Quicksort - Analysis of the average case

K. Subramani
Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

1 Strategy

In class, we showed that the worst-case complexity of Quick-sort is $O(n^2)$. In fact, an array which is already sorted is the worst-case input; this is because at each level of the recursion, we get an array of size 1 and another of size $length[array] - 1$ (Initially, $length[array] = n$). However, the worst-case analysis does not do justice to the performance of Quick-sort. It is and will be the algorithm of choice to implement sorting in institutions such as banks, airlines and even the federal government. We now show why this is the case.

The deterministic version of Quick-sort always partitioned the array $A[p..q]$ about the first element i.e. $A[p]$. In the randomized version of Quick-sort, we choose an element at random from $A[p, q]$ about which to partition the array.

```
Function RANDOMIZED-QUICK-SORT ($(A, p, q)$)

1: Let $r = random(p, q)$, i.e. $r$ is a random number between $p$ and $q$.
2: INTERCHANGE($A[p], A[r]$). { For example, let $p = 1, q = 100$; Let $r = 53$. Then $A[p] = A[53]$ and $A[53] = A[1]$
   after the exchange.}
3: $j$ = PARTITION($A, p, q$) { Note that the Partition is carried out about the new $A[p]$ which is the old $A[r]$. }
4: RANDOMIZED-QUICK-SORT($A, p, j$)
5: RANDOMIZED-QUICK-SORT($A, j + 1, q$)
```

Algorithm 1.1: Randomized Quick-Sort

2 Analysis

Definition: 2.1 *The rank of an element $a \in A$ is defined as the position index of a in A when A is sorted.*

For instance, the rank of element 3 in the array $A = [7, 2, 3, 9, 10]$ is 2.

Observation: 2.1 *The number j returned by* PARTITION *depends only the rank of the element $= A[p]$.*

Recollect that PARTITION partitons the array A into a left-hand portion $A[p..j]$ and a right-hand portion $A[j + 1..q]$. (Note that they are not equal in size as was the case with MERGE-SORT.) Let us compute the probability of various outcomes on the size of the left-hand portion. The following cases arise:

1. $rank[x] = 1$, i.e. the element used to achieve the partition is the smallest element in $A[p..q]$. If this is the case, then the left-hand portion will have size 1. This event has probability $\frac{1}{n}$, where $n = q - p + 1$ i.e. the number of elements in the call to PARTITION.

2. $rank[x] \geq 2$. When PARTITION terminates, each of $rank[x]-1$ elements is in the left-hand side, while the other elements are in the right-hand side. Thus, for each $i = 2, 3, \ldots, n-1$, when $rank[x] \geq 2$, the probability is $\frac{1}{n}$, that the left-hand side has i elements.

   Combining these two cases, we conclude that the size $j-p+1$ of the left-hand side is 1 with probability $\frac{2}{n}$ ($\frac{1}{n}$ when $rank[x]=1$ and $\frac{1}{n}$ when $rank[x]=2$), while the size is i with probability $\frac{1}{n}$ for $i = 2, 3, \ldots n-1$.

We now need a recurrence to describe the expected running time of the algorithm. Observe that PARTITION takes $\Theta(n)$ time to split up an array of size n into two portions of size q and $n-q$. Hence, we have

$$\begin{aligned} T(n) &= \frac{2}{n}(T(1)+T(n-1)) + \frac{1}{n}((T(2)+T(n-2)) + (T(3)+T(n-3)) + \ldots (T(n-1)+T(1))) + \Theta(n) \\ &= \frac{1}{n}(T(1)+T(n-1)+\sum_{q=1}^{n-1}(T(q)+T(n-q))) + \Theta(n) \qquad (1) \end{aligned}$$

Observe that $T(1) = O(1)$ and from our worst-case analysis, we know that $T(n-1) = O(n^2)$. Accordingly,

$$\frac{1}{n}(T(1)+T(n)) = \frac{1}{n}(O(1)+O(n^2)) = O(n) \qquad (2)$$

The $O(n)$ term in (2) can be absorbed in the $\Theta(n)$ term in (1) to give

$$T(n) = \frac{1}{n}\sum_{q=1}^{n}(T(q)+T(n-q)) + \Theta(n) \qquad (3)$$

Exploiting the symmetry of $T(q)$ and $T(n-q)$, we rewrite the above equation as:

$$T(n) = \frac{2}{n}\sum_{k=1}^{n} T(k) + \Theta(n) \qquad (4)$$

We now guess that $T(n) \leq cn.\log n$ and verify that this guess satisfies Equation (4) using mathematical induction. (The text uses $a.n\log n + b$ instead of $c.n\log n$ and the reason is they want 1 to be their base case. I prefer to use 2 as the base case as it keeps the analysis simpler!!)
Accordingly, using the principle of strong induction, we have that

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

But what is $\sum_{k=1}^{n-1} T(k)$. Observe Exercise $8.4-5$ on page 167. Thus, we have

$$\sum_{k=1}^{n-1} T(k) \leq \frac{n^2}{2}.\log n - \frac{n^2}{2} - \frac{1}{4} \qquad (5)$$

Hence, we have

$$\begin{aligned} T(n) &\leq \frac{2.c}{n}(\frac{n^2}{2}.\log n - \frac{n^2}{2} - \frac{1}{4}) + \Theta(n) \\ \Rightarrow T(n) &\leq c.n.\log n - c.n - \frac{1}{4.n} + \Theta(n) \\ \Rightarrow T(n) &\leq c.n.\log n, \qquad (6) \end{aligned}$$

for sufficiently large c.

3 Sophisticated Analysis

The analysis in Section §2, while informative, is not particularly elegant. In this section, we provide an analysis for RANDOMIZED-QUICK-SORT given in [MR95].

Let S denote the input array and let $S(i)$ denote the element having rank i. Let X_{ij} denote an indicator variable that takes value 1, if elements $S(i)$ and $S(j)$ (i.e. the elements of ranks i and j respectively) are compared in the execution and 0 otherwise Thus, X_{ij} counts the number of comparisons between $S(i)$ and $S(j)$ (X_{ij} can be at most 1). Hence, we are interested in the expected value of $\sum_{i=1}^{n} \sum_{j>i} X_{ij}$ which represents the total number of comparisons (Why ?). Note that,

$$\mathbf{E}[\sum_{i=1}^{n} \sum_{j>i} X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} \mathbf{E}[X_{ij}] \tag{7}$$

by linearity of expectation.

How to calculate $\mathbf{E}[X_{ij}]$? Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared in an execution. Since X_{ij} assumes values 0 and 1 only, we have,

$$\mathbf{E}[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij} \tag{8}$$

Now all that remains, is the calculation of p_{ij}. Towards that end, we view the execution of Randomized Quick-Sort as a binary tree, which is constructed as follows: The root of the tree is the first element that is picked i.e. the element around which the initial partition is carried out. Let us denote this element by y. Observe that in the PARTITION procedure, every element in S is compared against y. The comparisons result in the elements which are less than or equal to y, being thrown in the left subtree and the other elements being thrown in the right subtree. This procedure is then recursively carried out at the roots of the left and right subtrees.

Consider the permutation π of elements in the level-order traversal of the binary tree constructed as above. (Level-order traversal means, visit all the nodes at level i before visiting the nodes at level $(i+1)$ i.e. a breadth-first search of the tree.)

Observation: 3.1 *Two elements $S(i)$ and $S(j)$ can be compared if and only if either $S(i)$ or $S(j)$ occurs earlier in the permutation π than any element $S(l)$ such that $i < l < j$. To see this let $S(k)$ be the earliest element in the permutation π which belongs to the set of elements having rank between i and j. If $k \notin \{i, j\}$, then $S(i)$ will belong to the left subtree of the tree rooted at $S(k)$, while $S(j)$ will belong to the right subtree and hence they cannot be compared. Conversely, if $k \in \{i, j\}$, then the comparison between $S(i)$ and $S(j)$ will be made.*

Observation: 3.2 *The elements between $S(i)$ and $S(j)$ are all equally likely to be picked in a call to* PARTITION. *Hence, the probability that $S(i)$ and $S(j)$ are compared is exactly $\frac{2}{j-i+1}$.*

Using the formula for p_{ij} from Observation (3.2) in Equations (7) and (8), we get

$$\begin{aligned} \sum_{i=1}^{n} \sum_{j>i} p_{ij} &= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\ &\leq \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\ &\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\ &\leq 2n. \ln n \\ &= O(n. \log n) \end{aligned}$$

References

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, England, June 1995.