Assignment 1: Hash Functions and PAC Algorithms

Christopher Mertin/u1010077
cmertin@cs.utah.edu
January 24, 2017

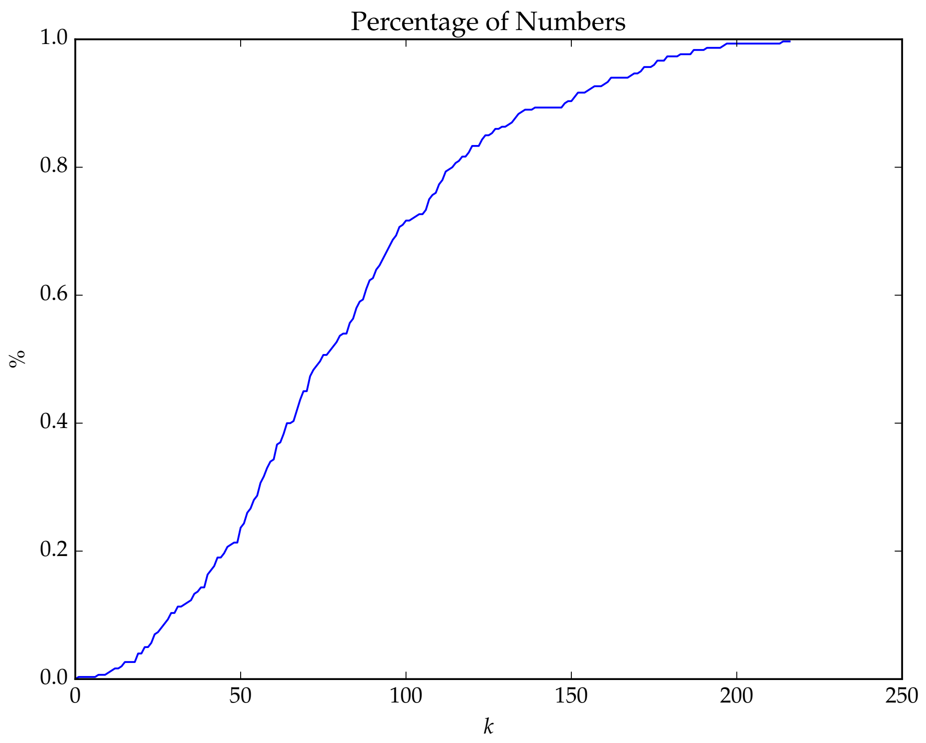
1 Birthday Paradox (30 points)

Consider a domain of size $n = 4000$.

A: (5 points) Generate random numbers in the domain $[n]$ until two have the same value. How many random trials did this take? We will use k to represent this value.

```
k = 48
```

B: (10 points) Repeat the experiment $m = 300$ times, and record for each time how many random trials this took. Plot this data as a *cumulative density plot* where the x-axis records the number of trials required k, and the y-axis records the fraction of experiments that succeeded (a collision) after k trials. The plot should show a curve that starts at a y value of 0, and increases as k increases, and eventually reaches a y value of 1.

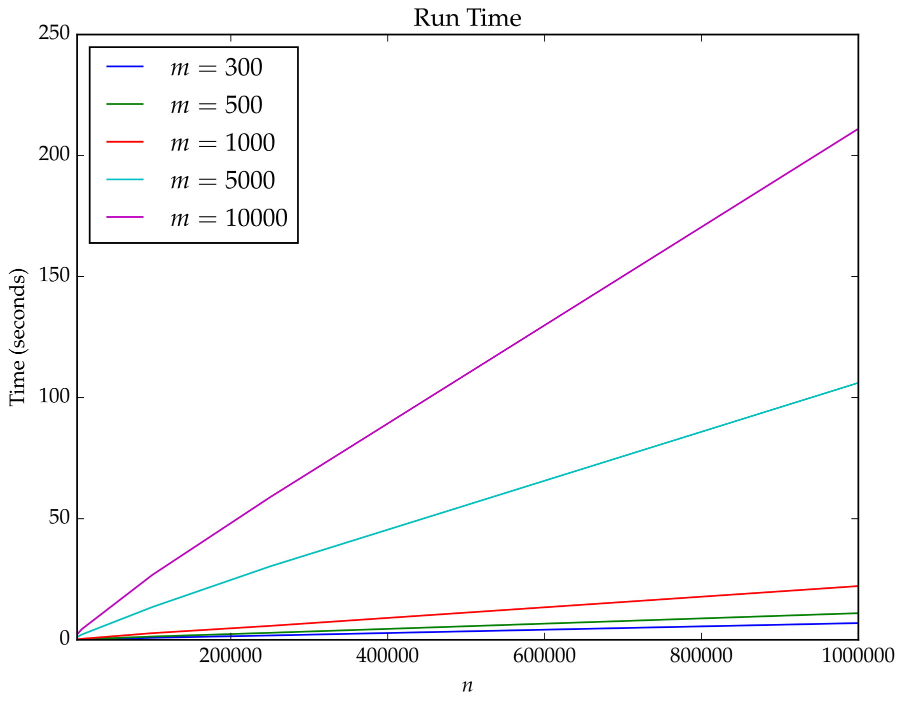


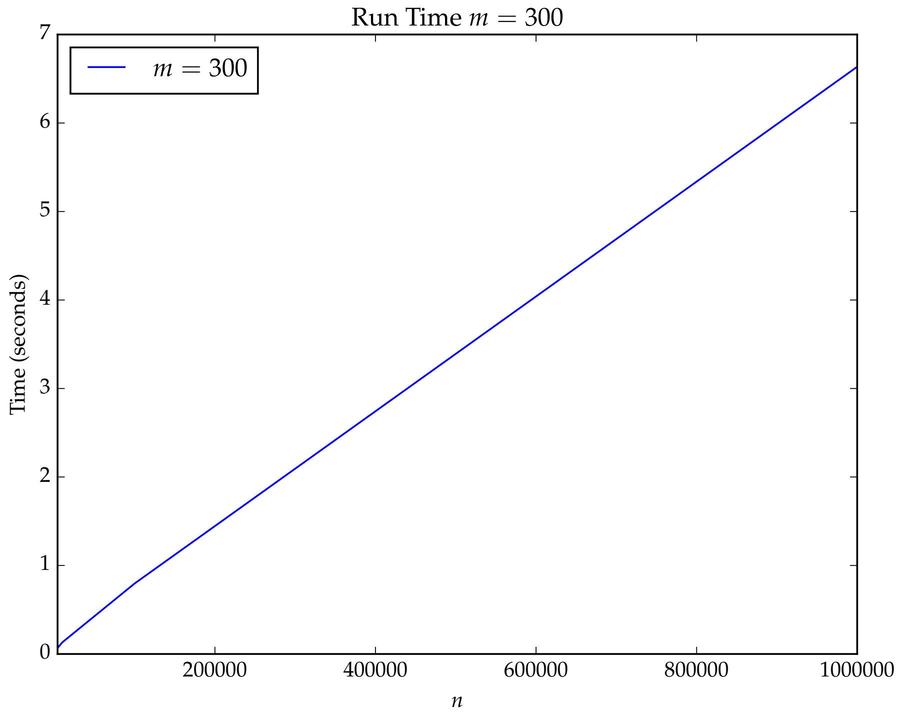
C: (5 points) Empirically estimate the expected number of k random trials in order to have a collision. That is, add up all values k, and divide by m.

```
k mean = 78.32
```

D: (10 points) Describe how you implemented this experiment and how long it took for $m = 300$ trials.

Show a plot of the run time as you gradually increase the parameters n and m. (For at least 3 fixed values of m between 300 and 10,000, plot the time as a function of n.) You should be able to reach values of $n =$ 1,000,000 and $m =$ 10,000.





To implement this code, an empty list was created where each non-matching number would be stored. It would simply append to the list every random number generated in the domain and append it to the list if it was not in the list already.

If it was in the list, the program would end as we would have reached the terminating condition outlined above. The length of the array that stored the numbers was the value of k for that iteration.

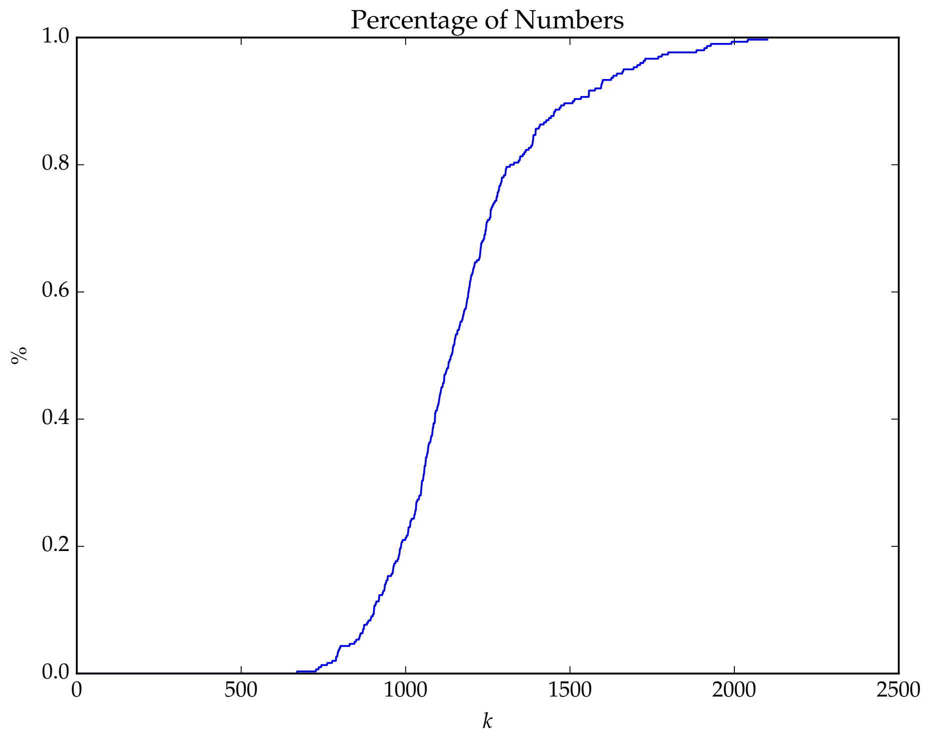
2 Coupon Collectors (30 points)

Consider a domain $[n]$ of size $n = 200$.

A: (5 points) Generate random numbers in the domain $[n]$ until every value $i \in [n]$ has had one random number equal to i. How many random trials did this take? We will use k to represent this value.

```
k = 1134
```

B: (10 points) Repeat step A for $m = 300$ times, and for each repetition record the value k of how many random trials we required to collect all values $i \in [n]$. Make a cumulative density plot as in 1.B.



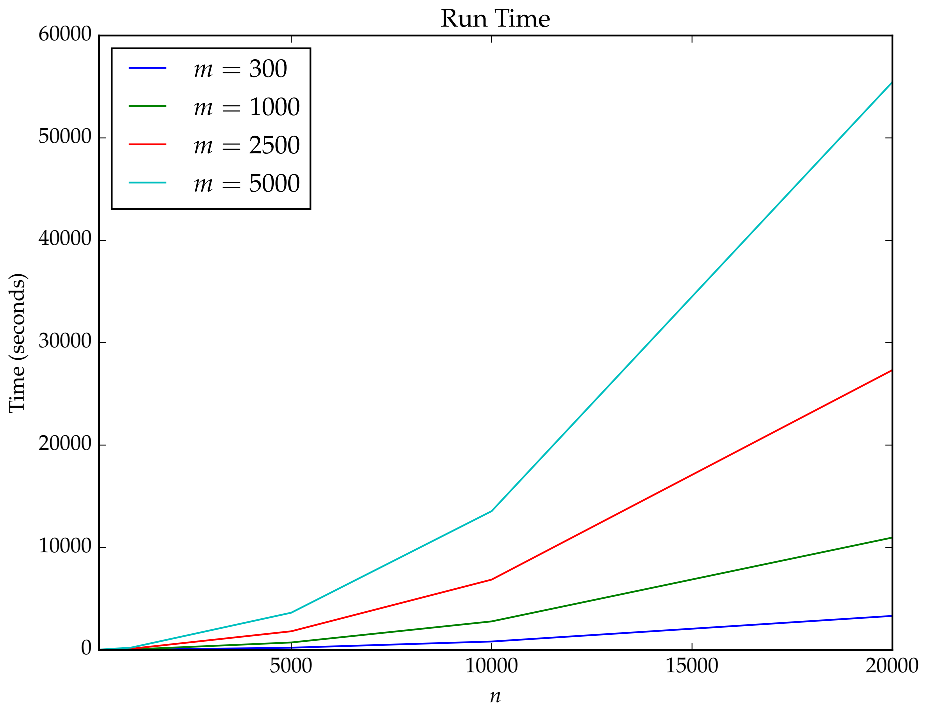
C: (5 points) Use the above results to calculate the empirical expected value of k.

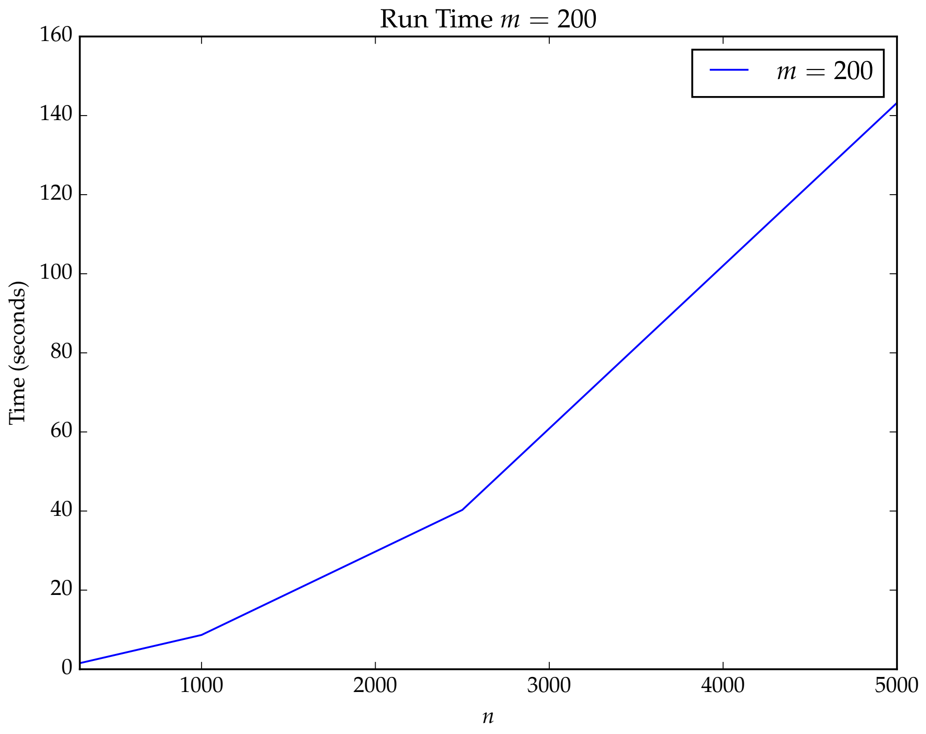
```
k mean = 1184.14
```

D: (10 points) Describe how you implemented this experiment and how long it took for $n = 200$ and $m = 300$ trials.

Show a plot of the run time as you gradually increase the parameters n and m. (For at least 3 fixed values of m between 300 and 5,000, plot the time as a function of n.) You should be able to reach $n = 20{,}000$ and $m = 5{,}000$.

In order to optimize efficienty, the `numpy` package was used. A `numpy` array was created at each instantiation of the Coupon Collector problem. The array had dimensions $n + 1$, and each time a random number was created, that index of the array was incremented. After this, the command `numpy.count_nonzero(array) == len(array)` was used as the termiantion sequence for when it was true. This sped up efficiency as the `numpy` commands are optimized. This would *only* be true when every number in the array had a non-zero value.


Run Time
$m = 300$
$m = 1000$
$m = 2500$
$m = 5000$
Time (seconds)
n
0
10000
20000
30000
40000
50000
60000
5000
10000
15000
20000


Run Time $m = 200$
$m = 200$
Time (seconds)
n
0
20
40
60
80
100
120
140
160
1000
2000
3000
4000
5000

3 Comparing Experiments to Analysis (24 points)

A: (12 points) Calculate analytically (using formulas from the notes in **L2**) the number of random trials needed so there is a collision with probability at least 0.5 when the domain size is $n = 4000$. There are a few formulas stated with varying degree of accuracy, you may use any of these – the more accurat formula, the more sure you may be that your experimental part is verified, or is not (and you need to fix something).
[Show your work, including describing which formula you used.]

How does this compare to your results from Q1.C?

The probability of having a collision on the k^{th} iteration in domain n is the product of each individual probability. Therefore, the probability of a collision happening on the k^{th} iteration is

$$p(k) = 1 - \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

Where, using the Taylor Series Expansion we get

$$p(k) \approx 1 - e^{-k(k-1)/2n}$$

From the Taylor Series expansion, we can substitute $n = 4000$ and solve for the required value of k such that the probability is greater than $1/2$.

$$\begin{aligned}\frac{1}{2} &\geq 1 - e^{-k(k-1)/2n} \\ \frac{1}{2} &\leq e^{-k(k-1)/2n} \\ \ln\left(\frac{1}{2}\right) &\leq \frac{-n(n-1)}{4000}\end{aligned}$$

By the Pythagorean Theorem, we get

$$\begin{aligned}0 &= n^2 + n - 4000 \ln\left(\frac{1}{2}\right) \\ n &= 52.16\end{aligned}$$

Our calculated average was `78.32`, which is relatively close to the actual result. As the number of trials are increased, our accuracy would also increase.

B: (12 points) Calculate analytically (using formulas from the notes in **L2**) the expected number of random trials before all elements are witnessed in a domain of size $n = 200$? Again, there are a few formulas you may use – the more accurate, the more confidence you might have in your experimental part.
[Show your work, including describing which formula you used.]

How does this compare to your results from Q2.C?

The expected number of random trials needed to generate all values in a domain of size n is defined by

$$\begin{aligned}T = nH_n &= n\left[\gamma + \ln(n) + \sigma(1/n)\right] \\ &= n(0.6 + \ln(n))\end{aligned}$$

For $n = 200$

$$\begin{aligned} T &= 200(0.6 + \ln(200)) \\ T &= 1179.66 \end{aligned}$$

The calculated trials in question 2 gave a mean of `1184.14` which is a really close calulation for only 300 trials. This is most likely due to the fact that there is not going to be a lot of variance in choosing random numbers until every number has a value, as shown in the figure for Question 2 B.

4 Random Numbers (16 points)

Consider when the only random function you have is one that choses a bit at random. In particular `rand-bit()` returns `0` or `1` at uniformly random.

A: (6 points) How can you use this to create a random integer number between 1 and $n = 1024$?

To create 1024, it requires 10 bits. So you can populate the values of the bits by just calling it n times and assigning the bit value to the current bit.

B: (5 points) Describe a Las Vegas randomized algorithm ("Las Vegas" randomized algorithm means: it may run forever, but you can bound how long you expect it to take, and when it finishes you know it is done and correct) for generating a random number in $[n]$ when $n = 1000$.

You need $\lceil \log_2(n) \rceil$ bits, and consequently that many function calls. For $n = 1000$, we'd need 10 bits. What we would have to do, is randomly assign the bits using the `rand-bit()` function, and then check to see if the bit string has a value less than n. If it does, return that number, if not, rerun it.

C: (5 points) Describe how many `rand-bit()` calls does the above procedure (extending your algorithm in **4B**) take as a function of n (say I were to increase the value n, how does the number of calls to `rand-bit()` change)?

[Keep in mind that in many settings generating a random bit is much more expensive that a standard CPU operation (like an addition, if statement, or a memory reference), so it is critical to minimize them.]

If we have n_1 as the first range, and n_2 as the second range, the difference in the number of calls to `rand-bit()` would be approximately $\mathcal{O}\left(\log_2(n_2) - \log_2(n_1)\right)$.