Chapter 3

Analyzing QuickSort and QuickSelect via Expectation

By Sariel Har-Peled, April 26, 2022[1]

NOBODY expects the Spanish Inquisition! Our chief weapon is surprise...surprise and fear...fear and surprise.... Our two weapons are fear and surprise...and ruthless efficiency.... Our *three* weapons are fear, surprise, and ruthless efficiency...and an almost fanatical devotion to the Pope.... Our *four*...no... *Amongst* our weapons.... Amongst our weaponry...are such elements as fear, surprise....

The Spanish Inquisition, Monty Python

3.1. QuickSort

Let the input be a set $T = \{t_1, \ldots, t_n\}$ of n items to be sorted. We remind the reader, that the **QuickSort** algorithm randomly pick a pivot element (uniformly), splits the input into two subarrays of all the elements smaller than the pivot, and all the elements larger than the pivot, and then it recurses on these two subarrays (the pivot is not included in these two subproblems). Here we will show that the expected running time of **QuickSort** is $O(n \log n)$.

Let $S_1, \ldots, S_n$ be the elements in their sorted order (i.e., the output order). Let $X_{ij} = 1$ be the indicator variable which is one $\iff$ **QuickSort** compares S_i to S_j, and let p_{ij} denote the probability that this happens. Clearly, the number of comparisons performed by the algorithm is $C = \sum_{i<j} X_{ij}$. By linearity of expectations, we have

$$\mathbb{E}\left[C\right] = \mathbb{E}\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} \mathbb{E}\left[X_{ij}\right] = \sum_{i<j} p_{ij}.$$

We want to bound p_{ij}, the probability that the S_i is compared to S_j. Consider the last recursive call involving both S_i and S_j. Clearly, the pivot at this step must be one of $S_i, \ldots, S_j$, all equally likely. Indeed, S_i and S_j were separated in the next recursive call.

Observe, that S_i and S_j get compared if and only if pivot is S_i or S_j. Thus, the probability for that is $2/(j-i+1)$. Indeed,

$$p_{ij} = \mathbb{P}\left[S_i \text{ or } S_j \text{ picked} \,\middle|\, \text{picked pivot from } S_i, \ldots, S_j\right] = \frac{2}{j-i+1}.$$

Thus,

$$\sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} 2/(j-i+1) = \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \le 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \le 2nH_n \le n + 2n \ln n,$$

where H_n is the ***harmonic number***[2] $H_n = \sum_{i=1}^{n} 1/i$. We thus proved the following result.

[1]This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/3.0/` or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

[2]Using integration to bound summation, we have $H_n \le 1 + \int_{x=1}^{n} \frac{1}{x} dx \le 1 + \ln n$. Similarly, $H_n \ge \int_{x=1}^{n} \frac{1}{x} dx = \ln n$.

Lemma 3.1.1. **QuickSort** *performs in expectation at most $n + 2n\ln n$ comparisons, when sorting n elements.*

Note, that this holds for all inputs. No assumption on the input is made. Similar bounds holds not only in expectation, but also with high probability.

This raises the question, of how does the algorithm pick a random element? We assume we have access to a random source that can get us number between 1 and n uniformly.

Note, that the algorithm always works, but it might take quadratic time in the worst case.

Remark 3.1.2 (Wait, wait, wait). Let us do the key argument in the above more slowly, and more carefully. Imagine, that before running **QuickSort** we choose for every element a random priority, which is a real number in the range $[0, 1]$. Now, we re-implement **QuickSort** such that it always pick the element with the lowest random priority (in the given subproblem) to be the pivot. One can verify that this variant and the standard implementation have the same running time. Now, a_i gets compares to a_j if and only if all the elements $a_{i+1}, \ldots, a_{j-1}$ have random priority larger than both the random priority of a_i and the random priority of a_j. But the probability that one of two elements would have the lowest random-priority out of $j - i + 1$ elements is $2 * 1/(j - i + 1)$, as claimed.

3.2. QuickSelect: Median selection in linear time

3.2.1. Analysis via expectation and indicator variables

We remind the reader that **QuickSelect** receives an array $\mathcal{T}[1 \ldots n]$ of n real numbers, and a number k, and returns the element of rank k in the sorted order of the elements of $\mathcal{T}$, see Figure 3.1. We can of course, use **QuickSort**, and just return the kth element in the sorted array, but a more efficient algorithm, would be to modify **QuickSelect**, so that it recurses on the subproblem that contains the element we are interested in. Formally, **QuickSelect** chooses a random pivot, splits the array according to the pivot. This implies that we now know the rank of the pivot, and if its equal to $\overline{\mathsf{m}}$, we return it. Otherwise, we recurse on the subproblem containing the required element (modifying $\overline{\mathsf{m}}$ as we go down the recursion. Namely, **QuickSelect** is a modification of **QuickSort** performing only a single recursive call (instead of two).

As before, to bound the expected running time, we will bound the expected number of comparisons. As before, let $S_1, \ldots, S_n$ be the elements of t in their sorted order. Now, for $i < j$, let X_{ij} be the indicator variable that is one if S_i is being compared to S_j during the execution of **QuickSelect**. There are several possibilities to consider:

(i) If $i < j < \overline{\mathsf{m}}$: Here, S_i is being compared to S_j, if and only if the first pivot in the range $S_i, \ldots, S_k$ is either S_i or S_j. The probability for that is $2/(k - i + 1)$. As such, we have that

$$\alpha_1 = \mathbb{E}\Big[\sum_{i<j<\overline{\mathsf{m}}} X_{ij}\Big] = \mathbb{E}\left[\sum_{i=1}^{\overline{\mathsf{m}}-2}\sum_{j=i+1}^{\overline{\mathsf{m}}-1} X_{ij}\right] = \sum_{i=1}^{\overline{\mathsf{m}}-2}\sum_{j=i+1}^{\overline{\mathsf{m}}-1} \frac{2}{\overline{\mathsf{m}} - i + 1} = \sum_{i=1}^{med-2} \frac{2(\overline{\mathsf{m}} - i - 1)}{\overline{\mathsf{m}} - i + 1} \le 2(\overline{\mathsf{m}} - 2).$$

(ii) If $\overline{\mathsf{m}} < i < j$: Using the same analysis as above, we have that $\mathbb{P}\big[X_{ij} = 1\big] = 2/(j - \overline{\mathsf{m}} + 1)$. As such,

$$\alpha_2 = \mathbb{E}\left[\sum_{j=\overline{\mathsf{m}}+1}^{n}\sum_{i=\overline{\mathsf{m}}+1}^{j-1} X_{ij}\right] = \sum_{j=\overline{\mathsf{m}}+1}^{n}\sum_{i=\overline{\mathsf{m}}+1}^{j-1} \frac{2}{j - \overline{\mathsf{m}} + 1} = \sum_{j=\overline{\mathsf{m}}+1}^{n} \frac{2(j - \overline{\mathsf{m}} - 1)}{j - \overline{\mathsf{m}} + 1} \le 2\Big(n - \overline{\mathsf{m}}\Big).$$

```
QuickSelect($\mathcal{T}[\![1:n]\!]$, $k$)
    // Input: $\mathcal{T}[\![1:n]\!]$ array with $n$ numbers, parameter $k$.
    // Assume all numbers in $t$ are distinct.
    // Task: Return $k$th smallest number in $\mathcal{T}$.
    $y \leftarrow$ random element of $\mathcal{T}$.
    $r \leftarrow$ rank of $y$ in $\mathcal{T}$.
    if $r = k$ then return $y$
    $\mathcal{T}_< =$ array with all elements in $\mathcal{T} <$ than $y$
    $\mathcal{T}_> =$ all elements in $\mathcal{T} >$ than $y$
    // By assumption $|\mathcal{T}_<| + |\mathcal{T}_>| + 1 = |\mathcal{T}|$.
    if $r < k$ then
        return QuickSelect( $\mathcal{T}_>$, $k - r$ )
    else
        return QuickSelect( $\mathcal{T}_<$, $k$ )
```

Figure 3.1: **QuickSelect** pseudo-code.

(iii) $i < \overline{\mathsf{m}} < j$: Here, we compare S_i to S_j if and only if the first indicator in the range $S_i, \ldots, S_j$ is either S_i or S_j. As such, $\mathbb{E}[X_{ij}] = \mathbb{P}[X_{ij} = 1] = 2/(j - i + 1)$. As such, we have

$$\alpha_3 = \mathbb{E}\left[\sum_{i=1}^{\overline{\mathsf{m}}-1} \sum_{j=\overline{\mathsf{m}}+1}^{n} X_{ij}\right] = \sum_{i=1}^{\overline{\mathsf{m}}-1} \sum_{j=\overline{\mathsf{m}}+1}^{n} \frac{2}{j - i + 1}.$$

Observe, that for a fixed $\Delta = j - i + 1$, we are going to handle the gap Δ in the above summation, at most $\Delta - 2$ times. As such, $\alpha_3 \le \sum_{\Delta=3}^{n} 2(\Delta - 2)/\Delta \le 2n$.

(iv) $i = \overline{\mathsf{m}}$. We have $\alpha_4 = \displaystyle\sum_{j=\overline{\mathsf{m}}+1}^{n} \mathbb{E}[X_{ij}] = \sum_{j=\overline{\mathsf{m}}+1}^{n} \frac{2}{j - \overline{\mathsf{m}} + 1} = \ln n + 1$.

(v) $j = \overline{\mathsf{m}}$. We have $\alpha_5 = \displaystyle\sum_{i=1}^{\overline{\mathsf{m}}-1} \mathbb{E}[X_{ij}] = \sum_{i=1}^{\overline{\mathsf{m}}-1} \frac{2}{\overline{\mathsf{m}} - i + 1} \le \ln \overline{\mathsf{m}} + 1$.

Thus, the expected number of comparisons performed by **QuickSelect** is bounded by

$$\sum_i \alpha_i \le 2(\overline{\mathsf{m}} - 2) + 2(n - \overline{\mathsf{m}}) + 2n + \ln n + 1 + \ln \overline{\mathsf{m}} = 4n - 2 + \ln n + \ln \overline{\mathsf{m}}.$$

Theorem 3.2.1. *In expectation,* **QuickSelect** *performs at most $4n - 2 + \ln n + \ln \overline{\mathsf{m}}$ comparisons, when selecting the $\overline{\mathsf{m}}$th element out of n elements.*

A different approach can reduce the number of comparisons (in expectation) to $1.5n + o(n)$. More on that later in the course.

3.2.2. Analysis of QuickSelect via conditional expectations

Consider the problem of given a set X of n numbers, and a parameter k, to output the kth smallest number (which is the number with ***rank*** k in X). This can be easily be done by modifying **QuickSort** only to perform one recursive call. See Figure 3.1 for a pseud-code of the resulting algorithm.

Intuitively, at each iteration of **QuickSelect** the input size shrinks by a constant factor, leading to a linear time algorithm.

Theorem 3.2.2. *Given a set X of n numbers, and any integer k, the expected running time of* **QuickSelect**(X, n) *is $O(n)$.*

Proof: Let $X_1 = X$, and X_i be the set of numbers in the ith level of the recursion. Let y_i and r_i be the random element and its rank in X_i, respectively, in the ith iteration of the algorithm. Finally, let $n_i = |X_i|$. Observe that the probability that the pivot y_i is in the "middle" of its subproblem is

$$\alpha = \mathbb{P}\left[\frac{n_i}{4} \leq r_i \leq \frac{3}{4}n_i\right] \geq \frac{1}{2},$$

and if this happens then

$$n_{i+1} \leq \max(r_i - 1, n_i - r_i) \leq \frac{3}{4}n_i.$$

We conclude that

$$\begin{aligned}\mathbb{E}[n_{i+1} \mid n_i] &\leq \mathbb{P}[y_i \text{ in the middle}]\frac{3}{4}n_i + \mathbb{P}[y_i \text{ not in the middle}]n_i \\ &\leq \alpha\frac{3}{4}n_i + (1-\alpha)n_i = n_i(1-\alpha/4) \leq n_i(1-(1/2)/4) = (7/8)n_i.\end{aligned}$$

Now, we have that

$$\begin{aligned}m_{i+1} &= \mathbb{E}[n_{i+1}] = \mathbb{E}[\mathbb{E}[n_{i+1} \mid n_i]] \leq \mathbb{E}[(7/8)n_i] = (7/8)\,\mathbb{E}[n_i] = (7/8)m_i \\ &= (7/8)^i m_0 = (7/8)^i n,\end{aligned}$$

since for any two random variables we have that $\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}\big[X \,\big|\, Y\big]\big]$. In particular, the expected running time of **QuickSelect** is proportional to

$$\mathbb{E}\left[\sum_i n_i\right] = \sum_i \mathbb{E}[n_i] \leq \sum_i m_i = \sum_i (7/8)^i n = O(n),$$

as desired. ■