CS 161	Lecture 5: Randomized Algorithms and Quicksort
Winter 2021	Wed, Jan 27

Adapted from Virginia Williams' lecture notes. Additional credits go to Sam Keller (2015), Seth Hildick-Smith (2016), Gregory Valiant (2017).

Please direct all typos and mistakes to Moses Charikar and Nima Anari (2021).

Today we'll study another sorting algorithm. Quicksort was invented in 1959 by Tony Hoare. You may wonder why we want to study a new sorting algorithm. We have already studied MergeSort, which we showed to perform significantly better than the trivial $O(n^2)$ algorithm. While MergeSort achieves an $O(n \log n)$ worst-case asymptotic bound, in practice, there are a number of implementation details about MergeSort that make it tricky to achieve high performance. Quicksort is an alternative algorithm, which is simpler to implement in practice. Quicksort will also use a divide and conquer strategy but will use randomization to improve the performance of the algortithm in expectation. Java, Unix, and C stdlib all have implementations of Quicksort as one of their built-in sorting routines.

1 Quicksort Overview

As in all sorting algorithms, we start with an array A of n numbers; again we assume without loss of generality that the numbers are distinct. Quicksort is very similar to the Select algorithm we studied last lecture. The description of Quicksort is the following:

if $length(A) \leq 1$ **then**
  return A `// It is trivially sorted`
else
  Pick some element $x \leftarrow A[i]$. `// We call` x `the pivot.`
  Split A into $A_< = \{A[i] \mid A[i] < x\}$ and $A_> = \{A[i] \mid A[i] > x\}$.
  Rearrange A into $[A_<, x, A_>]$.
  Recurse on $A_<$ and $A_>$.

The above steps define a "partition" function on A. The partition function of Quicksort can vary depending on how the pivot is chosen and also on the implementation. Quicksort is often used in practice because we can implement this step in linear time, and with very small constant factors. In addition, the rearrangement (Step 3) can be done in-place, rather than making several copies of the array (as is done in MergeSort). In these notes we will not describe the details of an in-place implementation, but the pseudocode can be found in CLRS.

2 Speculation on the Runtime

The performance of Quicksort depends on which element is chosen as the pivot. Assume that we choose the k^{th} smallest element; then $|A_<| = k-1, |A_>| = n-k$.

This allows us to write a recurrence; let $T(n)$ be the runtime of Quicksort on an n-element array. We know the partition step takes $O(n)$ time; therefore the recurrence is

$$T(n) \leq cn + T(k-1) + T(n-k).$$

For the worst pivot choice (the maximum or minimum element in the array), the runtime will resolve to $T(n) = T(n-1) + O(n) = O(n^2)$.

One way that seems optimal to define the partition function is to pick the *median* as the pivot. In the above recurrence this would mean that $k = \lceil \frac{n}{2} \rceil$. We showed in the previous lecture that the algorithm Select can find the median element in linear time. Therefore the recurrence becomes $T(n) \leq cn + 2T(\frac{n}{2})$. This is exactly the same recurrence as MergeSort, which means that this algorithm is guaranteed to run in $O(n \log n)$ time.

Unfortunately, the median selection algorithm is not practical; while it runs in linear time, it has much larger constant factors than we would like. To improve it, we will explore some alternative methods for choosing a pivot.

We leave the proof of correctness of Quicksort as an exercise to the reader (a proof by induction is suggested).

2.1 Random Pivot Selection

One method of “defending" against adversarial input is to choose a random element as the pivot. We observe that it is unlikely that the random element will be either the median (best-case) or the maximum or minimum (worst-case). Note that we have a uniform distribution over the n order statistics of the array (that is, for every $1 \leq i \leq n$ we pick the i-th highest element with the same probability $\frac{1}{n}$). However, for the first time, we encounter a randomized algorithm, and the analysis now becomes more complex. How do we analyze a randomized algorithm?

3 Worst-Case Analysis

In this section we will derive a bound on the worst-case running time of Quicksort. If we consider the worst random choice of pivot at each step, the running time will be $\Theta(n^2)$. We are thus interested in what is the running time of Quicksort on average over all possible choices of the pivots. Note that we still consider the running time for a worst-case input, and average only over the random choices of the algorithm (which is different from averaging over all possible inputs). We formalize the idea of averaging over the random choices of the algorithm by considering the running time of the algorithm on an input I as a random variable and bounding the expectation of that random variable.

Proposition 1. *For every input array of size n, the expected running time of Quicksort is $O(n \log n)$.*

Recall that a random variable (often abbreviated as RV) is a function that maps every element in the sample space to a real number. In the case of rolling a die, the real number (or value of the point in the sample space) would be the number on the top of the die. Here the sample space is the set of all possible choices of pivots, and an example for a random variable can be the running time of Quicksort on a specific input I.

Denote by z_i the i-th element in the sorted array. For each i, j, we define a random variable $X_{i,j}(\sigma)$ to be the number of times z_i and z_j are compared for a given series of pivot choices σ. What are the possible values for $X_{i,j}(\sigma)$? It can be 0 if z_i and z_j are not compared. Note that all comparisons are with the pivot, and that the pivot is not included in the elements of the arrays in the recursive calls. Thus, no two elements are compared twice. Therefore, $X_{i,j}(\sigma) \in \{0, 1\}$.

Our goal is to compute the expected number of comparisons that Quicksort makes. Recall the definition of expectation:

$$\mathbb{E}[X] = \sum_{\sigma} \mathbb{P}[\sigma] X(\sigma) = \sum_{k} k \, \mathbb{P}[X = k].$$

An important property of expectation is the linearity of expectation. For any random variables $X_1, \ldots, X_n$:

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i].$$

We start with computing the expected value of $X_{i,j}$. These variables are *indicator random variables*, which take the value 1 if some event happens, and 0 otherwise. The expected value is

$$\begin{aligned} \mathbb{E}[X_{i,j}] &= \mathbb{P}[X_{i,j} = 1] \cdot 1 + \mathbb{P}[X_{i,j} = 0] \cdot 0 \\ &= \mathbb{P}[X_{i,j} = 1] \end{aligned}$$

Let $C(\sigma)$ be the total number of comparisons made by Quicksort for a given set of pivot choices σ:

$$C(\sigma) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}(\sigma).$$

We wish to compute $\mathbb{E}[C]$ to get the expected number of comparisons made by Quicksort

for an input array of size n.

$$\begin{aligned}\mathbb{E}[C] &= \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}(\sigma)\right] \\ &= \sum_{i=1}^{n}\mathbb{E}\left[\sum_{j=i+1}^{n} X_{i,j}(\sigma)\right] \\ &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathbb{P}[z_i, z_j \text{ are compared}]\end{aligned}$$

Now we find $\mathbb{P}[z_i, z_j \text{ are compared}]$. Note that each element in the array except the pivot is compared to the pivot at each level of the recurrence. To analyze $\mathbb{P}[z_i, z_j \text{ are compared}]$, we need to examine the portion of the array $[z_i \ldots., z_j]$. After the array is split using a pivot from $[z_i \ldots., z_j]$, z_i and z_j can no longer be compared. Hence, z_i and z_j are compared only when from $[z_i \ldots., z_j]$, either z_i or z_j is the first one picked as the pivot. So,

$$\begin{aligned}\mathbb{P}[z_i, z_j \text{ compared}] &= \mathbb{P}[z_i \text{ or } z_j \text{ is the first pivot picked from } [z_i \ldots., z_j]] \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1}\end{aligned}$$

We return to the expected value of C:

$$\begin{aligned}\mathbb{E}[C] &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathbb{P}[z_i, z_j \text{ are compared}] \\ &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{2}{j-i+1}\end{aligned}$$

Note that for a fixed i,

$$\begin{aligned}\sum_{j=i+1}^{n}\frac{1}{j-i+1} &= \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1} \\ &\leq \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\end{aligned}$$

And using $\sum_{k=2}^{n}\frac{1}{k} \leq \ln n$, we get that

$$\begin{aligned}\mathbb{E}[C] &= \mathbb{E}[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}(\sigma)] \\ &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{2}{j-i+1} \\ &\leq 2n\ln n\end{aligned}$$

Thus, the expected number of comparisons made by Quicksort is no greater than $2n \ln n = O(n \log n)$. To complete the proof, we have to show that the running time is dominated by the number of comparisons. Note that in each recursive call to Quicksort on an array of size k, the algorithm performs $k - 1$ comparisons in order to split the array, and the amount of work done is $O(k)$. In addition, Quicksort will be called on single-element arrays at most once for each element in the original array, so the total running time of Quicksort is $O(C + n)$. In conclusion, the expected running time of Quicksort on worst-case input is $O(n \log n)$.

3.1 Alternative Proof

Here we provide an alternative method for bounding the expected number of comparisons. Let $T(n)$ be the *expected* number of comparisons performed by Quicksort on an input of size n. In general, if the pivot is chosen to be the i-th order statistic of the input array,

$$T(n) = n - 1 + T(i - 1) + T(n - i).$$

where we define $T(0) = 0$. Each of the n possible choices of i are equally likely. Thus, the expected number of comparisons is:

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n} \sum_{i=1}^{n} (T(i - 1) + T(n - i)) \\ &= n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} (T(i)) \end{aligned}$$

We use two facts:

1. $\sum_{i=1}^{n-1} f(i) \leq \int_1^n f(x)dx$ for an increasing function f.
2. $\int 2x \ln x dx = x^2 \ln x - \frac{x^2}{2} + C$.

Now we show that $T(n) \leq 2n \ln n$ by induction.

Inductive Hypothesis. $T(i) \leq 2i \ln i$ for all $i < n$.

Base case. An array of size 1 requires no comparisons. Thus, $T(1) = 0$.

Inductive step. We bound $T(n)$:

$$\begin{aligned}
T(n) &= n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} T(i) \\
&\leq n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} 2i \ln i \\
&\leq n - 1 + \frac{2}{n}\int_{1}^{n} (2x \ln x)\, dx \\
&= n - 1 + \frac{2}{n}\left[n^2 \ln n - \frac{n^2}{2} + \frac{1}{2}\right] \\
&= 2n \ln n + n - 1 - n + \frac{1}{n} \\
&= 2n \ln n - 1 + \frac{1}{n} \\
&\leq 2n \ln n.
\end{aligned}$$