Average-Case Analysis of Quicksort

Hanan Ayad

1 Introduction

Quicksort is a divide-and-conquer algorithm for sorting a list S of n comparable elements (e.g. an array of integers). The steps of quicksort can be summarized as follows.

1. If n is 0 or 1, then return.
2. Pick an element $p \in S$, which is called the *pivot*. We suppose that we pick randomly.
3. Partition $S - \{p\}$ into two disjoint sublists S_L (left sublist) and S_R (right sublist), as follows: $S_L = \{s \in S - \{p\} | s \leq p\}$, and $S_R = \{s \in S - \{p\} | s \geq p\}$
4. Recursively apply quicksort on each of the sublists S_L and S_R.

Given its recursive design, the analysis of quick sort involves solving the recurrence relation $t(n)$ that describes its run time. Its run time $t(n)$ is equal to the sum of run times of the two recursive calls and of the run time $f(n)$ required for selecting the pivot and partitioning S into S_L and S_R. We have $f(n) = \Theta(n)$, because this work can be done in one pass through S. Hence, $t(n)$ is given by:

$$t(n) = t(i) + t(n-i-1) + n, \quad \text{for } \; n > 1, \; \text{ and } \; t(0) = t(1) = 1, \tag{1}$$

where $i = |S_L|$ is the size of the left sublist.

2 Average-Case Analysis

For the pivoting and partitioning strategy defined above, we can assume that each possible size of S_L (and consequently S_R) is equally likely. Possible sizes are $0, 1 \ldots, n-1$, each having a probability $1/n$. Hence, the average value of the run time $t(n)$ in Eq. 1 is given by,

$$t(n) = \frac{1}{N}[\sum_{i=0}^{n-1} t(i) + t(n-i-1)] + n \tag{2}$$

Since $\sum_{i=0}^{n-1} t(i) = \sum_{i=0}^{n-1} t(n-i-1)$, $t(n)$ can be written as,

$$t(n) = \frac{2}{N}[\sum_{i=0}^{n-1} t(i)] + n \tag{3}$$

The steps below apply some algebraic manipulations in order to put the recurrence relation in a solvable form. First, multiplying by n, we get

$$nt(n) = 2[\sum_{i=0}^{n-1} t(i)] + n^2 \tag{4}$$

Substituting n by $n-1$, we get,

$$(n-1)t(n-1) = 2[\sum_{i=0}^{n-2} t(i)] + (n-1)^2 \tag{5}$$

By subtracting Eq. 5 from Eq. 4, we obtain,

$$nt(n) - (n-1)t(n-1) = 2t(n-1) + 2n - 1 \tag{6}$$

Rearranging and simplifying, we get,

$$nt(n) = (n+1)t(n-1) + 2n \tag{7}$$

Dividing both sides by $n(n+1)$, we get,

$$\frac{t(n)}{n+1} = \frac{t(n-1)}{n} + \frac{2}{n+1} \tag{8}$$

Applying back substitution we get,

$$\begin{array}{rcc} \frac{t(n)}{n+1} & = & \frac{t(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ & = & \frac{t(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ & = & \cdots \\ & = & \frac{t(1)}{2} + 2\displaystyle\sum_{i=3}^{n+1} \frac{1}{i} \\ & \approx & \frac{1}{2} + 2[\ln(n+1) + \gamma - \frac{3}{2}] \end{array}$$

where $\gamma \approx 0.577$ is Euler's constant. Therefore,

$$\begin{array}{rcc} \dfrac{t(n)}{n+1} & = & O(\log n) \quad \text{and hence,} \\ t(n) & = & O(n \log n). \end{array}$$

References

[1] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++*. Addison Wesley, 2006.