Asmt 1: Hash Functions and PAC Algorithms

Turn in (**a pdf**) through Canvas by 2:45pm:
Wednesday, January 25

Overview

In this assignment you will experiment with random variation over discrete events.

It will be very helpful to use the analytical results and the experimental results to help verify the other is correct. If they do not align, you are probably doing something wrong (this is a very powerful and important thing to do whenever working with real data).

As usual, it is highly recommended that you use LaTeX for this assignment. If you do not, you may lose points if your assignment is difficult to read or hard to follow. Find a sample form in this directory: `http://www.cs.utah.edu/~jeffp/teaching/latex/`

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.

It took me 130 trials to get a collision. I did not set the seed for the random number generator and so I got a different value for each run. The value of 130 was for one of the runs.

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.

The cumulative density plot is shown in figure 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.

Empirical estimate for the expected number of random trials till collision is 85.9466.

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

For each iteration of the experiment where it was run m times, the number of trials till collision was recorded. The trials till collision were put into an array and the array was sorted. For each entry in the array with index x, the corresponding fraction of experiments that succeeded in having a collision was computed as $\frac{x}{m}$.

For 300 trials and a domain size of 4000, it took 29 ms to run.

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

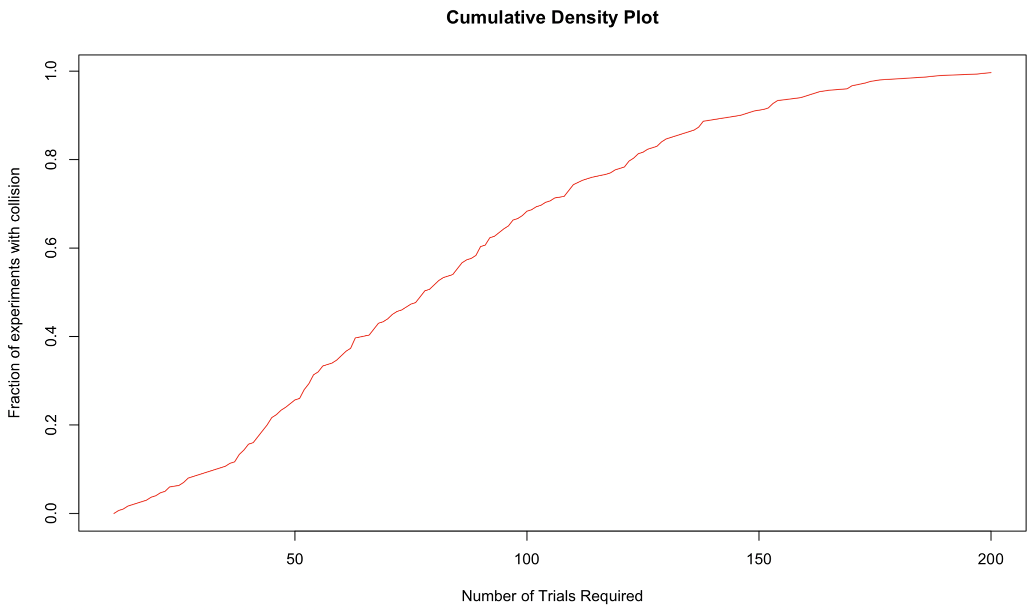

Figure 1: Cumulative density plot for trials till collision for experiment repeated 300 times for a domain of size 4000

$n = 1{,}000{,}000$ and $m = 10{,}000$.

The run time plot is shown in figure 2.

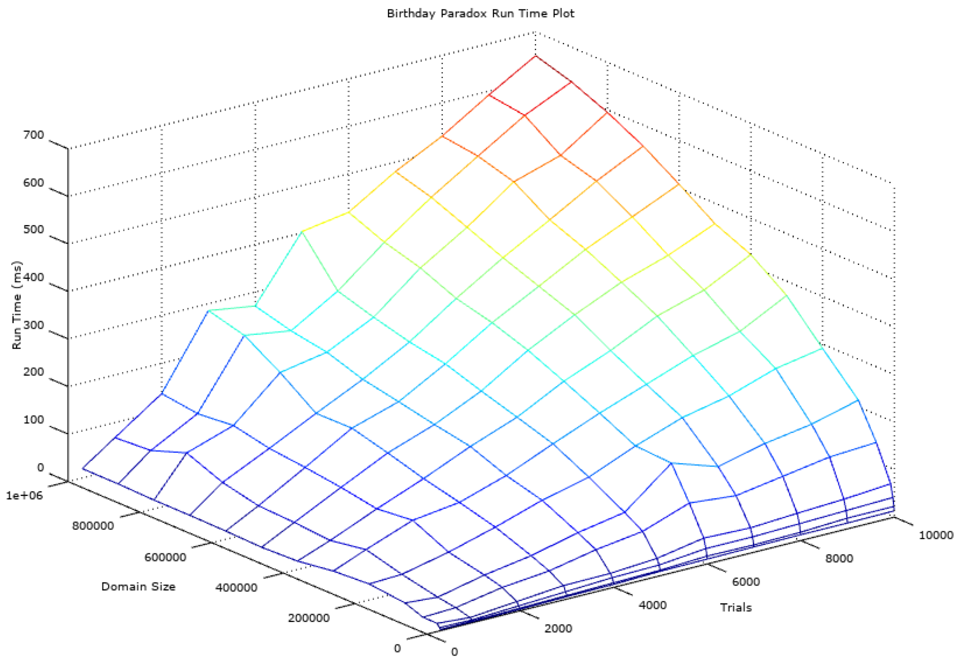

Figure 2: Birthday Paradox run time plot

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.

It took 1434 random trials for generating all numbers up to 200.

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.

The cumulative density plot is shown in figure 3.

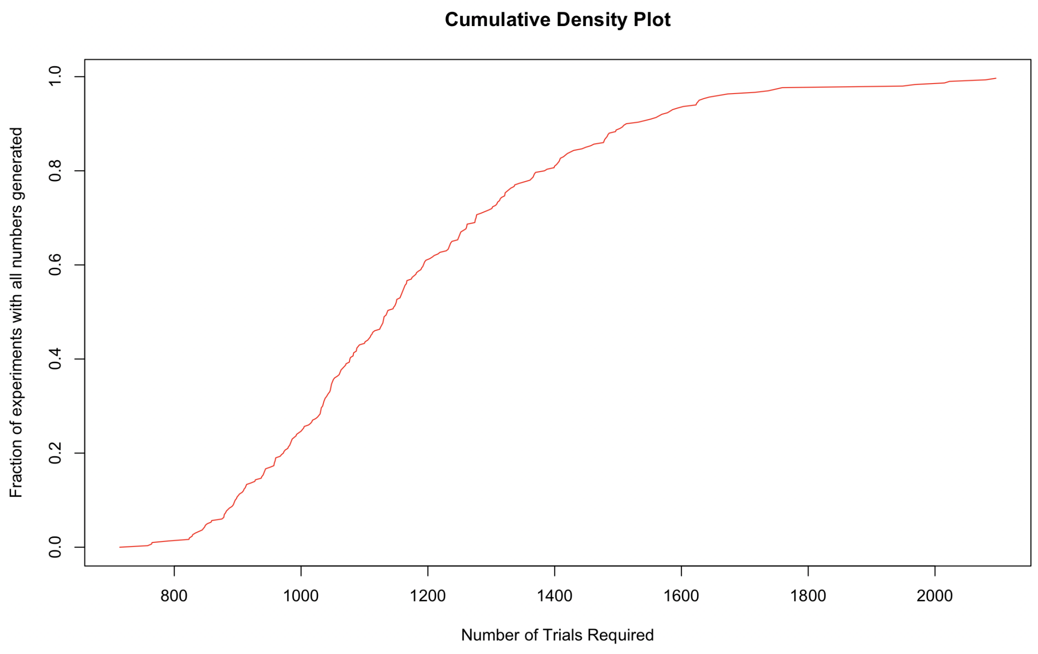

Figure 3: Cumulative density plot for trials till all numbers in domain are generated for experiment repeated 300 times for a domain of size 4000

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

Empirical estimate for the expected number of random trials till all values are generated is 1191.0276.

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

For each iteration of the experiment where it was run m times, the number of trials till every number in the domain was generated, was recorded. The numbers of trials were put into an array and the array was sorted. For each entry in the array with index x, the corresponding fraction of experiments that succeeded in generating every number in the domain was computed as $\frac{x}{m}$.

For 300 trials and domain size of 200, it took 40 ms to run.

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 $m = 5{,}000$.

The run time plot is shown in figure 4.

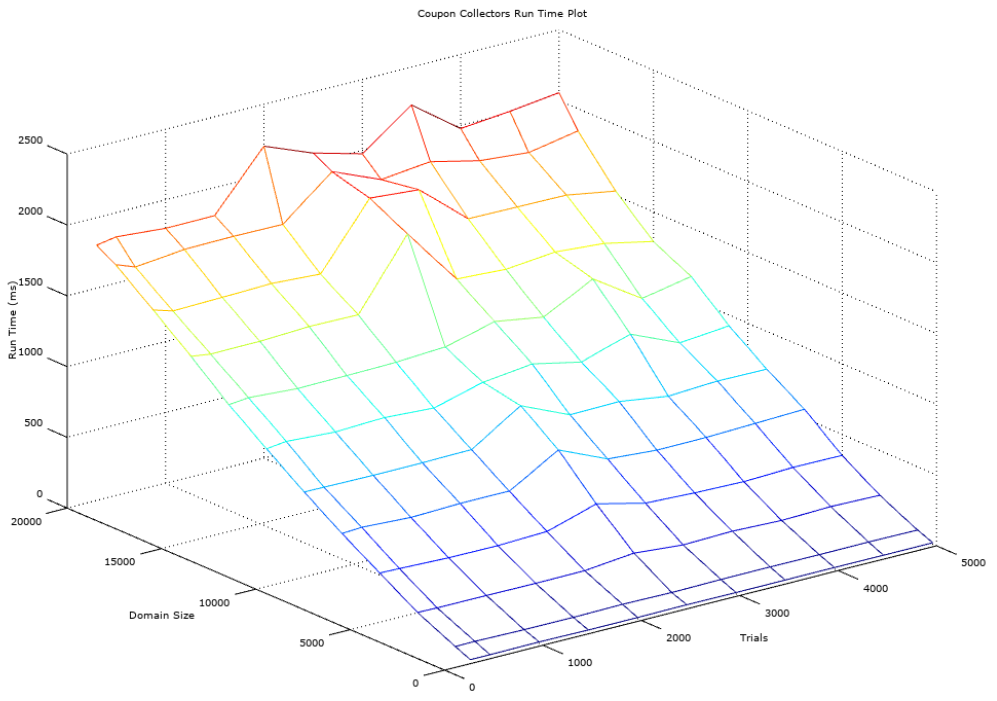

Figure 4: Coupon Collectors run time plot

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 accurate 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.]*

The probability of a collision after k trials is given by

$$\begin{aligned}\mathbf{Pr}(collision) &= 1 - \mathbf{Pr}(no\,collision)\\ &= 1 - \prod_{i=1}^{k-1} \frac{n-i}{n}\end{aligned}$$

By trial and error and running the computation in a program, I found out that for a domain size of $n = 4000$, after 74 trials, the probability of a collision is 0.4931 and after 75 trials the probability is 0.5025.

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

In Q1.C the empirical result for expected number of trials till a collision was 86 compared to the analytical result of 75 trials for a collision probability of at least 0.5.

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.]

The expected number of trials for seeing all n elements is given by

$$T = n \sum_{i=1}^{n} \frac{1}{i}$$

For a domain size of $n = 200$, this was computed to be 1175.

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

In Q2.C the empirical result for expected number of trials till for seeing all elements was 1192 compared to the analytical result of 1175 trials.

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 $n = 1024$?

We can create an array of 1024 booleans and each entry in the array can be set to $True$ or $False$ based on the value returned by `rand-bit()`. This array can be considered to be the binary representation of the random number that needs to be generated. Algorithm 1 is shown below that uses this technique for generating a random number when the upper limit is a power of 2.

Algorithm 1 Random Number Generation Algorithm

```
1:  procedure RANDOM NUMBER GENERATION($upperLimit$)
2:      $exponent = \log_2(upperLimit)$
3:      $randomNumberArray$ = $boolean[exponent]$
4:      for $arrayIndex\ in\ range(1\ to\ exponent)$ do
5:          if rand-bit() $== 0$ then
6:              $randomNumberArray[arrayIndex] = False$
7:          else
8:              $randomNumberArray\ [arrayIndex]$ = $True$
9:      $randomNumber = 1$
10:     for $arrayIndex\ in\ range(1\ to\ exponent)$ do
11:         if $randomNumberArray[arrayIndex] == True$ then
12:             $randomNumber = randomNumber + 2^{arrayIndex-1}$
        return $randomNumber$
```

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 $n = 1000$.

Algorithm 2 shows a Las Vegas algorithm for generating a random number. When the upper limit for the random number is not a power of 2, as in this case, the binary representation of the random number has a size such that the maximum number that can be generated, is larger than the upper limit. So in case that the random number generated is larger than the upper limit, the process is repeated till the generated random number is within the upper limit. Due to this reason, there is the possibility that the process may run forever.

In each run, the probability of generating a number within the upper limit of 1000 is $\frac{1000}{1024}$ and the probability of failure is $\frac{24}{1024}$. So the $while$ loop in algorithm 2 is expected to have $\lceil \frac{1024}{1000} \rceil = 2$ iterations.

Algorithm 2 Las Vegas Random Number Generation Algorithm

```
1: procedure LAS VEGAS RANDOM NUMBER GENERATION($upperLimit$)
2:     $exponent = \lceil \log_2(upperLimit) \rceil$
3:     $randomNumberGenerated = False$
4:     while $not\, randomNumberGenerated$ do
5:         $randomNumberArray$ = $boolean[exponent]$
6:         for $arrayIndex\, in\, range(1\, to\, exponent)$ do
7:             if rand-bit() $== 0$ then
8:                 $randomNumberArray[arrayIndex] = False$
9:             else
10:                $randomNumberArray\ [arrayIndex]$ = $True$
11:        $randomNumber = 1$
12:        for $arrayIndex\, in\, range(1\, to\, exponent)$ do
13:            if $randomNumberArray[arrayIndex] == True$ then
14:                $randomNumber = randomNumber + 2^{arrayIndex-1}$
15:        if $randomNumber \leq upperLimit$ then
16:            $randomNumberGenerated = True$
    return $randomNumber$
```

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.]

In the above case where $n = 1000$, `rand-bit()` will be called 10 times per iteration of the $while$ loop. The $while$ loop is expected to have 2 iterations as described above. Hence we can expect 20 calls to be made to `rand-bit()`. This expected number of calls will not change till n is increased till 1023. When $n = 1024$, the $while$ loop will execute only once as the random number generated will be at most 1024.

When n is 1025, the binary number used for random number generation will have 11 bits and the largest number that can be generated is 2048. The probability of success becomes $\frac{1025}{2048}$ and that of failure becomes $\frac{1023}{2048}$. We can expect to call `rand-bit()` $11 \times 2 = 22$ times.

The number of calls N to `rand-bit()` as a function of n will be

$$N = \lceil \log_2 n \rceil \times 2$$

5 BONUS : PAC Bounds (2 points)

Consider a domain size n and let k be the number of random trials run, where each trial obtains each value $i \in [n]$ with probability $1/n$. Let f_i denote the number of trials that have value i. Note that for each $i \in [n]$ we have $\mathbf{E}[f_i] = k/n$. Let $\mu = \max_{i \in [n]} f_i/k$.

Consider some parameter $\delta \in (0, 1)$. As a function of parameter δ, how large does k need to be for $\mathbf{Pr}[|\mu - 1/n| \geq 0.01] \leq \delta$? That is, how large does k need to be for *all* counts to be within 1% of the average with probability δ? *(Fine print: you don't need to calculate this exactly, but describe a bound as a function of δ for the value k which satisfies PAC property. Notes here should help:* `http://www.cs.utah.edu/~jeffp/teaching/cs5140/L2+Chern-Hoeff.pdf`*)*

How does this change if we want $\mathbf{Pr}[|\mu - 1/n| \geq 0.001] \leq \delta$ (for 0.1% accuracy)?

[Make sure to show your work.]