CSE 632 (Fall 2019): Analysis of Algorithms II : Randomized Algorithms

Lecture 8 (9/20/2019): Randomized Quick-Sort

Lecturer: Shi Li *Scriber: Xian Zhou*

1 Randomized version of quick-sort

Recall the following randomized quick-sort algorithm. For simplicity, we assume all the n elements in A are distinct.

Algorithm 1 randomized-quick-sort(A)

1: **if** $|A| \leq 1$ **then return** A
2: choose "a pivot" x from A uniformly at random
3: $B \leftarrow$ array of elements in A that are smaller than x
4: $C \leftarrow$ array of elements in A that are bigger than x
5: **return** randomized-quick-sort(B) concatenating (x) concatenating randomized-quick-sort(C)

Notice that (randomized-)quick-sort can be implemented as an "in-place" algorithm: we can share the memory between B, C and A. That is, after shuffling, we can the elements in A are placed such that: elements smaller than x appear before x, which appears before all elements bigger than x. Then, B and C are sub-arrays of A. So, we do not need an extra array of size $\Theta(n)$. This is one reason why quick-sort is better in practice than other $O(n \log n)$-time algorithms.

Recall that if x is always the median of A, then the running time of the algorithm will be $O(n \log n)$. There are two ways to analyze the running time:

- Using master theorem. The recurrence for the running time is $T(n) = 2T(n/2) + O(n)$; solving the recurrence gives $T(n) = O(n \log n)$.
- By expansion of the recursion tree. Each recursion step we divide the original array A into two equal parts, which yields to recursive calls for arrays of size $n/2$. The non-recursive procedure takes time $O(n)$. Expanding the whole recursion tree gives us a tree of depth $O(\log n)$. Each node at level i contributes $O(n/2^i)$ to the running time (the root has level 0, its children have level 1, and so on). There are 2^i nodes at level i. So, each level contributes to a running time of $O(n)$ and the total running time is $\mathrm{O}(n \log n)$.

2 Analyzing Expected Running Time of Randomized Quick-Sort

We give two different methods for analyzing the expected running time of the randomized quick-sort algorithm.

2.1 Direct Analysis Using Recursion

We assume for an array A of size $n = |A|$, the running time of the randomized-quick-sort for A, not counting the running time for the two recursive calls, is cn. Let $f(n)$ be the expected running time of the randomized quick-sort algorithm for an array of size n. Then, we prove the following lemma by mathematical induction:

Lemma 1. *Let $c' = 6c$. Then $f(n) \leq c' n \log n$ for every $n \geq 0$.* [1]

[1] Notice that the right side is 0 if $n = 1$; we can assume that we never call the procedure if the array size becomes 0 or 1.

Proof. The lemma holds for $n = 0$ and $n = 1$. Now suppose for some integer $n' \geq 2$ the lemma holds for every $n < n'$. We shall show that it holds for $n = n'$. We have

$$f(n) \leq \frac{1}{n}\sum_{i=1}^{n}\left[f(i-1) + f(n-i)\right] + cn$$
$$= \frac{2}{n}\sum_{i=1}^{n-1} f(i) + cn \leq \frac{2c'}{n}\sum_{i=1}^{n-1} i\log i + cn.$$

Above i is the rank of the pivot we have chosen. So, when the rank is i, the B and C arrays will have size $i-1$ and $n-i$ respectively. The cn is the running time for statements done within the recursion. In the first inequality, we implicitly used the linearity of expectation[2]. The second inequality used the induction hypothesis.

To prove $f(n) \leq c'n\log n$ when c' is large enough, we should show that $\frac{2}{n}\sum_{i=1}^{n-1} i\log i$ is $\Omega(n)$ less than $n\log n$; this way the difference can cover the cn term. We can first try to replace each $\log i$ by $\log n$. Then we get

$$\frac{2}{n}\sum_{i=1}^{n-1} i\log i \leq \frac{2}{n}\sum_{i=1}^{n-1} i\log n = \frac{2}{n}\left(\sum_{i=1}^{n-1} i\right)\log n = \frac{2}{n}\frac{(n-1)\,n}{2}\log n = (n-1)\log n.$$

The difference is only $\log n$, which is not big enough.

Now we try to save more: for $i \leq \lfloor \frac{n}{2} \rfloor$, we replace i with $\log \frac{n}{2}$ instead of $\log n$. We show that this saving will give an $\Omega(n)$ term:

$$\frac{2}{n}\sum_{i=1}^{n-1} i\log i \leq \frac{2}{n}\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} i\log\frac{n}{2} + \frac{2}{n}\sum_{i=\lfloor \frac{n}{2} \rfloor+1}^{n-1} i\log n$$
$$= \frac{2}{n}\sum_{i=1}^{n-1} i\log n - \frac{2}{n}\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} i \leq (n-1)\log n - \frac{2}{n}\frac{\lfloor \frac{n}{2} \rfloor\left(\lfloor \frac{n}{2} \rfloor + 1\right)}{2}$$
$$\leq n\log n - \frac{1}{n}\left(\frac{n^2}{4} - \frac{1}{4}\right) = n\log n - \frac{n}{4} + \frac{1}{4n} \geq n\log n - \frac{n}{6}.$$

Now this difference is enough to cover the cn term since we let $c' = 6c$:

$$f(n) \leq c'\left(n\log n - \frac{n}{5}\right) + cn \leq c'n\log n.$$

This finishes the proof of the lemma. □

2.2 Indirect Analysis via Counting number of Comparisons

Instead of analyze the running time directly, the second method tries to bound the expected number of comparisons in the algorithm. It is easy to see that the running time of the algorithm is of the same order as the number of comparisons the algorithm did.

For every $1 \leq i < j \leq n$, we let $D_{i,j}$ be 1 if the algorithm compared the i-th smallest number with the j-th smallest number, and 0 otherwise. Thus, the number of comparisons is exactly $\sum_{1\leq i<j\leq n} D_{i,j}$. [3]

By linearity of expectation, we have that the expected number of comparisons the algorithm does is exactly $\sum_{1\leq i<j\leq n} \mathbb{E}[D_{i,j}]$. So, we now focus on bounding $\mathbb{E}[D_{i,j}]$ for a fixed pair (i,j). Notice that this expectation depends on i, j. For example, if $i = 3$ and $j = 4$, then the algorithm has to compare the pair; while if $i = 1$ and $j = 100$, we can imagine that the algorithm compares the pair with very small probability. The lemma we shall prove is

Lemma 2. $\mathbb{E}[D_{i,j}] = \frac{2}{j-i+1}$.

[2]More specifically, we let $\tilde{f}(n)$ be the running time of the algorithm for an array of size n in one execution of the algorithm; so $\tilde{f}(n)$ is a randomized variable. Then, we have $\tilde{f}(n) \leq \frac{1}{n}\sum_{i=1}^{n}\left(\tilde{f}(i-1) + \tilde{f}(n-i)\right) + cn$. Taking $\mathbb{E}[\cdot]$ on both sides and using linearity of expectation gives the inequality.

[3]It is easy to see that we only compare each pair of numbers at most once.

Proof. For simplicity, let A' be the sorted array for A. We focus on the elements in A with rank between i and j (inclusive); that is, the elements in $A'[i..j]$. Consider an execution of the randomized quick-sort algorithm and start from the root recursion:

- If we chose a pivot $x < A'[i]$, then all elements in $A'[i \cdots j]$ will be passed to the right recursion. We consider the right recursion and repeat.
- Similarly, if we chose a pivot $x > A'[j]$, then we consider the left recursion and repeat.
- Finally if $A'[i] \le x \le A'[j]$, then $A'[i]$ and $A'[j]$ will be separated from this recursion. We define this recursion to be the "critical recursion" for the (i, j) pair.

Let x be the pivot chosen in the critical recursion for the pair (i, j). Then, if $x = A'[i]$ or $x = A'[j]$, then we shall compare the $A'[i]$ and $A'[j]$ in the recursion. However, if x is strictly between $A'[i]$ and $A'[j]$, then $A'[i]$ and $A'[j]$ will never be compared. Then the question is, what is the probability that $x = A'[i]$ or $x = A'[j]$? The answer is $\frac{2}{j-i+1}$ since all the numbers in $A'[i..j]$ will have equal chance of being x. Thus, we have $\mathbb{E}[D_{i,j}] = \frac{2}{j-i+1}$. So,

$$\begin{aligned}
&\mathbb{E}\left[\text{number of comparisons}\right] = \mathbb{E}\left[\sum_{1 \le i < j \le n} D_{i,j}\right] \\
&= \sum_{1 \le i < j \le n} \mathbb{E}\left[D_{i,j}\right] = 2 \sum_{1 \le i < j \le n} \frac{1}{j-i+1} \\
&\le 2n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) \\
&= \Theta\left(n \log n\right).
\end{aligned}$$

To see the inequality, we notice that there are $n - t \le n$ pairs (i, j) with $j - i = t$. The last equation used the fact that $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \Theta(\log n)$. □