Randomized Algorithms Analysis

Wilmer Uruchi Ticona

December 11, 2018

Abstract

The following is an experiment of the running time of three algorithms to find the median of a vector. The first algorithm is Random QuickSort followed by a selection of the $[n/2]$ (position of the median) item in the vector. The second algorithm is Random QuickSelect aiming for the $[n/2]$ (position of the median) item in the vector. The third and last algorithm is RMedian, a randomized algorithm to find the median.

1 The Experiment

Three algorithms to find the median were implemented using python:

- Random **QuickSort** Expected complexity: $O(n \log n)$ Worst case: $O(n^2)$
- Random **QuickSelect** Expected complexity: $O(n)$ Worst case: $O(n^2)$
- **RMedian** Expected complexity: $O(n)$

The code also includes a data generation class that builds the sample data for the experiment. For this case 200 vectors were generated, each with 50001 random 3-digit integers. In order to expand the sample space of the experiment, the first 100 vectors were left as they were generated, distributed uniformly at random, the next 50 vectors were ordered in ascending order, and the last 50 vectors were ordered in descending order. As a result, the python program generated the file *data.txt*, where the vectors are stored in an easy-to-read array, where the vectors are distributed in the following way:

- **0-99** (Vectors not ordered)
- **100-149** (Vectors in ascending order)
- **150-199** (Vectors in descending order)

Then, the vectors in the file are read by the *RandomTests* class of the program. This class hosts in its methods the algorithms that are going to be run for the experiment. After reading the data, the class stores the data in a 2-dimensional array, and then performs the 3 algorithms for each of the 200 vectors. Code has been implemented to make sure that each algorithm receives the same input for each case. After all the experiments have been performed, the program stores the results in the file *resultsExperiment.txt*.

The results are stored in the form of data separated by commas that follows the structure:

Table 1: Results Structure

Column	Detail
Array	Index of vector (array).
timeQSort	Running time of algorithm in seconds.
timeQSelect	Running time of algorithm in seconds.
timeRMedian	Running time of algorithm in seconds.
RMedianSuccess	Shows the result of RMedian (to identify when it fails).

After the experiment has been completed and the results have been generated, this file is imported into *RStudio* to build the corresponding scatter plot and data summary table:

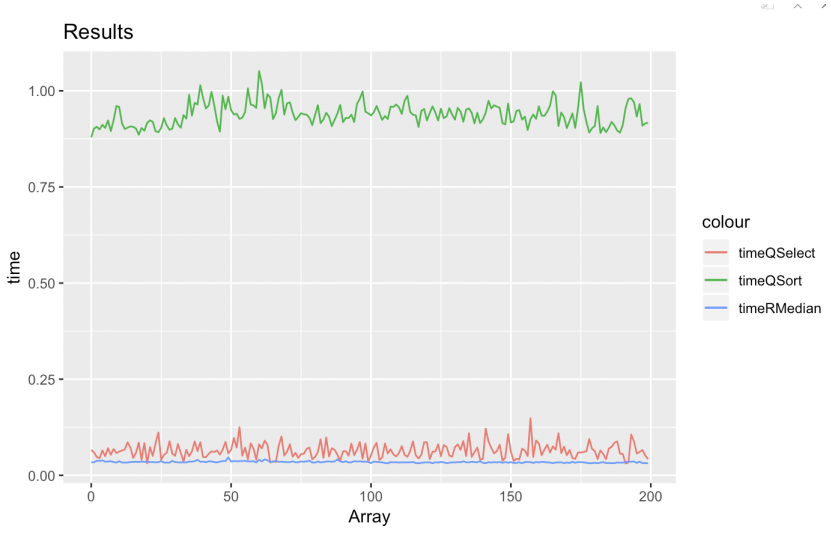

Figure 1: Plot of results

mean_qsort <dbl>	mean_qselect <dbl>	sd_qselect <dbl>	mean_rmedian <dbl>	sd_rmedian <dbl>
0.9383383	0.06414497	0.01922242	0.03451528	0.002084763

Figure 2: Summary of results

2 Which algorithm is faster?

As we can see in the scatter plot of the result of the experiment, the mean running time of Random QuickSelect and RMedian are very close, and both are much lower than the mean running time of Random QuickSort. Between them, Random QuickSelect shows a higher standard deviation because, although of expected complexity $O(n)$, has a worst case scenario complexity of $O(n^2)$. By using the randomized version of this algorithm we are making the probability of hitting the worst case scenario very low, however, there are scenarios that will generate the variation found in the results. On the other hand, RMedian maintains a nearly linear running time, which is assured by the algorithm, and a mean running time of almost half of that of Random QuickSelect.

Although there is at least one case in which Random QuickSelect performed better than RMedian (in a vector that was already ordered in descending order), it is almost certain that RMedian will perform better in the vast majority of cases. In conclusion, **RMedian** is the faster algorithm.

3 Which algorithm is more reliable?

We have concluded that RMedian is the faster algorithm. There is the fact that Random QuickSort and Random QuickSelect are always going to give a result, but in an expected running time higher than that of RMedian. However, RMedian does not always guarantee a result. There are three events so that if at least one of them happen, the result would be "FAIL", indicating that the algorithm couldn't find the median. These events are:

$Event_1 : Y_1 = |r \in R | r \leq m| < \frac{1}{2}n^{3/4} - \sqrt{n}$

If this happens, then the median was not in the sample space.

$Event_2 : Y_2 = |r \in R | r \geq m| < \frac{1}{2}n^{3/4} - \sqrt{n}$

If this happens, then the median was not in the sample space.

$Event_3 : |C| > 4n^{3/4}$

The sample space is too big to guarantee linear time.

By Applying Chebyshev's inequality it is possible to bound the probability of $Event_1$:

$$Pr[Event_1] \leq \frac{1}{4} n^{-1/4}$$

We also know a bound for the probability of $Event_3$, that is, that the median was not in the sample space:

$$Pr[Event_3] \leq \frac{1}{2} n^{-1/4}$$

If $Event_3$ happens, then at least one of these two events happens:

$Event_{3,1}$: at least $2n^{3/4}$ elements of C are greater than the median.
$Event_{3,2}$: at least $2n^{3/4}$ elements of C are smaller than the median.

Again, applying Chebyshev's inequality we can find a bound for these events (they have the same bound because of symmetry).

$$Pr[Event_{3,2}] \leq \frac{1}{4} n^{-1/4}$$

So we have that:

$$Pr[Event_3] \leq Pr[Event_{3,1}] + Pr[Event_{3,2}] \leq \frac{1}{2} n^{-1/4}$$

Combining the bounds, we have that the probability that the algorithm outputs FAIL is bounded by:

$$Pr[Event_1] + Pr[Event_2] + Pr[Event_3] \leq n^{-1/4}$$

In the case of the experiments performed, that would be:

$$Pr[FAIL] \leq 50001^{-1/4}$$

This bound is fairly high, but it is possible to build an iterative version of RMedian that guarantees a solution with a random running time. This version of the RMedian algorithm will have an expected linear running time.

In conclusion, in its current state (not iterative) we cannot say that RMedian is more reliable than Randomized QuickSelect or Random Pivot QuickSort, which always return a valid solution. So the option is between them, and in that case I think the faster one, **Randomized QuickSelect** would satisfy most criteria to be considered the most reliable of the three.