Assignment #1 (Data Mining)

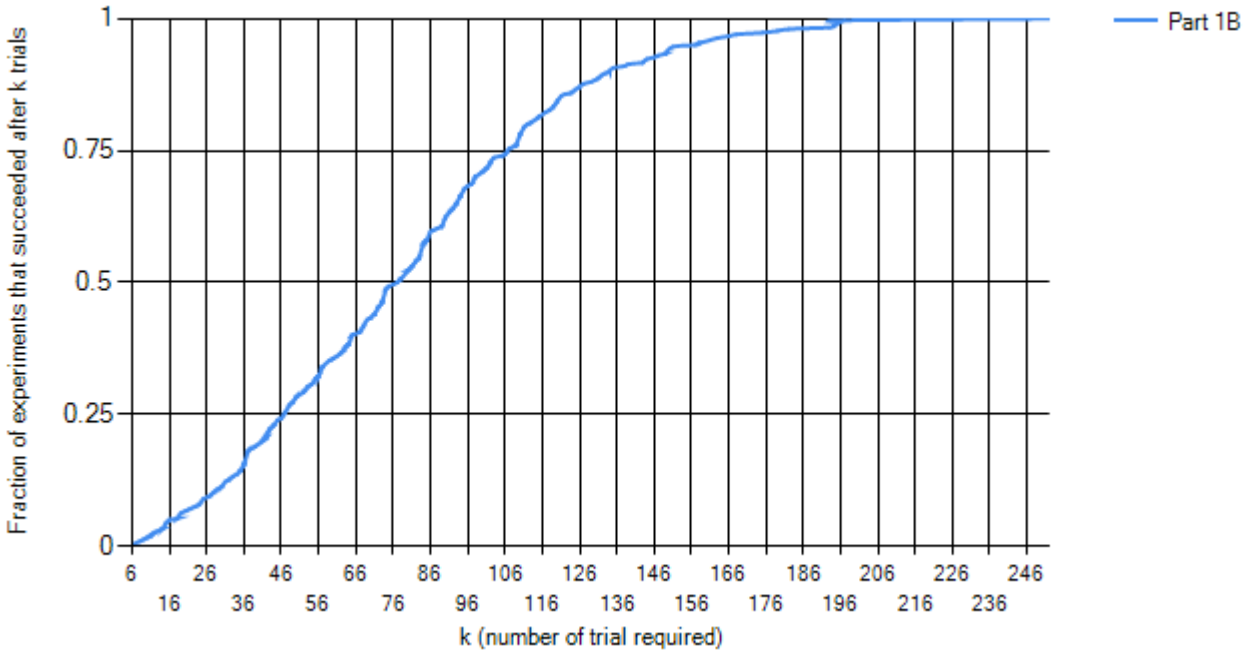
Name: Spencer Fronberg
UID: u0766439

January 23, 2018

1 Birthday Paradox

(A) It took a total of **52** random trails for me to get two that have the same value in the domain of size n = 4000.

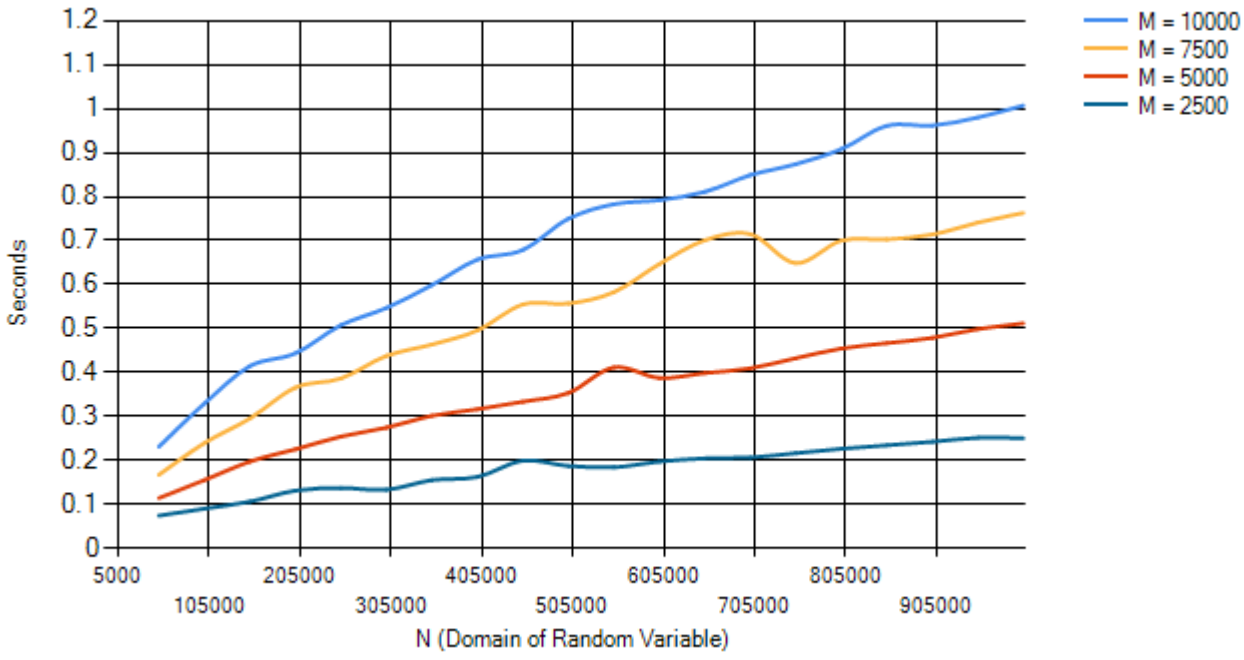
(B) The following is the cumulative density plot for when m = 300:



(C) The empirical expected value of k is: **79.43**

(D) For 300 trials, it took a total of just **2 Milliseconds**.
I coded this entire project in C#. First off, for me to determine if two random numbers generate the same value, I initially generate a random number, add it to a **HashSet** of random numbers if it does not already exist. If that number already exists in the

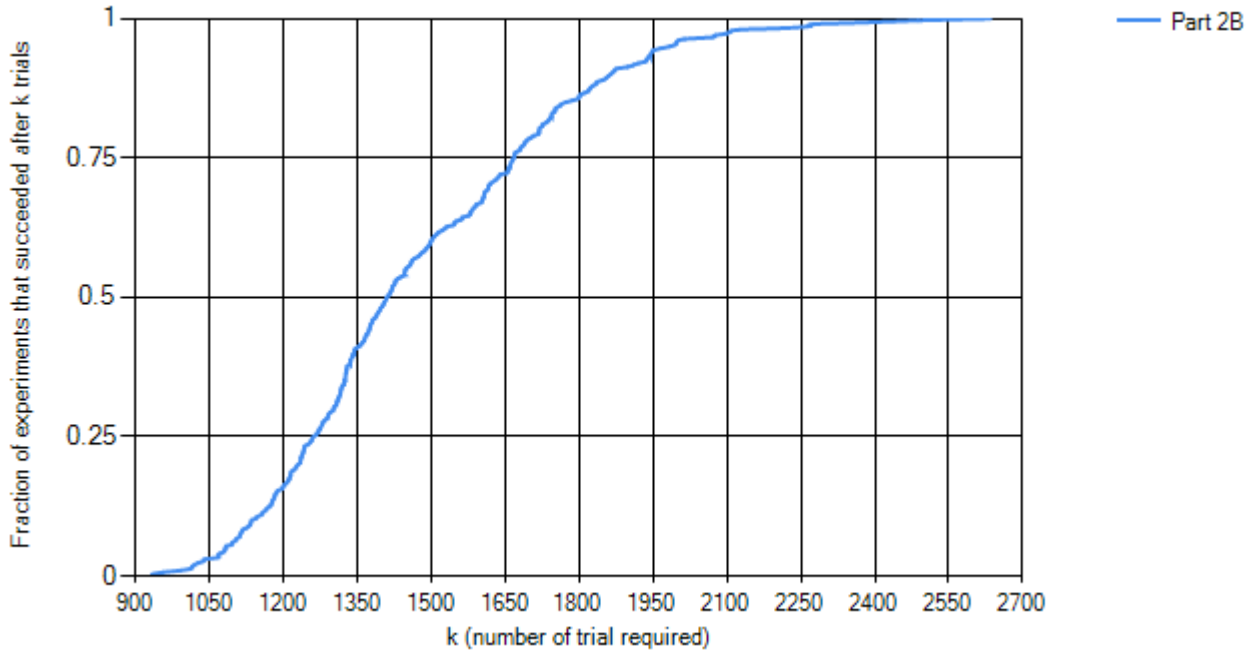
HashSet, then there is a collision and I record the total amount times it took to get a collision just by keeping a counter in the loop. For me to determine the cumulative density plot, first I calculated how many times it would take to get a collision for each of the 300 times (m) and stored 300 different numbers in a list where each number (k) was the number of times it took to get a collision. This list stored all the x values of each point in the graph for part B. For each specific k in the list, I calculate how many k's are less than specific k (x). I then divide x by the total number of k's which in this case is 300. The result of this value was my y value for each point in the graph. I then took all the points and graphed them as a curve.
The following is the plot for the runtime as n and m gradually increase:



2 Coupon Collectors

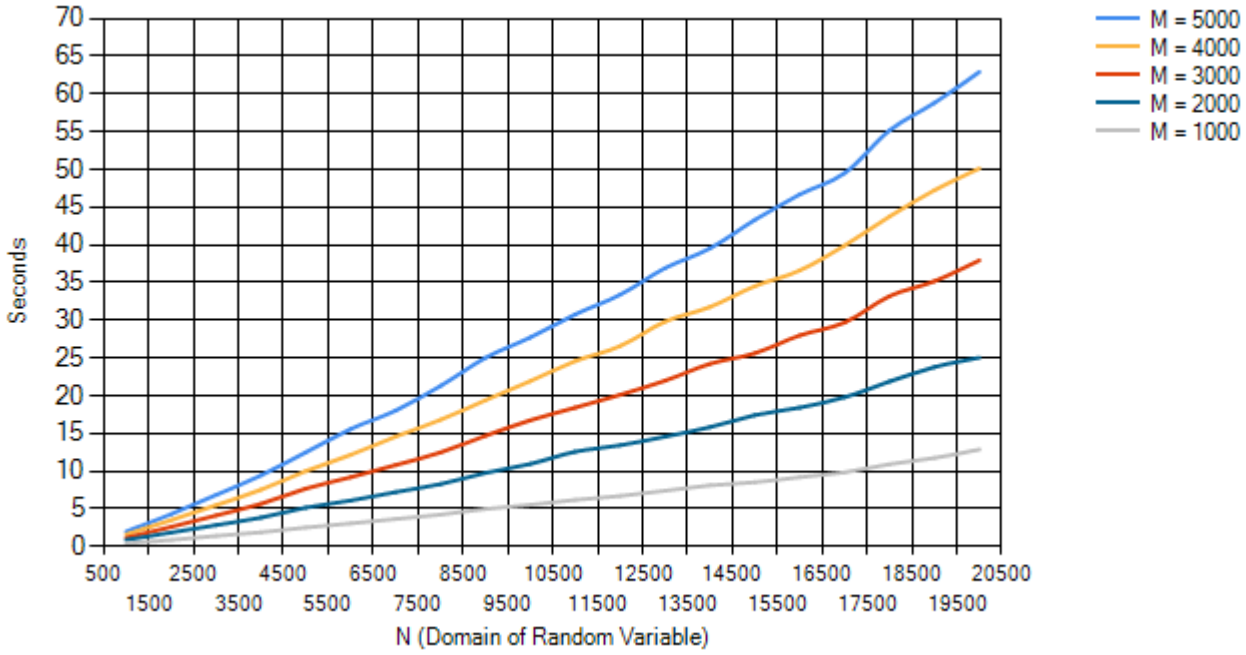
(A) It took a total of **1194** random trails for me to generate random numbers in the domain $[n]$ until every value i $\in [n]$ has at least one random number equal to i.

(B) The following is the cumulative density plot for when m = 300:



(C) The empirical expected value of k is: $\mathbf{1476.43\overline{6}}$

(D) For n = 250 and m = 300 trials, it took a total of 25 Milliseconds.
For this section, I did it a lot similar than the Birthday Paradox. I generate a random number, add it to the **HashSet** of random numbers if it does not already exist (because in a **HashSet** you cannot have duplicates). For me to keep track of the total amount of random trials, I just kept a counter in the loop and once the **HashSet**'s count was equal to n, then all the numbers in the domain had been generated because as I said before that a **HashSet** cannot have duplicate values. I generated the cumulative density plot the same way as in the Birthday Paradox problem.
The following is the plot for the runtime as n and m gradually increase:



3 Comparing Experiments to Analysis

(A) For this problem, I used the following formula:

$$P(Collision) = 1 - \prod_{i=1}^{k-1}(\frac{n-i}{n})$$

$$P(Collision) = 0.5, n = 4000$$

$$0.5 = 1 - \prod_{i=1}^{k-1}(\frac{4000-i}{4000})$$

I then wrote a small function in C# that calculated the probability with the k that you input. When k is 74 i get 0.493 and when k is 75 I get 0.502. So the number of random trials needed so there is a collision with probability at least 0.5 when the domain size is n = 4000 is: **k = 75**

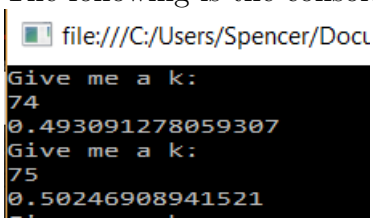
The following picture is my code that calculates the probability:

```
while (true)
{
    //Console.WriteLine("Give me an n:\t");
    double n = 4000; // Console.Read();
    Console.WriteLine("Give me a k:\t");
    double k = Convert.ToDouble(Console.ReadLine());

    double probability = 1;
    for (int i = 1; i < k; i++)
    {
        probability = probability * ((n - i) / n);
    }
    probability = 1 - probability;

    Console.WriteLine(probability);
}
```

The following is the console while I input my values:



(B) I will use the following formula for this problem:

$$k = n(\gamma + ln(n))$$
$$n = 250, \gamma = 0.577$$
$$k = 250(0.577 + ln(250))$$
$$k = 1524.615$$

So the expected number of random trials before all elements are witnessed in a domain of size n = 250 is **1525**

4 Random Numbers

(A) I would generate 10 random bits where 0000000000 would be represented by 1 and 1111111111 would be represented by 1024. I would just take the binary value of what the ten bits would be equal to and add one to it to get a number between 1 and 1024. If it was just the exact binary value (not adding 1 to it), it would be a number between 0 and 1023. That is why we have to add one to the binary value.

(B) This algorithm is very simple. We will have a function (*Las Vegas*) and we will figure out how many bits we need by the following:
ceiling($log_2(n)$), where n = 1000.
This will then make it so we will need to make 10 calls to rand-bit(). We will then

determine the binary value of the ten bits generated, add one to it, and store it as x. Now if x is less than 1001, then we will return x, but if x is greater than 1000, we will recursively recall *Las Vegas*. This makes it so it will always return a number between 1 and 1000. The following is the pseudocode for this algorithm:

Algorithm 1 Las Vegas

```
procedure LAS VEGAS(n = 1000)
    List<bit> bits = new List<bit>
    int p = Ceiling(log2(n))          ▷ In this case, p = 10 when n = 1000.
    for (i = 0, i < p, i++) do
        bits.Add(rand-bit())
    int x = CalculateIntegerValue(bits) + 1
    if x > n then
        return LAS VEGAS(n)
    else
        return x
```

(C) First off, if the first call the Algorithm above (Las Vegas) generates a number within the domain (n), then there will only be $\lceil log_2(n) \rceil$ calls to rand-bit() (this is for any value of n). The probability that it will fail is:

$$\frac{2^{\lceil log_2 n \rceil} - n}{2^{\lceil log_2 n \rceil}}$$

The probability that it will succeed is:

$$\frac{n}{2^{\lceil log_2 n \rceil}}$$

So to determine how many times we need to call rand-bit(), I will use the following notation where x is the number times my Las Vegas algorithm will be called and y is the number of times rand-bit() will be called with just one iteration of my Las Vegas algorithm:

$$x * y$$

x = $\frac{1}{P_{success}}$, and y = $\lceil log_2 n \rceil$
The reason y = $\lceil log_2 n \rceil$ is because we need to figure out how many times we will call rand-bit() with just one iteration of my Las Vegas algorithm. So by us taking the log base 2 of n, we are figuring out how many bits we need to support that domain, and we have to take the ceiling of it because if we don't we will not support our entire domain. So:

$$\frac{1}{P_{success}} * \lceil log_2 n \rceil$$

Above I show what the probability of success is, so:

$$\frac{2^{\lceil log_2 n \rceil}}{n} * \lceil log_2 n \rceil$$

With this I determined that no matter what, my algorithm will never be called more than twice no matter the n value. Below shows why it will never be called more than twice.
Worst Cases:
n = 2049:

$$\frac{2^{\lceil log_2 2049 \rceil}}{2049} = 1.9990$$

n = 4097:

$$\frac{2^{\lceil log_2 4097 \rceil}}{4097} = 1.9995$$

Best Cases:
n = 2048

$$\frac{2^{\lceil log_2 2048 \rceil}}{2048} = 1$$

n = 4097:

$$\frac{2^{\lceil log_2 4097 \rceil}}{4097} = 1$$

We can see that the worse case is always one number above a power of two number and the best case is always a number that is a power of two. So the number of times we expect rand-bit() to be called is no greater than the following:

$$O(2\lceil log_2 n \rceil)$$

5 Bonus: PAC Bounds

I will use the following formula:

$$Pr[|\mu - 1/n| \geq \epsilon] \leq 2exp(\frac{-2r\epsilon^2}{\Delta^2})$$

So I can say the following because we want to bound it so the probability of failure does not get greater than 0.02 (so it is at most 0.02%):

$$2exp(\frac{-2r\epsilon^2}{\Delta^2}) \leq 0.02$$

We can now set r = k and $\Delta = n - 1$

$$2exp(\frac{-2k\epsilon^2}{(n-1)^2}) \leq 0.02$$

$$exp(\frac{-2k\epsilon^2}{(n-1)^2}) \leq 0.01$$

$$ln(exp(\frac{-2k\epsilon^2}{(n-1)^2})) \leq ln(0.01)$$

$$\frac{-2k\epsilon^2}{(n-1)^2} \leq ln(0.01)$$

$$-2k\epsilon^2 \leq ln(0.01)(n-1)^2$$

$$k \leq -\frac{ln(0.01)(n-1)^2}{2\epsilon^2}$$

Which is about:

$$k \leq \frac{2.303(n-1)^2}{\epsilon^2}$$

Now for us to show that the probability of failure is only 0.002%. We can start with the first formula on this page and just replace 0.02 with 0.002:

$$2exp(\frac{-2k\epsilon^2}{(n-1)^2}) \leq 0.002$$

$$exp(\frac{-2k\epsilon^2}{(n-1)^2}) \leq 0.001$$

$$ln(exp(\frac{-2k\epsilon^2}{(n-1)^2})) \leq ln(0.001)$$

$$\frac{-2k\epsilon^2}{(n-1)^2} \leq ln(0.001)$$

$$-2k\epsilon^2 \leq ln(0.001)(n-1)^2$$

$$k \leq -\frac{ln(0.001)(n-1)^2}{2\epsilon^2}$$

Which is about:

$$k \leq \frac{3.454(n-1)^2}{\epsilon^2}$$