Chapter 3

Analyzing QuickSort and QuickSelect via Expectation

By Sariel Har-Peled, April 26, 2022

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 if and only if 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

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

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,

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

Thus,

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

where $H_n$ is the harmonic number $H_n = \sum_{i=1}^{n} 1/i$. 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 $H_n \leq 1 + \int_{x=1}^{n} \frac{1}{x} \, dx \leq 1 + \ln n$. Similarly, $H_n \geq \int_{x=1}^{n} \frac{1}{x} \, dx = \ln n$. 

---
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 compared 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 \( 2 \times 1/(j - i + 1) \), 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 \( T[1 \ldots n] \) of \( n \) real numbers, and a number \( k \), and returns the element of rank \( k \) in the sorted order of the elements of \( T \), see Figure 3.1. We can of course, use QuickSort, and just return the \( k \)th 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 \( m \), we return it. Otherwise, we recurse on the subproblem containing the required element (modifying \( m \) 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 \( i \) 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 \( i < j < m \): 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

\[
\alpha_1 = \mathbb{E} \left[ \sum_{i<j<m} X_{ij} \right] = \mathbb{E} \left[ \sum_{i=1}^{m-2} \sum_{j=i+1}^{m-1} X_{ij} \right] = \sum_{i=1}^{m-2} \sum_{j=i+1}^{m-1} \frac{2}{m-i+1} = \sum_{i=1}^{m-2} \frac{2(m-i-1)}{m-i+1} \leq 2(m-2).
\]

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

\[
\alpha_2 = \mathbb{E} \left[ \sum_{j=m+1}^{n} \sum_{i=m+1}^{j-1} X_{ij} \right] = \sum_{j=m+1}^{n} \sum_{i=m+1}^{j-1} \frac{2}{j-m+1} = \sum_{j=m+1}^{n} \frac{2(j-m-1)}{j-m+1} \leq 2(n-m).
\]
QuickSelect$(T[1:n], k)$

// Assume all numbers in $T$ are distinct.
// Task: Return $k$th smallest number in $T$.

$y \leftarrow$ random element of $T$.
$r \leftarrow$ rank of $y$ in $T$.

if $r = k$ then return $y$

$T_\prec$ = array with all elements in $T$ less than $y$
$T_\succ$ = all elements in $T$ greater than $y$

// By assumption $|T_\prec| + |T_\succ| + 1 = |T|$.

if $r < k$ then
    return QuickSelect$(T_\succ, k - r)$
else
    return QuickSelect$(T_\prec, k)$

Figure 3.1: QuickSelect pseudo-code.

(iii) $i < m < j$: 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, $E[X_{ij}] = P[X_{ij} = 1] = 2/(j - i + 1)$. As such, we have

$$a_3 = E\left[\sum_{i=1}^{m-1} \sum_{j=m+1}^{n} X_{ij}\right] = \sum_{i=1}^{m-1} \sum_{j=m+1}^{n} \frac{2}{j - i + 1}.$$ 

Observe, that for a fixed $\Delta = j - i + 1$, we are going to handle the gap $\Delta$ in the above summation, at most $\Delta - 2$ times. As such, $a_3 \leq \sum_{\Delta=3}^{n} 2(\Delta - 2)/\Delta \leq 2n$.

(iv) $i = m$. We have $a_4 = \sum_{j=m+1}^{n} E[X_{ij}] = \sum_{j=m+1}^{n} \frac{2}{m + 1} = \ln n + 1$.

(v) $j = m$. We have $a_5 = \sum_{i=1}^{m-1} E[X_{ij}] = \sum_{i=1}^{m-1} \frac{2}{m - i + 1} \leq \ln m + 1$.

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

$$\sum_{i} a_i \leq 2(m - 2) + 2(n - m) + 2n + \ln n + 1 + \ln m = 4n - 2 + \ln n + \ln m.$$ 

Theorem 3.2.1. In expectation, QuickSelect performs at most $4n - 2 + \ln n + \ln m$ comparisons, when selecting the $m$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 $k$th 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 $i$th level of the recursion. Let $y_i$ and $r_i$ be the random element and its rank in $X_i$, respectively, in the $i$th 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

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

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

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

Now, we have that

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

since for any two random variables we have that $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X | Y]]$. 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. \[\blacksquare\]