5(1) (1962) 10-15.

[4] R. Karp. Probabilistic Recurrence Relations. in: Proc. STOC (1991) 190-197.