Quick Sort

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Today we will discuss another sorting algorithm named quick sort. It is indeed quick in practice, but what is more interesting about the way it is designed and analyzed. As we will see, this is a randomized algorithm that runs in $O(n^2)$ time in the worst case (to sort n numbers), but $O(n \log n)$ time in expectation.

Recall:

The Sorting Problem

Problem Input:

A set S of n integers is given in an array of length n.

Goal:

Design an algorithm to store S in an array where the elements have been arranged in ascending order.

Quick Sort

We will denote the input array as A, and describe the algorithm by recursion.

Base Case. If $n = 1$, return directly.

Reduce. Otherwise, the algorithm runs the following steps:

1. Randomly pick an integer p in A—call it the pivot.
    - This can be done in $O(1)$ time using RANDOM$(1, n)$.
2. Re-arrange the integers in an array A' such that
    - All the integers smaller than p are positioned before p in A'.
    - All the integers larger than p are positioned after p in A'.
3. Sort the part of A' before p recursively.
4. Sort the part of A' after p recursively.

Example

After Step 1 (suppose that 26 was randomly picked as the pivot):

				p																									
38	28	88	17	26	41	72	83	69	47	12	68	5	52	35	9														

After Step 2:

				p																									
17	12	5	9	26	38	28	88	41	72	83	69	47	68	52	35														

After Steps 3 and 4:

				p																									
5	9	12	17	26	28	35	38	41	47	52	68	69	72	83	88														

Analysis of Quick Sort

Quick sort's running time is not attractive in the worst case: its worst case time is $O(n^2)$ (why?). However, quick sort is fast in expectation—we will prove next that its expected time is $O(n \log n)$. Remember: this holds on **every** input array A.

Remark: You may be wondering whether quick sort has any advantage over merge sort, which guarantees $O(n \log n)$ in the worst case. The answer is: no advantage in theory, but there is an advantage in practice—quick sort permits a faster implementation that leads to a smaller hidden constant compared to merge sort. We will discuss this in the tutorial.

The rest of the slides will not be tested.

Analysis of Quick Sort

First, convince yourself that it suffices to analyze the number X of comparisons. The running time is bounded by $O(n + X)$.

Next, we will prove that $\mathbf{E}[X] = O(n \log n)$.

Analysis of Quick Sort

Denote by e_i the i-th smallest integer in S. Consider e_i, e_j for any i, j such that $i \neq j$.

What is the probability that quick sort compares e_i and e_j?

This question—which seems to be difficult at first glance—has a surprisingly simple answer. Let us observe:

- Every element will be selected as a pivot precisely once.
- e_i and e_j are not compared, if any element between them gets selected as a pivot before them.
  - For example, consider $i = 7$ and $j = 12$. If e_9 is the pivot, then e_i and e_j will be separated by e_9. There is no chance that e_i and e_j can get compared in the subsequent execution.

Analysis of Quick Sort

Therefore, e_i and e_j are compared if and only if either one is the first among $e_i, e_{i+1}, ..., e_j$ picked as a pivot.

The probability is $2/(j - i + 1)$ (random pivot selection).

Analysis of Quick Sort

Define random variable X_{ij} to be 1, if e_i and e_j are compared. Otherwise, $X_{ij} = 0$. We thus have $\mathbf{Pr}[X_{ij} = 1] = 2/(j - i + 1)$. That is, $\mathbf{E}[X_{ij}] = 2/(j - i + 1)$.

Clearly, $X = \sum_{i,j} X_{ij}$. Hence:

$$
\begin{aligned}
\mathbf{E}[X] &= \sum_{i,j} \mathbf{E}[X_{ij}] = \sum_{i,j} \frac{2}{j - i + 1} \\
&= 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j - i + 1} \\
&= 2 \sum_{i=1}^{n-1} O(\log(j - i + 1)) \\
&= 2 \sum_{i=1}^{n-1} O(\log n) = O(n \log n).
\end{aligned}
$$

Analysis of Quick Sort

As a final remark, the above analysis used the following fact:

$$1 + 1/2 + 1/3 + 1/4 + ... + 1/n = O(\log n).$$

The left hand side is called the harmonic series, which is frequently encountered in computer since.