CouponCollector QuickSort RandomizedMedian

FullName: Christodoulides Kyriakos **AM:** 2016030025
PLH421 - Randomized Algorithms
University of Crete

March 26, 2021

Coupon Collector

There are n different types of coupons and the collector wishes to collect all n coupons. At each trial a coupon is chosen at random and each coupon is equally likely and the choices are independent. We search how much is the waiting time to collect all n coupons.
Let C_n denote the random variable defined to be the number of trials required to collect all n coupons. Let X_i, denote the number of trials in the i^{th} subinterval. So we have:

$$C_n = \sum_{i=0}^{n-1} X_i$$

The X_i are independent random variables with geometric distribution, so the propability of success is:

$$p_i = \frac{n-i}{n}, \qquad i = 0, 1 ... n-1$$

So because X is a random variables with geometric distribution, we know that (p is the probability of success) :
$E(X) = \frac{1}{p}$ so we have :

$$E(C_n) = \sum_{i=0}^{n-1} E(X_i) = \sum_{i=0}^{n-1} \frac{n}{n-i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n$$

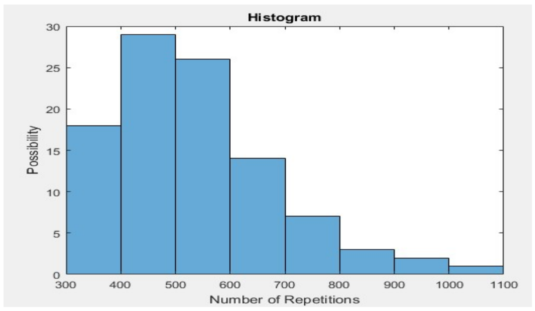
Here H_n is n^{th} Harmonic number. In calculus is proved that for large n H_n is:

$H_n = logn + \gamma + \frac{1}{2n} + O(\frac{1}{n^2})$ where $\gamma \cong 0.57721..$ is Euler constant, and $n \to \infty$

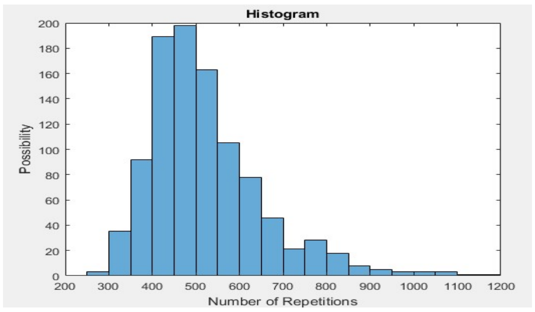
So the mean time we need to collect all n coupons is:

$$E(C_n) = nH(n) = nlogn + n\gamma + \frac{1}{2} + O(\frac{1}{n})$$

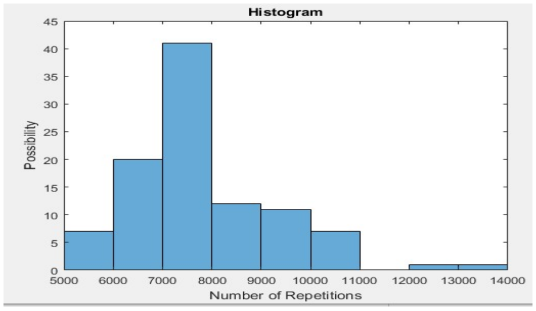
Simulations



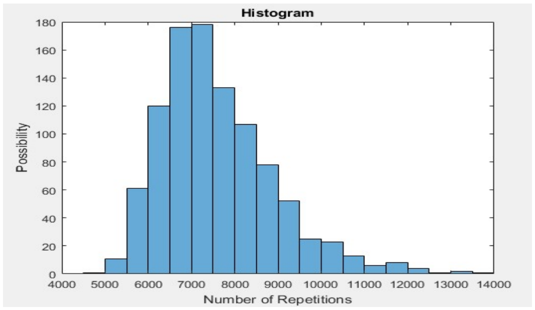
(a) n = 100, simulations = 100



(b) n = 100, simulations = 1000



(c) n = 1000, simulations = 100



(d) n = 1000, simulations = 1000

QuickSort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are some ways to peak the pivot, in my implementation,i always pick first element as pivot. The complexity of the algorithm is O(nlogn) and the solution can be solved using case 2 of Master Theorem. The worst case analysis, is O(n^2), and it happens when the pivot is always the smallest or the greatest element in the list. In my case (always pick first element as pivot) to happen something like that the uniform random generated list must be already sorted or reversed sorted. Because the possibility something like that to happen is almost zero, more specifically:

$$p = \frac{2}{(n-1)(n-2)...2*1} = \frac{2}{(n-1)!}$$

The average complexity is O(nlogn). So we count each assignation to compute the exact complexity of the algorithm.

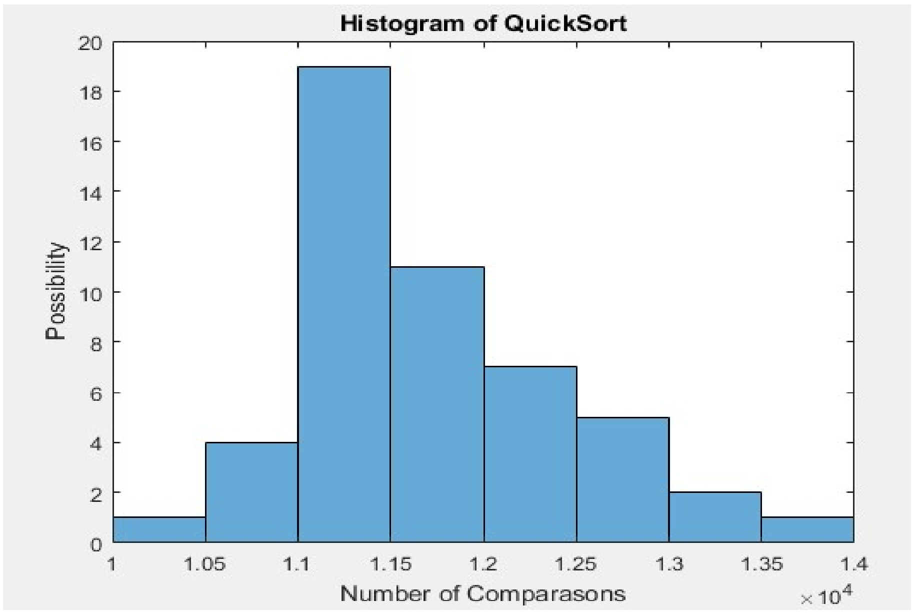


Figure 2: n=1000 simulations=50

We see that the average comparations are 11000-12000, and with n=1000, $nlog_2n = 9965.7$

Randomized Median

Here we found Median With High Probability, and the complexity is O(n). The is being described by this pseudocode. Note that the below algorithm works as long as the elements in S are distinct. If S is allowed to have repeat elements,then the algorithm must be slightly modified so that in line 5, the comparisons being done take into account a total order defined on the element. Like QuickSort, we count each assignation that the algorithm does to find the median.

```
   input  : List $S$ of $n$ integers, $n$ odd
   output: The median of $S$, or FAIL
 1 $R \leftarrow$ choose $n^{3/4}$ elements from $S$ (u.a.r. and with replacement);
 2 sort $R$;
 3 $\ell \leftarrow \left(\frac{n^{3/4}}{2} - \sqrt{n}\right)^{\text{th}}$ smallest element of $R$;
 4 $u \leftarrow \left(\frac{n^{3/4}}{2} + \sqrt{n}\right)^{\text{th}}$ smallest element of $R$;
 5 $C \leftarrow \{x \in S \mid \ell \le x \le u\}$,  $S_{<\ell} \leftarrow \{x \in S \mid x < \ell\}$,  $S_{>u} \leftarrow \{x \in S \mid x > u\}$;
 6 if $|S_{<\ell}| \ge \frac{n}{2}$ or $|S_{>u}| \ge \frac{n}{2}$ then
 7     output FAIL;
 8 if $|C| > 4n^{3/4}$ then
 9     output FAIL;
10 else
11     sort $C$;
12     output $\left(\frac{n+1}{2} - |S_{<\ell}|\right)^{\text{th}}$ smallest element of $C$;
```

Simulations

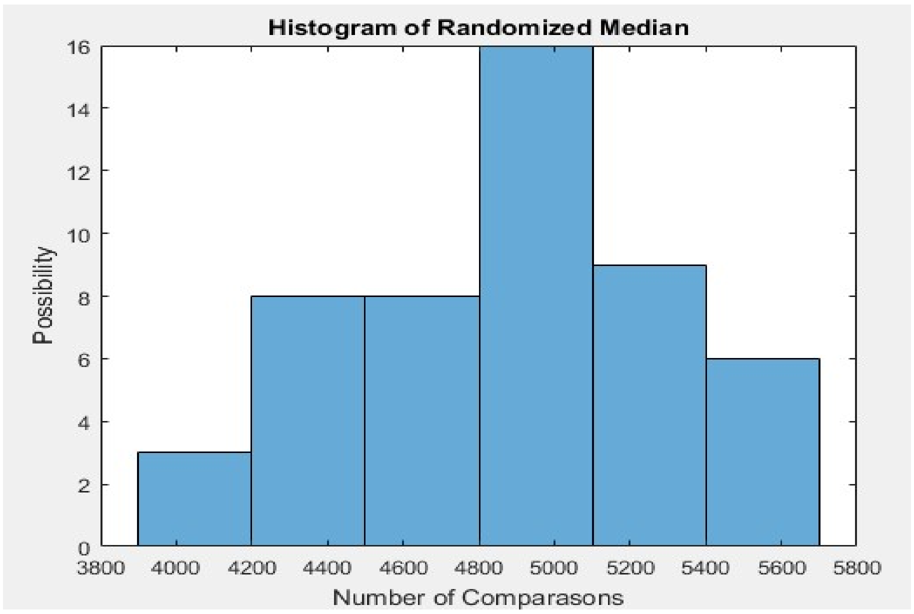


Figure 3: n=1000 simulations=50

Comparing this algorithm with QuickSort for n=200:200:10000 we have this results:

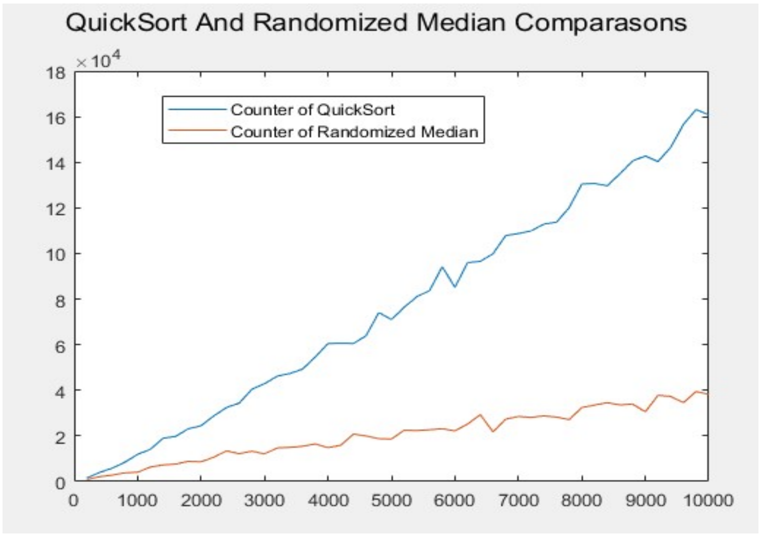


Figure 4: n=200:200=10000

The Randomized Median needs O(n) time to find Median, so it's obviously faster than QuickSort, but it doesn't sort the list. So for this problem the Best algorithm is Randomized Median. Note, we can use Randomized Median algorithm to set the pivot each time in QuickSort, that would made the QuickSort complexity to aprroach to it's best case scenario.