Runtime Analysis of Randomized Quicksort

We give a proof that the average case running time of randomized quicksort is $\Theta(n \log n)$.[1]

There are several slight variations of the quicksort algorithm, and although the exact running times are different for each, the asymptotic running times are all the same. We begin by presenting the following version of `Quicksort`, written in C++.

The Quicksort Algorithm

```
void Quicksort(int A[],int l, int r) {
   if (r > l) {
     int p = RPartition(A,l,r);
     Quicksort(A,l,p-1);
     Quicksort(A,p+1,r);
     }
}
```

The Random Partition Algorithm

```
int RPartition(int A[], int l, int r) {
    int piv=l+(rand()%(r-l+1);
    Swap(A[l],A[piv]);
    int i = l+1;
    int j = r;
    while (1) {
        while (A[i] <= A[l] && i<r) ++i;
        while (A[j] >= A[l] && j>l) --j;
        if (i >= j) {
           Swap(A[j],A[l]);
           return j;
           }
        else Swap(A[i],A[j]);
        }
}
```

We will base our analysis on this version of `Quicksort`. It is straightforward to see that the runtime of `RPartition` is $\Theta(n)$. (A proof of this is left to the reader). We start by developing a recurrence relation for the average case runtime of `Quicksort`.

Theorem 1: *Let $T(n)$ be the average case runtime of* `Quicksort` *on an array of size n. Then*

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n).$$

Proof: Since the pivot element is chosen randomly, it is equally likely that the pivot will end up at any position from l to r. That is, the probability that the pivot ends up at location $l + i$ is $1/n$ for each $i = 0, \ldots, r - l$. If we average over all of the possible pivot locations, we obtain

$$\begin{aligned}
T(n) &= \frac{1}{n}\left(\sum_{k=0}^{n-1} (T(k) + T(n-k-1))\right) + \Theta(n) \\
&= \frac{1}{n}\sum_{k=0}^{n-1} T(k) + \frac{1}{n}\sum_{k=0}^{n-1} T(n-k-1) + \Theta(n) \\
&= \frac{1}{n}\sum_{k=0}^{n-1} T(k) + \frac{1}{n}\sum_{k=0}^{n-1} T(k) + \Theta(n) \\
&= \frac{2}{n}\sum_{k=0}^{n-1} T(k) + \Theta(n) \\
&= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n).
\end{aligned}$$

[1] All logs are base 2.

The last step holds since $T(0) = 0$. □

We will need the following result in order to solve the recurrence relation.

Lemma 2: For any $n \geq 3$,

$$\sum_{k=2}^{n-1} k \log k \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2.$$

Proof: We can write the sum as

$$\sum_{k=2}^{n-1} k \log k = \sum_{k=2}^{\lceil n/2 \rceil - 1} k \log k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log k$$

Then we can bound $(k \log k)$ by $(k \log(n/2)) = k(\log n - 1)$ in the first sum, and by $(k \log n)$ in the second sum. This gives

$$\begin{aligned}
\sum_{k=2}^{n-1} k \log k &= \sum_{k=2}^{\lceil n/2 \rceil - 1} k \log k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log k \\
&\leq \sum_{k=2}^{\lceil n/2 \rceil - 1} k(\log n - 1) + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log n \\
&= (\log n - 1) \sum_{k=2}^{\lceil n/2 \rceil - 1} k + \log n \sum_{k=\lceil n/2 \rceil}^{n-1} k \\
&= \log n \sum_{k=2}^{\lceil n/2 \rceil - 1} k - \sum_{k=2}^{\lceil n/2 \rceil - 1} k + \log n \sum_{k=\lceil n/2 \rceil}^{n-1} k \\
&= \log n \sum_{k=2}^{n-1} k - \sum_{k=2}^{\lceil n/2 \rceil - 1} k \\
&\leq \log n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \\
&\leq (\log n) \frac{1}{2}(n-1)n - \frac{1}{2}(\frac{n}{2} - 1)\frac{n}{2} \\
&= \frac{1}{2} n^2 \log n - \frac{n}{2} \log n - \frac{1}{8} n^2 + \frac{n}{4} \\
&\leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2.
\end{aligned}$$

The last step holds since

$$\frac{n}{4} \leq \frac{n}{2} \log n,$$

when $n \geq 3$. □

Now we are ready for the final analysis.

Theorem 3: *Let $T(n)$ be the average case runtime of* `Quicksort` *on an array of size n. Then*

$$T(n) = \Theta(n \log n).$$

Proof: We need to show that $T(n) = O(n \log n)$ and $T(n) = \Omega(n \log n)$. To prove that $T(n) = O(n \log n)$, we will show that for some constant a,

$$T(n) \leq an \log n \text{ for all } n \geq 2.^2$$

When $n = 2$,

$$an \log n = a2 \log 2 = 2a,$$

and a can be chosen large enough so that $T(2) \leq 2a$. Thus, the inequality holds for the base case. Assume that $T(1) = C$, for some constant C. For $2 < k < n$, assume $T(k) \leq ak \log k$. Then

$$\begin{aligned}
T(n) &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n) \\
&\leq \frac{2}{n}\sum_{k=2}^{n-1} ak \log k + \frac{2}{n}T(1) + \Theta(n) && \text{(by assumption)} \\
&= \frac{2a}{n}\sum_{k=2}^{n-1} k \log k + \frac{2}{n}C + \Theta(n) \\
&\leq \frac{2a}{n}\sum_{k=2}^{n-1} k \log k + C + \Theta(n) && \text{(since } \tfrac{2}{n} \leq 1\text{)} \\
&\leq \frac{2a}{n}\left(\frac{1}{2}n^2 \log n - \frac{1}{8}n^2\right) + C + \Theta(n) && \text{(by Lemma 2)} \\
&= an \log n - \frac{a}{4}n + C + \Theta(n) \\
&= an \log n + \left(\Theta(n) + C - \frac{a}{4}n\right) \\
&\leq an \log n && \text{(choose } a \text{ so } \Theta(n) + C \leq \tfrac{a}{4}n\text{)}
\end{aligned}$$

We have shown that with an appropriate choice of a, $T(n) \leq an \log n$ for all $n \geq 2$, so $T(n) = O(n \log n)$. We leave it to the reader to show that $T(n) = \Omega(n \log n)$. □

The proof here is based on the one presented in Section 8.4 of [1]. The algorithm they give is slightly different, and they include some interesting insights. It is recommended that you read their proof.

References

[1] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[2] We pick 2 for the base case since $n \log n$=0 if $n = 1$, so we cannot make the inequality hold. Another solution would be to show that $T(n) \leq an \log n + b$. In this case, b can be chosen so that the inequality holds for $n = 1$.