International Journal of Computer Applications (0975 - 8887)
Volume 174 - No.9, September 2017

Probabilistic Analysis of Perfect Partitioning in Randomized Quicksort

Vivek Kumar
THDC - Institute of Hydropower Engineering and Technology
Uttarakhand
India

Mahesh Kumar Aghwariya
THDC - Institute of Hydropower Engineering and Technology
Uttarakhand
India

ABSTRACT

The paper analyzes the probability of a scenario where Randomized-Quicksort performs a perfect partitioning of the input array. The RANDOMIZED PARTITION procedure, which is a subroutine of the Randomized-Quicksort, randomly picks an element of the given array as the pivot element, it then partitions the array around that element. A perfect partitioning occurs when every successive call to the RANDOMIZED-PARTITION procedure results in the picking of the median element as the pivot element, which partitions the array into two halves consisting of exact no. of elements. In this scenario, the algorithm yields an Θ(n lg n) runtime.

Keywords

Quicksort, Randomized-Quicksort

1. INTRODUCTION

Sorting is one of the primitive task that is faced during the designing of various software systems. Over the years many sorting algorithms have been proposed which perform the task in efficient manner. One such algorithm is Quicksort, developed by C. A. R. Hoare[1]in 1962, the algorithm is recipient of much of the research work in the field of sorting. The algorithm is inflicted with $\Theta(n^2)$ worst case runtime, if the array is already sorted. A randomized version of quicksort however, ensures an Θ(n lg n) expected runtime for an array of size n. This variant of Quicksort differs from the classical algorithm in the way it chooses the pivot element for partitioning. While the classical Quicksort deterministicaly picks the pivot element in the given array, the randomized version of quicksort makes the choice randomly i.e, it randomly picks the pivot elements for partitioning. The probabilistic analyses of the randomized version of quicksort gives a Θ(n lg n) expected run time of the algorithm for an array of size n.

In this paper, the probability of perfect partitioning in Randomized-Quicksort is explored. As discussed, perfect partitioning occurs when every successive call to the Randomized-partition procedure of the Randomized-Quicksort yields a median element as the pivot element, which partitions the array into two equal halves. The algorithm in this case always gives an Θ(n lg n) runtime. The next section presents the randomized version of Quicksort and the later section discusses the probability of such an event.

2. RANDOMIZED QUICKSORT

The randomized version of quicksort[2] consists of three procedures. In addition to the two procedures of classical quicksort, RANDOMIZED-QUICKSORT has one more procedure, the RANDOMIZED-PARTITION. The RANDOMIZED-PARTITION procedure (Algorithm 2) picks a random index, i, between array indices p and r, it then swaps the element at index i with element at index r. Next, a call to the PARTITION procedure (Algorithm 1) is made which partitions the input array A around this randomly chosen pivot element. The RANDOMIZED-QUICKSORT (Algorithm 2) recursively partitions the array until it is arranged in sorted (ascending) order.
The RANDOMIZED-QUICKSORT procedure has been proved to yield a $\Theta(n\ lg\ n)$ expected running time. The next section deliberates the probability of an event where the RANDOMIZED-PARTITION procedure performs a perfect partitioning.

```
1:  function PARTITION(A, p, r)
2:      x ← A[r]
3:      i ← p − 1
4:      for j ← p to r − 1 do
5:          if A[j] ≤ x then
6:              i ← i + 1
7:              exchange A[i] with A[j]
8:          end if
9:          exchange A[i + 1] with A[r]
10:     end for
11:     return i + 1
12: end function
```

Algorithm 1: The PARTITION procedure

```
1: function RANDOMIZED-PARTITION(A, p, r)
2:     i ← RANDOM(p, r)
3:     exchange A[r] with A[i]
4:     return PARTITION(A, p, r)
5: end function
```

Algorithm 2: The RANDOMIZED-PARTITION procedure

```
1: function RANDOMIZED-QUICKSORT(A, p, r)
2:     if p < r then
3:         q ← RANDOMIZED-PARTITION(A, p, r)
4:         RANDOMIZED-QUICKSORT(A, p, q-1)
5:         RANDOMIZED-QUICKSORT(A, q+1, r)
6:     end if
```

7: **end function**

Algorithm 3: The RANDOMIZED-QUICKSORT procedure

3. THE PERFECT PARTITIONING

The RANDOMIZED-PARTITION procedure randomly picks an element as the pivot element. This element may or may not be the median element of the array. Consider a case where the RANDOMIZED-PARTITION procedure always picks the median element as the pivot element. This will always lead to the partitioning of the array into two equal halves in every successive call to the RANDOMIZED-PARTITION procedure. In such a scenario the runtime for RANDOMIZED-QUICKSORT will be

$$T(n) = T(n/2) + T(n/2) + \Theta(n) \tag{1}$$

which will give an $\Theta(n\ lg\ n)$ runtime for an array of size n.

Let us define the event E_i, for *i = 1, 2, ...,n* to be the event that an ith recursive call to the RANDOMIZED-PARTITION procedure leads to the picking of the median elements as the pivot element of all the sub-arrays in the same recursive depth. In other words,

E_1	the event where the first call to the RANDOMIZED-PARTITION yields the median element as the pivot element leading to the partitioning of the array into two equal halves consisting of $n/2$ elements each.
E_2	the event where the second call to the RANDOMIZED-PARTITION yields the median elements of the two arrays (resulting from the first call) leading to the partitioning of the array into 4 equal halves consisting of $n/4$ elements each.
.	.
.	.
.	.
E_n	the event where every single element is pivot in itself and the array is sorted.

The probability that we are seeking is the probability of the intersection of the events $E_1 \cap E_2 \cap ... \cap E_n$, and so we have

$$Pr\{E_1 \cap E_2 \cap ... \cap E_n\} = Pr\{E_1\}.Pr\{E_2|E_1\}.Pr\{E_3|E_1 \cap E_2\}... Pr\{E_n|E_1 \cap E_2 \cap ...E_{n-1}\} \tag{2}$$

There are two assumptions that are taken during the course of this analysis. Firstly, we ignore the floor and ceiling on the size of the partitioned array, i.e, even though for an array of size n with the pivot element being the median, the partitioned array will contain $n/2$ and $n/2 - 1$ elements respectively, we will assume the size to be $n/2$ and $n/2$ respectively. Secondly, the size of the array is assumed to be a power of 2, i.e, $n = 2^k$ for some integer, $k \geq 0$.

Initially, for the original array of size n, the probability of RANDOMIZED-PARTITION picking the median element as the pivot element will be

$$Pr\{E_1\} = \frac{1}{n} \tag{3}$$

Post this event, the array will be partitioned into two equal halves containing $n/2$ elements each. The probability of RANDOMIZED-PARTITION picking median elements of these two arrays as the respective pivot elements is

$$Pr\{E_2|E_1\} = \frac{1}{n/2}.\frac{1}{n/2} = \frac{2^2}{n^2} \tag{4}$$

Similarly,

$$Pr\{E_3|E_1 \cap E_2\} = \frac{1}{n/4}.\frac{1}{n/4}.\frac{1}{n/4}.\frac{1}{n/4} = \frac{(2^2)^{2^2}}{n^{2^2}} \tag{5}$$

and so on. The probability of intersection of these events becomes (from eq. 2)

$$Pr\{E_1 \cap E_2 \cap ... \cap E_n\} = \prod_{i=0}^{lg\ n} \left(\frac{2^i}{n}\right)^{2^i} \tag{6}$$

$$\begin{aligned}
Pr\{E_1 \cap E_2 \cap ... \cap E_n\} &= \frac{1}{n} \times \left(\frac{2}{n}\right)^2 \times \left(\frac{2^2}{n}\right)^{2^2} \times .....\times \left(\frac{2^{lg\,n-1}}{n}\right)^{2^{lg\,n-1}} \times 1 \\
lg(Pr) &= lg\left(\frac{1}{n}\right) + lg\left(\frac{2}{n}\right)^2 + lg\left(\frac{2^2}{n}\right)^{2^2} + ...+ lg\left(\frac{2^{lg\,n-1}}{n}\right)^{2^{lg\,n-1}} + 0 \\
&= lg\left(\frac{1}{n}\right) + 2lg\left(\frac{2}{n}\right) + 2^2 lg\left(\frac{2^2}{n}\right) + ...+ 2^{lgn-1} lg\left(\frac{2^{lgn-1}}{n}\right) + 0 \\
&= lg1 - lgn + 2 - 2lgn + 2.2^2 - 2^2 lgn + 3.2^3 - 2^3 lgn + ... + (lgn-1).2^{lg\,n-1} - 2^{lg\,n-1} lgn \\
&= (lg1 + 2 + 2.2^2 + 3.2^3 + ...+ (lgn-1).2^{lg\,n-1}) - lgn(2^0 + 2^1 + 2^2 + 2^3 + ... + 2^{lg\,n-1}) \\
&= nlgn + 2 - 2n - lgn(n-1) \\
&= 2 - 2n + lgn \\
Pr\{E_1 \cap E_2 \cap ... \cap E_n\} &= 2^{2-2n+lg\,n}
\end{aligned} \tag{7}$$

4. CONCLUSION

The last section deliberated the probability of perfect partitioning for *RANDOMIZED-QUICKSORT* procedure. Given the low probability of perfect partitioning the RANDOMIZED-QUICKSORT still manages to give a $\Theta(n\ lg\ n)$ expected runtime which is due to its good performance even in the scenarios of unbalanced partitioning. This makes Quicksort a widely used sorting algorithm in various computational tasks.

5. REFERENCES

[1] Hoare, C.A., 1962. Quicksort. The Computer Journal, 5(1), pp.10-16.

[2] Coremen, L., Rivest, Stein Introduction to Algorithm. PHI publication.