Assignment 2

Randomized Algorithms (RA)

Master in Innovation and Research in Informatics - Advanced Computing (MIRI-AC)
Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya (FIB, UPC)

Irene Simó Muñoz
irene.munoz@estudiantat.upc.edu

November 6th, 2022

Contents

1 Quickselect

The object of this experiment is to analyze the behavior of the Randomized version of Hoare's Find (1961), also known as the Quickselect algorithm. Its efficiency will be evaluated in terms of the average number of comparisons.

The algorithm is a type of randomized version of the classical Quickselect that improves the worst-case computational complexity of $\mathcal{O}(n \log n)$ by randomizing the selection of the pivot element for each recursion. This is equivalent to shuffling the elements before picking the next pivot, as well.
Some improvements in regards to the number of comparisons performed have been observed for even more careful picks of the pivot; see median-of-$(2t+1)$ in Section 3 -. This first approach, however, is devoted to the study of the recursivity of the standard version "median-of-1" randomized quickselect.

1.1 Implementation

Let j be any ordinal in the total of elements per array, n. Quickselect can be used to find the j-th element from such an array. For a chosen fixed value of $n = 20000$ elements per array, quickselect has been used to find all possible elements 1-th (first) to 20000-th (last).

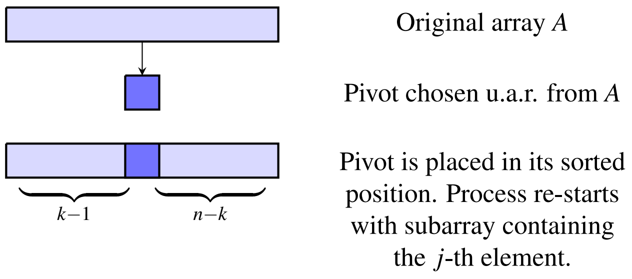
Original array A
Pivot chosen u.a.r. from A
Pivot is placed in its sorted position. Process re-starts with subarray containing the j-th element.
$k-1$
$n-k$

Figure 1: Scheme of randomized quickselect algorithm. Own elaboration.

Quickselect will place the chosen pivot at its exact position for the sorted array without the need to sort the rest of the elements. Instead, the rest of them will only be split either at the left of the pivot, if they are smaller or equal to the pivot; or at its right; otherwise. Quickselect will evaluate if the j-th element is to be found in the left or right subarray, and proceed to recursively call itself to perform the same operation only for the necessary split. This is schematically represented in Figure 1

The implementation of this algorithm consists of three main operations; the main procedure, swapping, and random partitioning
The general recursive procedure is detailed in Algorithm 1, in which the key steps are recursively calling the own procedure only for the useful subarray [3]. The PARTITION and swapping procedures are represented in Figure 1, but in essence, can be reduced to choosing uniformly at random an element from the given array and performing swapping operations from elements of the same array.

To extract useful statistical conclusions, the programs have been run 996 times, ensuring an accuracy of 9.5% with confidence of 90% - by the Sampling Theorem -.

1.2 Results

The presented data showcases the performance of Randomized Quickselect for the specified methodology. The average comparisons for each j-th element finding have been

Algorithm 1: QUICKSELECT

```
Input: An array A, lower and upper indexes p and
       q and the element to find g
Output: Index r of goal element g

if $p \leq q$ then
    r←PARTITION(A, p, q, g)
    if $r<g$ then return QUICKSELECT(A, p, r-1, g)
    else if $r>g$ then return QUICKSELECT(A,
      r+1, q, g)
    else return r
end
```

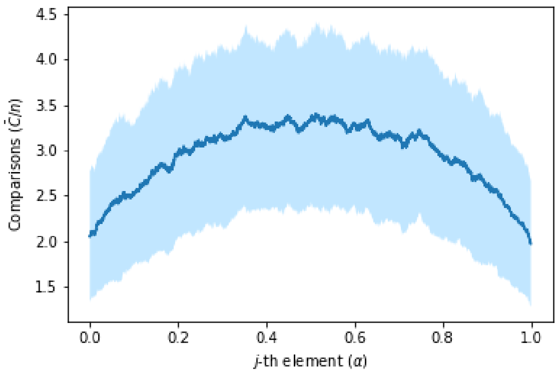

Figure 2: Results for Quickselect algorithm

normalized using $\alpha = j/n$.

Two main obersvations from Figure 2 are the high variance of the values and the peak of comparisons for $\alpha = 0.5$. This peak corresponds to the median finding.

1.3 Analysis & conclusions

The recursivity of Randomized Quickselect can be studied via the Continuous Master Theorem [4] as it's a divide-and-conquer algorithm.
Since for each recursive step, the pivot is chosen uniformly at random from the remaining elements, the probability of a particular element being chosen as the pivot is (from general expression (3.1)):

$$\pi_{n,k} = \frac{\binom{k-1}{0}\binom{n-k}{0}\binom{1}{1}}{\binom{n}{1}} = \frac{1}{n} \tag{1.1}$$

And the recurrence:

$$F_n^{(j)} = n - 1 + \sum \pi_{n,k}\mathbb{E}[\text{\# comp if pivot is the j-th}]$$

The toll cost $t_n = n-1$ is the cost of partitioning, and by symmetry, we can define the expected value of remaining recursive calls given that the pivot is the j-th element. This

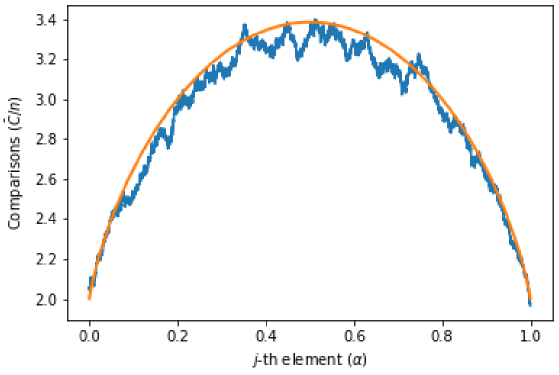

Figure 3: Results for Quickselect algorithm

will be the sum of the case when the j-th is smaller than the pivot, the j-th is larger than the pivot and the j-th is the pivot.

$$\begin{aligned}\mathbb{E}[\text{\# comp if pivot is the j-th}] = &\frac{j-1}{n}f_{j-1}+\\ &+\frac{n-j}{n}f_{n-j}+\\ &+\frac{1}{n}0\end{aligned}$$

Recursivity for the cases when the chosen pivot is not the j-th element is actually equivalent by symmetry. The previous expression can be rewritten as:

$$F_n^{(j)} = n - 1 + 2\pi_{n,k}\sum_{0\leq k<n}\frac{k}{n}f_k^{(j)} \tag{1.2}$$

With a shape function of $\omega(z) = 2z$, applying CMT yields $H = 1 - \int_0^1 2z^2dz = 1/3 > 0$, and:

$$F_n^{(j)} = \frac{t_n}{H} + o(t_n) = 3n + o(n)$$

The general case of Quickselect recurrence (for Median-of-$(2t+1)$) can be found in Section 3.
The expected number of comparisons for the normalized value of $\alpha = j/n$, for all values of j:

$$\lim_{n\to\infty}\frac{F_n^{(j)}}{n} = 2 + 2H(\alpha)$$

Where the cost entropy $H(\alpha)$ is:

$$H(\alpha) = -\alpha\ln(\alpha) - (1-\alpha)\ln(1-\alpha) \tag{1.3}$$

This is plotted along the previously presented results in Figure 3.
In conclusion, Quickselect performs best when the element to find is closer to the extremes, this is; j is either close to 0 or to n. This is rather logical, provided looking for these elements allows us to disregard larger subarrays. In consequence, the peak of comparisons is done when looking for the median, for which $\alpha = 0.5$.

2 Ramdomized Selection of Floyd-Rivest

This experiment aims to analyze the number of comparisons performed by Floyd-Rivest's or Randomized Selection algorithms for the median-finding case, where the median is defined as the j-th element such that $j = \lfloor (n+1)/2 \rfloor$

A naïve approach to a median finding algorithm would be to simply sort the element and find the element with rank $n/2$. This proceeding is effective but requires a computational cost of sorting and finding; $\mathcal{O}(n \log n)$ (for deterministic quicksort or mergesort, for instance) and $\mathcal{O}(1)$ (constant time) respectively.
Floyd-Rivest's randomized selection algorithm does the same job in linear time $\mathcal{O}(n)$. The key resides in the fact that for sorting n^a elements, if $n^a < n$, or, equivalently, $a < 1$, the cost of sorting is:

$$\mathcal{O}(n \log n)|_{n^a} = \mathcal{O}(n^a \log n^a) < \mathcal{O}(n)$$

The success of the Floyd-Rivest algorithm requires that the median is actually between keys d and u, and the size of C is small enough so that sorting can be done in linear time.

2.1 Implementation

This algorithm either returns the correct answer or fails. The probability of failure, with decreases as the input array size grows, is briefly described after the general procedure is described.

Floyd Rivest will firslty select a R sample of $\lceil n^{3/4} \rceil$ elements from the original array. After sorting it, it will strategically pick two elements d and u, which should serve as lower and upper bounds of the median, respectively. It can be shown that this is the case with high probability, and therefore picking a second sample C of elements from the original array for elements only larger or equal than d or smaller or equal than u should ensure that the median is in C. Sorting C and picking its median will return the median of the original array. This process is schematized in Figure 4.

However, there are three possible failing outputs for this randomized algorithm:

1. Size of smaller elements is bigger than half of the array. In other words, elements of A smaller than d represent more than half of the size of the array A.
2. Size of greater elements is bigger than half of the array. Similarly, elements of A larger than u represent more than half of the size of the array A.
3. Size of C array is larger or equal than $4n^{3/4}$. This can happen both if there at least $2n^{3/4}$ in C are greater or smaller than the median m.

Using Binomial distributions, Chebyshev's bounding and Union bound, it can be proved that the probability of the

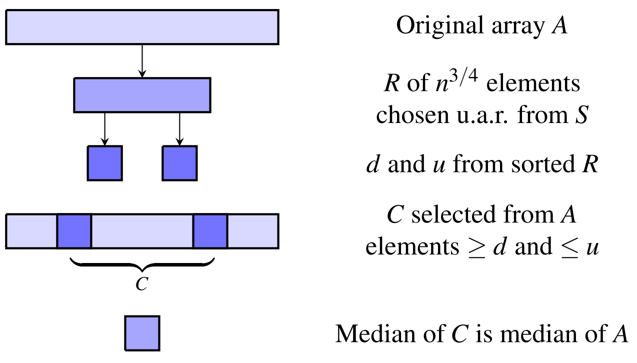
Original array A
R of $n^{3/4}$ elements chosen u.a.r. from S
d and u from sorted R
C selected from A elements $\geq d$ and $\leq u$
C
Median of C is median of A

Figure 4: Scheme of Floyd Rivest algorithm. Own elaboration.

algorithm succeeding is always

$$\mathbb{P}[\text{algorithm succeeds}] \geq 1 - \frac{1}{n^{1/4}}$$

The implementation is composed of the main procedure, a sorting algorithm, and the randomized partition and swapping routines. The sorting algorithm of choice is Quickselect, which follows the scheme of Quickselect almost identically. The main procedure, with the actions to put a flag when an error is encountered, is presented in Algorithm 2.

Algorithm 2: FLOYD RIVEST

```
Input: An array A of size $n$
Output: Median of A

R ← RAN SELECT(A, $n^{3/4}$)
d, u ← QUICKSORT(R)
for iterate over A do
    C ← A[i]≥d or A[i]≤u
    if $ld > n/2$ then Put error flag
    if $lu > n/2$ then Put error flag
    if $|C| > 4n^{3/4}$ then Put error flag
end
Median ← QUICKSORT(C)
return Median
```

To extract useful statistical conclusions, the programs have been run 996 times, ensuring an accuracy of 9.5% with confidence of 90% - by the Sampling Theorem -.

2.2 Results

The presented results consist of a brief comment on the failures encountered when running Floyd Rivest, and a comparison of the two studied algorithms for median finding.

In regards to the failure frequency, the plot in Figure 5 shows the frequency of failure for arrays from size 1 to 20k. The information given by this data is not useful except to

Array size	Quickselect ($\alpha = 0.5$)	Floyd Rivest
20k	3.3812	3.3952
50k	3.4194	2.8811
100k	3.4023	2.5939
500k	3.3392	2.0070
1.000k	3.3363	1.8142

Table 1: Normalized number of comparisons for median finding. Average values.

Array size	Quickselect ($\alpha = 0.5$)	Floyd Rivest
20k	0.9753	0.1862
50k	0.9460	0.1421
100k	1.0050	0.1089
500k	0.9460	0.0697
1.000k	0.9363	0.0544

Table 2: Normalized standard deviation for the number of comparisons for median finding.

indicate that the failures in arrays sufficiently large are negligible. In fact, no failures were detected in the run of this experiment.
The region of the plot for arrays from size 1 to 100, approximately, shows an incredibly high frequency of failure. Taking into account that Chebyshev's inequality, which bounds the probability of failing down to 0.6 at most for arrays of 100 elements, the definition of indexes such as d, u or the size of the R sample for arrays so small leads to inconsistencies - accessing elements not in arrays, for instance -. Mind that these kinds of algorithms are designed for large data management, and this comment is rather redundant.

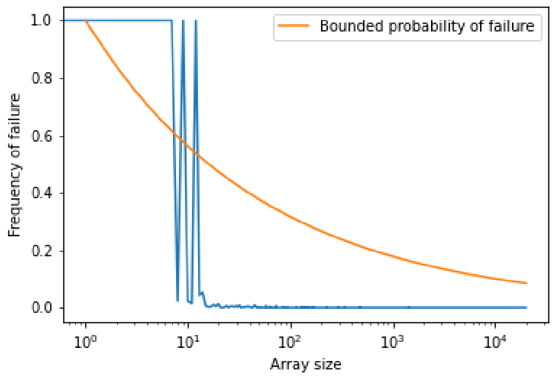

Figure 5: Failure probability of Floyd Rivest algorithm depending on array size.

The two studied algorithms can perform quick searches on arrays to find particular elements. Their performance varies especially depending on which j-th element the user is interested in, and the array size. For the median, the previous section's results indicated that Quickselect performs the most comparisons on average. For small arrays, even this peak on Quickselect's cost makes it a more efficient option than Floyd Rivest. However, for large arrays, the normalized average of comparisons is better for the latter than for the former. The following Tables 1 and 2 show the average values of the normalized comparisons and their respective standard deviations.

2.3 Analysis & conclusions

Randomized Median algorithm becomes useful for large arrays, for which finding the median is costly comparison-wise, as seen in the previous section in the Quickselect analysis.
The number of comparisons for Randomized Median is evergrowing, of course, as more comparisons are needed to find the median as larger the array is. The normalized comparisons, however, decrease as the array size grows, as the partitioning and sampling performed in this procedure benefit from large inputs.

In conclusion, Floyd Rivest becomes the better option for median finding if the array is large enough.

3 Bonus: Median-of-$(2t+1)$ quickselect

3.1 Implementation

There are possible improvements for Randomized Quickselect that rely on a more careful selection of the pivot at each recursive call [1,3]. Median-of-$(2t+1)$ algorithms select the pivot as the median of a randomly sampled set of elements from the original array. A simple scheme of the procedure is shown in Figure 6.
For Median-of-$(2t+1)$ algorithms, comparisons performed to select the median (pivot) are not accounted for [1]. In the implementation carried out for this report, the proposed algorithm relies on a rather naïve approach for this step, which although not impacting the obtained results, is worth mentioning for the reader's interest. The total amount of comparisons is computed exactly as done for the Quickselect analysis in the first section of this report.
In particular, Median-of-three has been tested for this report, although the produced script takes t as a parameter and can be used for further exploration of Median-of-$(2t+1)$ quickselect. The results are studied in this general scope.

The implementation is almost identical to the aforementioned Quickselect, with a modification in the Randomized partition procedure. For arrays larger than $(2t+1)$ size, the pivot is not chosen randomly, but instead, a sample of size $(2t+1)$ is taken from the given array and the median of such sample is used as a pivot for the partition.
This ensures that although still random, the pivot is chosen more carefully and will be closer to the median, hence partitioning more evenly and eventually reducing the total comparisons.

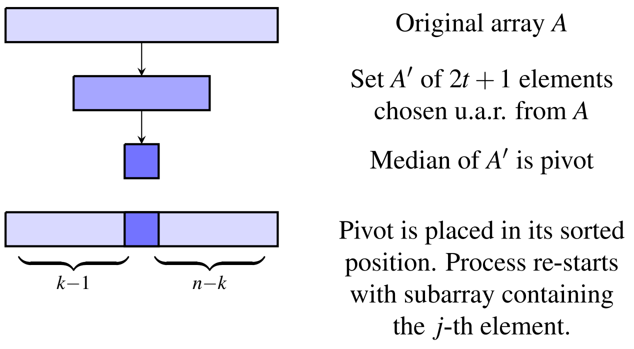
Original array A
Set A' of $2t+1$ elements chosen u.a.r. from A
Median of A' is pivot
Pivot is placed in its sorted position. Process re-starts with subarray containing the j-th element.
$k-1$
$n-k$

Figure 6: Scheme of median-of-$(2t+1)$ quickselect algorithms. Own elaboration.

3.2 Results

As done for the Qucickselect study, the results include the normalized plot for the average number of comparisons with the variance.

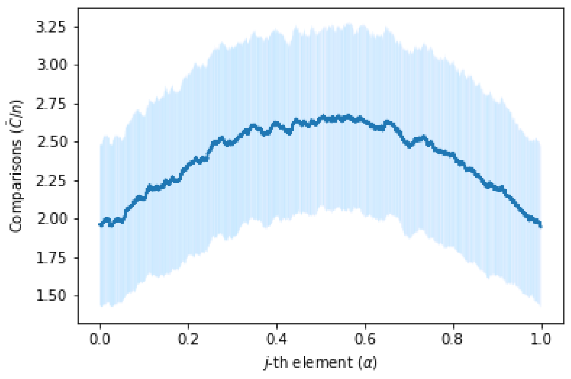

Figure 7: Results for Median-of-3 Quickselect algorithm

3.3 Analysis & conclusions

Any modification that ensures that the chosen pivot (k) is closer to the element that is being looked for (j) or, equivalently, less tending to the extremes, improves the expected number of comparisons. [1]

From a general perspective, all median-of-$(2t+1)$ quickselect possibilities can be analyzed similarly.

The probability of an element being chosen as the pivot is [2]:

$$\pi_{n,j} = \frac{\binom{j-1}{t}\binom{n-j}{t}}{\binom{n}{2t+1}} \tag{3.1}$$

And the recurrence:

$$f_n^{(t)} = n - 1 + \frac{2}{n}\sum_{j=t+1}^{n-t} \pi_{n,j}\frac{j-1}{n} f_j^{(t)}$$

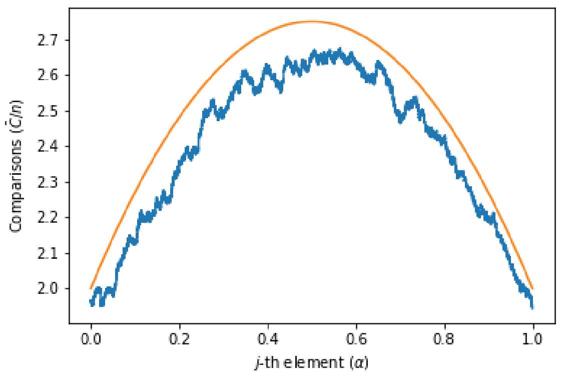

Figure 8: Results for Median-of-3 Quickselect algorithm

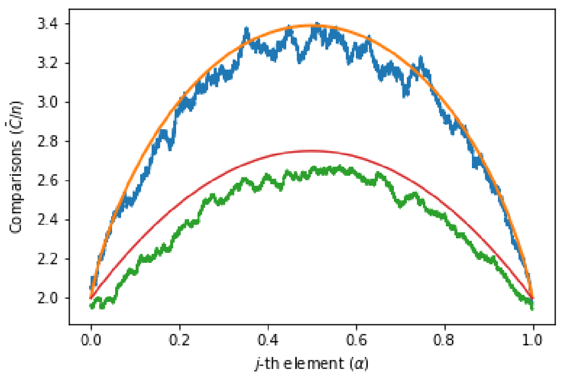

Figure 9: Results for standard (blue) and Median-of-3 (green) Quickselect algorithm along with the theoretical predictions (orange and red)

Following the Continuous Master Theorem [4], one finds the cost entropy is:

$$H = \frac{t+1}{2t+3}$$

And since it is strictly $H > 0\ \forall t \in \{0, \cdots, \infty\}$, the recurrence of Median-of-$(2t+1)$ is:

$$f_n^{(t)} = \frac{n}{H}\begin{cases} 3 \text{ if } t = 0 \\ \frac{5}{2} \text{ if } t = 1 \\ \frac{7}{3} \text{ if } t = 2 \\ \dots \end{cases} \tag{3.2}$$

Moreover, for Median-of-three, the number of comparisons can be expressed as:

$$\lim_{n\to\infty} \frac{f_n^{(t,j)}}{n} = 2 + 3\alpha(1-\alpha)$$

This is plotted in Figure 7. Figure 9 show already presented data for Randomized Quickselect and Median-of-three Quickselect in the same frame to highlight the improvement in comparisons between both versions.

References

[1] Peter Kirschenhofer, Helmut Prodinger, and Conrado Martinez. Analysis of hoare's find algorithm with median-of-three partition. *Random Structures & Algorithms*, 10(1-2):143–156, 1997.

[2] Conrado Martínez and Salvador Roura. Optimal sampling strategies in quicksort and quickselect. *SIAM Journal on Computing*, 31(3):683–705, 2001.

[3] Conrado Martínez Parra, Alois Panholzer, and Helmut Prodinger. The analysis of approximate quickselect and related problems. 2009.

[4] Salvador Roura. Improved master theorems for divide-and-conquer recurrences. *Journal of the ACM (JACM)*, 48(2):170–205, 2001.