HW Problem 1 Writeup
Homework partners: Sam Drake, Serena Chan
Grader Instructions: Each part a-f below contains the code to run each part in python 2.7 with the partx() functions in the problem1.py file provide in the .zip file. Note, the python file contains some functions that are used in each of the parts below, but are only in the python file, so make sure you run these functions after starting with the command python2.7 -i problem1.py, otherwise these functions do not work on their own. Each part also contains all the relevant plots and comments; the plots will also be in the zip file.

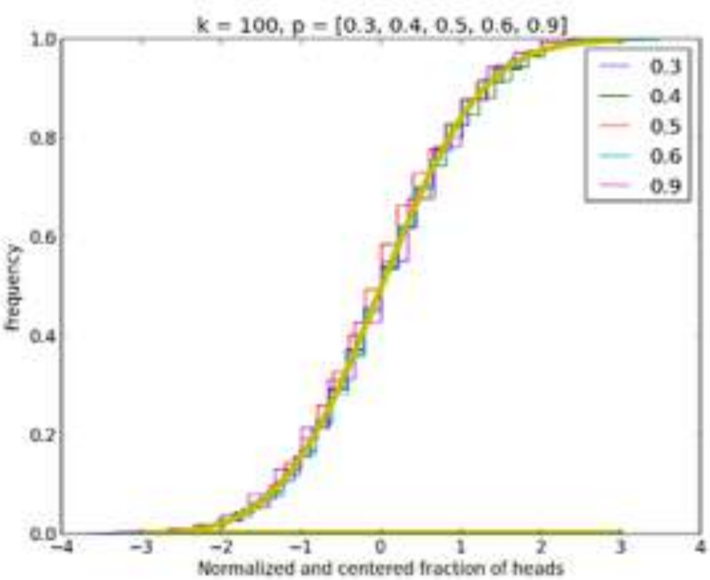
Helper Functions:import random

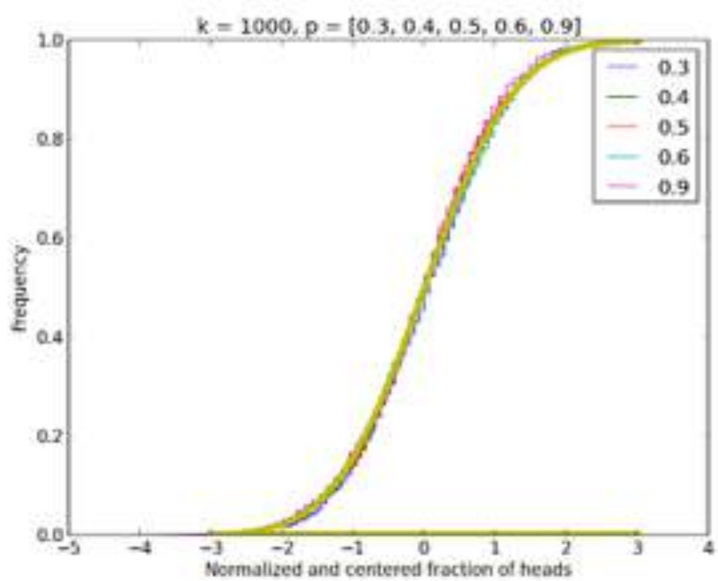
```
import numpy as np
import scipy.stats as stats
from scipy import integrate
import math
import matplotlib as mpl, matplotlib.pyplot as plot

def biasedCoin (p):
	# Creates a biased coin with p(Head) = p
	# Returns true if heads and false otherwise

	# Should check this!
	# assert p >= 0 and p <= 1
	return random.random() <= p

def runTrial (p, k):
	# Runs a trial of k tosses of a biased coin (w.p. p of heads)
	# and returns number of heads
	return sum([biasedCoin(p) for _ in xrange(k)])

def runManyTrials (p, k, m):
	# Runs m trials of k tosses of a biased coin (w.p. p of heads)
	# and returns all the numbers of heads
	return [runTrial(p, k) for _ in xrange(m)]

def calculateQuartileGap(results):
	# Calculates the quartile
	results.sort()
	n = len(results)
	q1 = int(round(0.25*n))
	q3 = int(round(0.75*n))
	return results[q3]-results[q1]

def linspace(a,b,n):
	# Returns n numbers evenly spaced between a and b, inclusive
	return [(a+(b-a)*i*1.0/(n-1)) for i in xrange(n)]
```

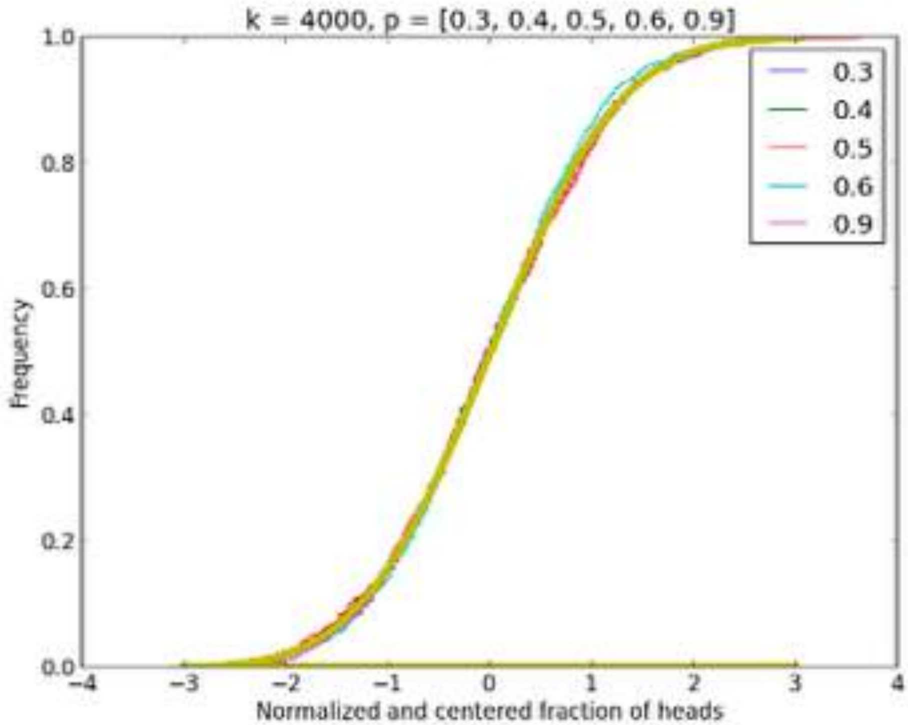
a) CODE:

```
def parta(pranges=[0.3,0.4,0.5,0.6,0.9], kranges = [100,1000,4000], m=1000,
show_indiv=False):
```

```
    # Q2 part (k)
    print('Question 2 part (k):')
    for k in kranges:
        plot.clf()
        results = {}
        print ('Number of trials k = %i'%k)
        for p in pranges:
            print ('Probability of head p = %.1f'%p)
            std = math.sqrt(p*(1-p))
            results[p] = [(Sk -k*p)/(math.sqrt(k)*std) for Sk in runManyTrials(p, k, m)]
            results[p].sort()
            plot.plot(results[p],linspace(0,1,m),label=str(p))
            x_values = np.arange(-3.0,3.1,.1)
            y_values = list()
            for i in x_values:
                y_values.append(integral(i))
            plot.plot(x_values,y_values,linewidth=4, color='y')
        plot.legend()
        plot.ylabel('Frequency')
        plot.xlabel('Normalized and centered fraction of heads')
        plot.title('k = %i, p = %s'%(k,str(pranges)))
        plot.show()

def integrand(x):
    return ((1/math.sqrt(2*math.pi))*math.exp(-1*(math.pow(x,2)/2)))

def integral(d):
    return integrate.quad(lambda x: integrand(x),-np.inf,d)
```





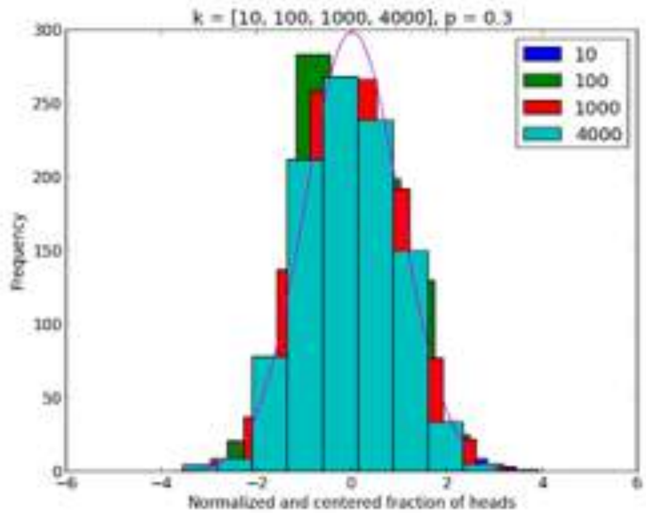


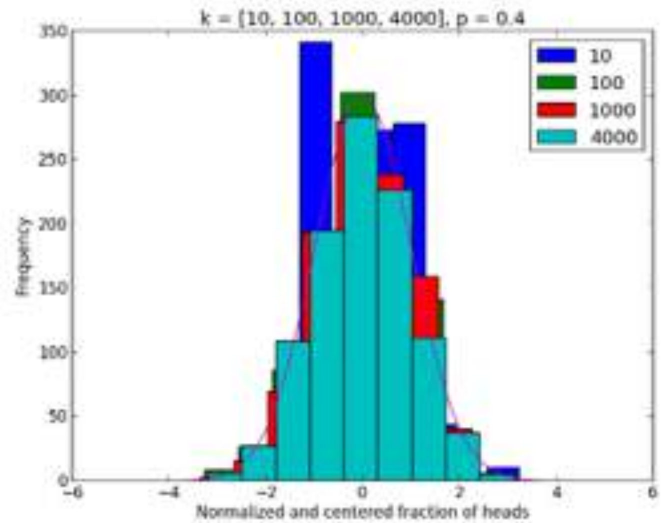
For this part, the function I plot was the integral as stated in the problem question, which i plotted against continuous values of d, since the x is just a dummy variable of integration and in general the integral is one value when evaluated for a certain d— so it made sense that I was plotting the function against d. The curve lines up with the S curves for all the k's.

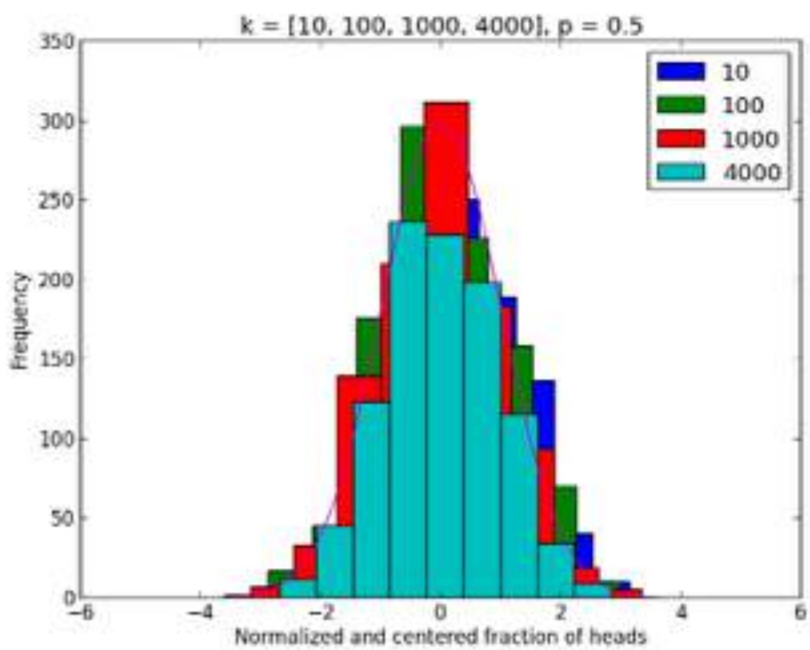
b) This part doesn't have codes or graphs
The total number of heads S is just the sum of all X_i from i=0 to to i = k. So S = X_1 + X_2 + ... + X_k. When you are at the jth toss, the number of heads so far is just the sum from i=0 to i=j. Of course, since S is a random number, the number of coin tosses S will have variations like what we have seen in previous homeworks.

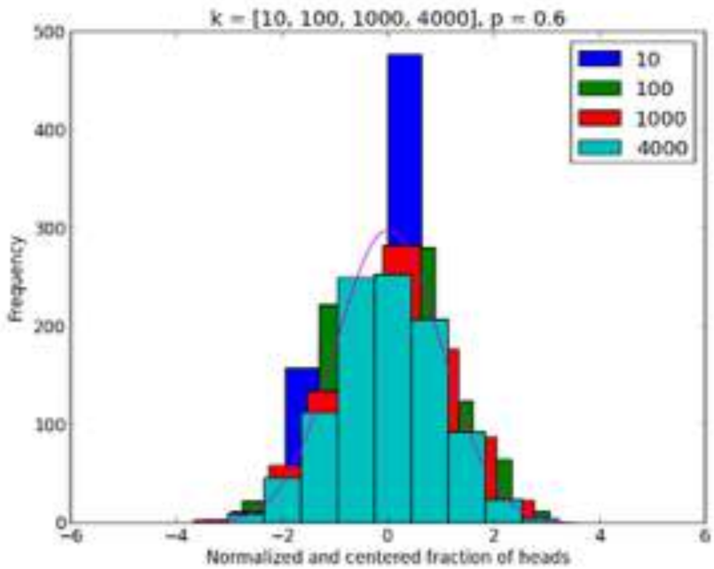
c)
```
def scale_integrand(x):
	return 750*integrand(x)

def partc(pranges=[0.3,0.4,0.5,0.6,0.7,0.9], kranges = [10,100,1000,4000], m=1000, 
show_indiv=False):
	# Q2 part (j)
	print('Question 2 part (j):')
	for p in pranges:
		print ('Probability of head p = %.1f'%p)
		std = math.sqrt(p*(1-p))
		results = {}
		for k in kranges:
			results[k] = [(Sk -k*p)/(math.sqrt(k)*std) for Sk in runManyTrials(p, k, m)]
			if show_indiv:
				# bins = 9 so as to not have gaps in the display
				plot.hist(results[k],bins=9,label=str(k))
```

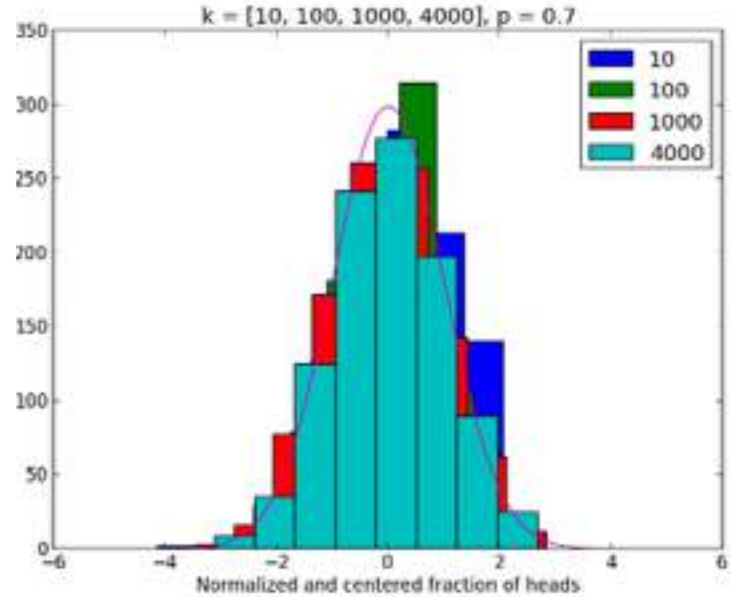
```
        plot.show()
plot.clf()
x_values = np.arange(-5,5,.1)
y_values = [scale_integrand(i) for i in x_values]
for k in kranges:
    # bins = 9 so as to not have gaps in the display
    plot.hist(results[k],bins=9,label=str(k),histtype='barstacked')
plot.legend()
plot.ylabel('Frequency')
plot.xlabel('Normalized and centered fraction of heads')
plot.title('k = %s, p = %.1f'%(str(kranges),p))
plot.plot(x_values,y_values)
plot.show()
```

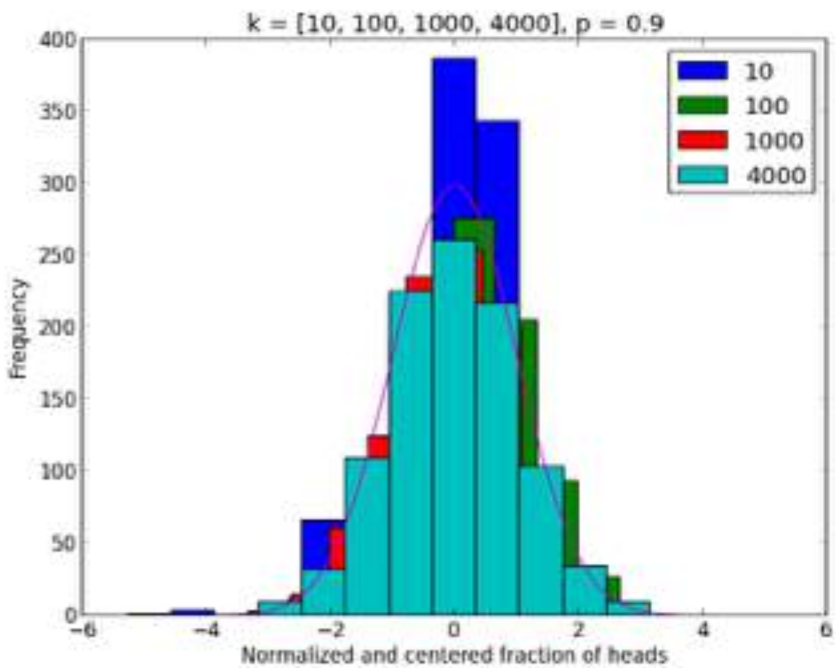










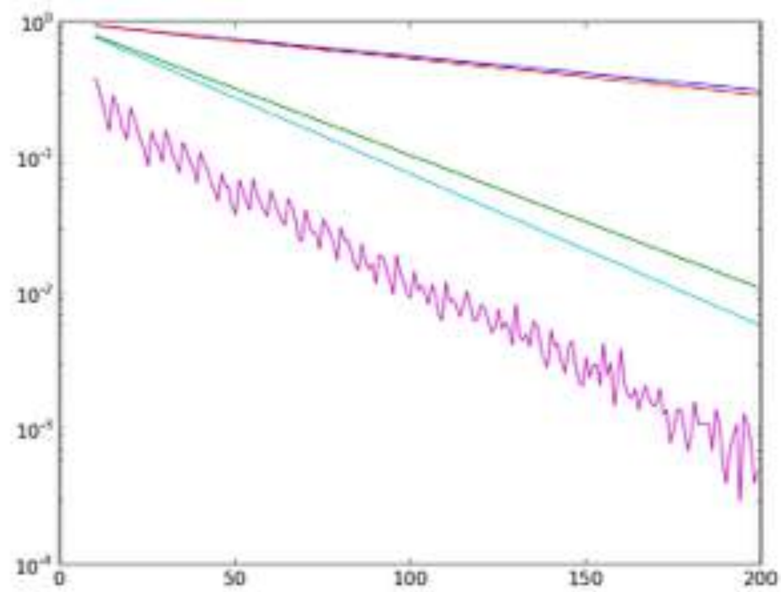


In order to gaussian function to lay over my histograms, I had to scale it by roughly 750, since the histogram y values are counts out of 1000 trials, whereas the gaussian itself has values between 0 and 1/sqrt(2*pi). This means that there is a way that we can represent our histograms since they seem to fit under the scaled gaussian function. That is, it looks like the area under the gaussian roughly corresponds to the "area" taken up by the histogram values. In essense, it means we can take an integral of the gaussian function as another representation of the histograms. This is exactly what we did in part a, and it seems that the integral of the gaussian produces the correct S curve shape that we got when we transformed the histograms into S curves.

d)

```
def int_a(a,p):
	return a*math.log((a/p),math.e)+(1-a)*math.log(((1-a)/(1-p)),math.e)
```

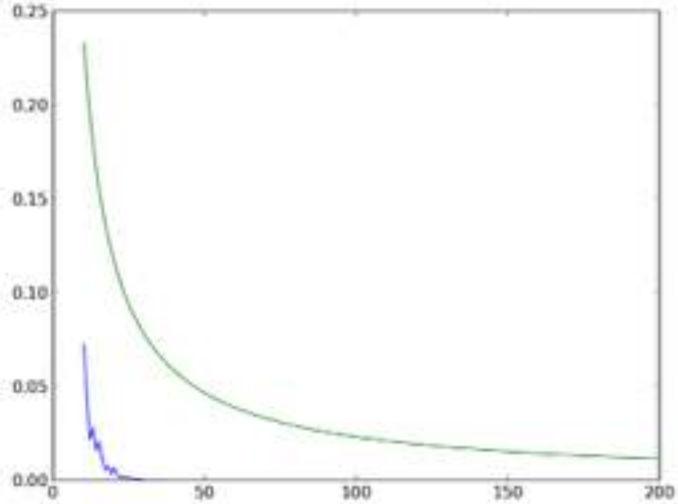
```
def func(a,p,k):
	return math.pow(math.e,(-1*int_a(a,p)*k))

def partd():
	r = np.arange(10,200,1)

	p_vals = [.3,.7]
	a_vals = [.05,.1]
	m = 10000
	for p in p_vals:
		for a_shift in a_vals:
			a = p + a_shift
			arr = [func(a,p,k) for k in r]
			plot.semilogy(r,arr)
			deltay = math.log(arr[-1])-math.log(arr[0])
			slope = deltay/190
			print("slope: " + str(slope))
	results = []
	for k in range(10,200):
		total = 0
		for n in runManyTrials(p,k,m):
			if n > a*k:
				total += 1
		results.append(total/float(m))
	plot.semilogy(range(10,200),results)
	plot.show()
```



In a log-linear graph, we can get the exponential drops to look like straight lines. It seems that the Chernoff bound (the Kullback-Liebler divergence against k) sets a lower limit to all the graphs. It suggests an inequality—all 4 of the plots are greater than the Chernoff bounds, and it looks like for all values of k along the x axis. In other words, the Chernoff bound gives us an asymptotic limit for the behavior of all combinations of p and a—no combination could ever be under this bound, as suggested by the first cases that we tried.
For p=.3 and a=.35, the KL divergence gives a value of .00578, which is roughly the negative of the slope of the first plot, which is approximate -.5/200=-.0025 on the log-linear scale.
For p=.3 and a=.4, the KL divergence gives a value of .0225; the negative of the slope of the second plot is approximatel -.5/200=-.0025 on the log-linear scale (same as the previous plot). This value doesn’t match up quite as well, and is strangely off by a factor of 10.
For p=.7and a=.75, the KL divergence gives a value of .061, which is roughly the negative of the slope of the first plot, which is approximate -2/200=-.01 on the log-linear scale. They match up okay.
For p=.7 and a=.8, the KL divergence gives a value of .0257, which is in the same order of magnitude as the negative of the slope of the fourth plot, which is -.01 (same as the previous plot).

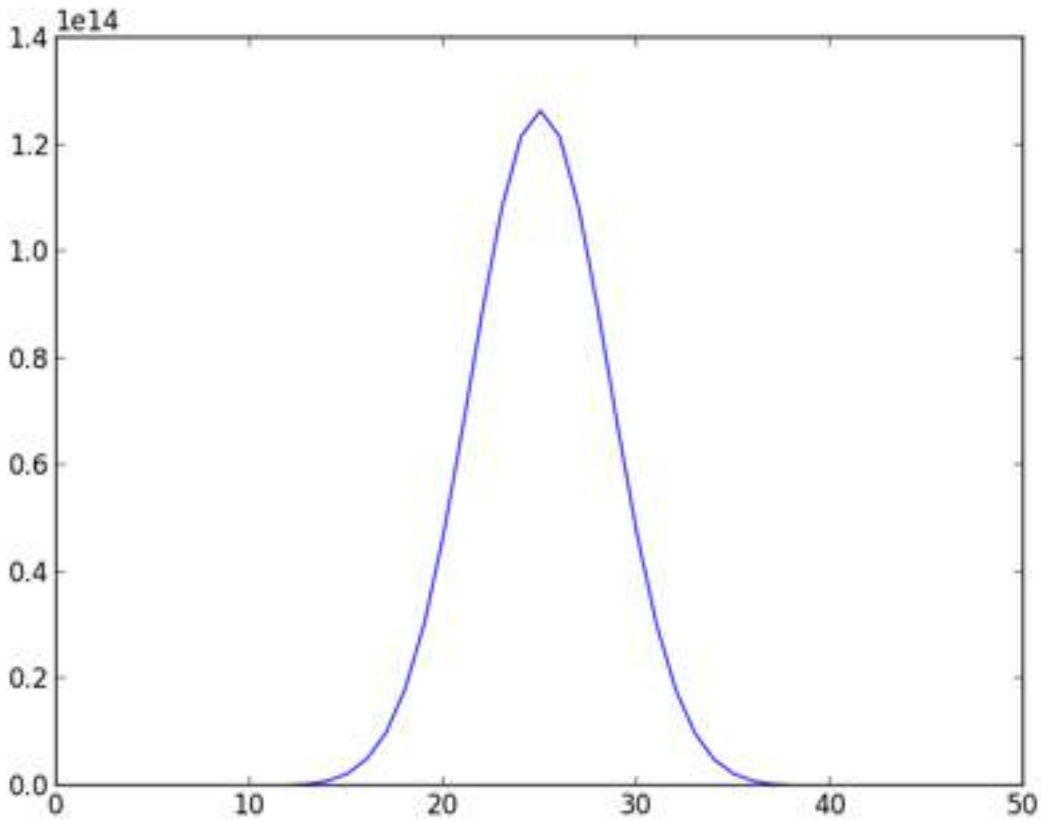
e)

```
def funct (eps,p,k):
	return p*(1-p)/(k*math.pow(eps,2))

def parte():
	x_values = np.arange(10,201)

	m = 10000
	p = .3
	for eps in [.1,.2,.3]:
		results = []
		for k in x_values:
			arr = [math.fabs(Sk -k*p) for Sk in runManyTrials(p, k, m)]
			total = 0
			for elem in arr:
				if elem >= eps*k:
					total += 1
			results.append(total / float(m))
	plot.plot(x_values,results)
	plot.plot(x_values,[funct(eps,p,k) for k in x_values])
	plot.show()
```



As the graph shows, we can get plot to be contained by the Chebyshev's equality, which is clearly greater than the plot at all points, so the inequality is valid for our coins. Its hard to plot of them on the same graph because they have different exponential behaviors, but in general it is clear that the inequality completely bounds every value in the actual frequencies.

f)



```
def choose(n,k):
	return math.factorial(n)/(math.factorial(k)*math.factorial(n-k))

def partf():
	x_values = np.arange(0,50.1,1)

	y_values = [choose(50,k) for k in x_values]

	plot.plot(x_values,y_values)
	plot.show()
```

The combination function (unordered picking, no replacements) is not always growing. It has a peak around the half way point of 50, and is fairly symmetric. It looks very much like a bell curve distribution, or the gaussian function that we plotted earlier, so I suspect that combinations are very related to the coin toss lab that we have been doing.

Kevin Chau
23816929
CS70-3h

Homework 9

Partners: Sam Drake, Serena Chan

2. a) 3 distinct toppings, order matters

$\Rightarrow {}_{10}P_3 = \frac{10!}{(10-3)!} = \frac{10!}{7!} = \boxed{720}$

This is just the permutation of 3 out of 10 toppings. Another way to see this is that you have 10 choices for the first, 9 for the second, and 8 for the third (where order matters):

$10 \times 9 \times 8 = \boxed{720}$

b) 3 distinct toppings out of 10, order does not matter. This is just the number of 3 topping combinations:

10 choose 3 $\rightarrow {}_{10}C_3 = \binom{10}{3} = \frac{10!}{3!(7!)} = \boxed{120}$

c) 0, 1, 2, or 3 distinct toppings.

0: $\binom{10}{0} = 1$ pizza
1: $\binom{10}{1} = 10$ pizzas
2: $\binom{10}{2} = 45$ pizzas
3: $\binom{10}{3} = 120$ pizzas

$\boxed{176 \text{ pizzas}}$

d) Exactly 3 toppings, does not have to be distinct

$\Rightarrow$ combinations with replacement: $\left(\!\binom{10}{3}\!\right) = \binom{10+3-1}{3} = \boxed{220}$ where $\left(\!\binom{n}{k}\!\right) \equiv \binom{n+k-1}{k}$

multichoose

e) 0 toppings: $\left(\!\binom{10}{0}\!\right) = 1$ pizza
1: $\left(\!\binom{10}{1}\!\right) = \binom{10}{1} = 10$ pizzas
2: $\left(\!\binom{10}{2}\!\right) = \binom{11}{2} = 55$ pizzas
3: $\left(\!\binom{10}{3}\!\right) = \binom{12}{3} = 220$ pizzas

$\boxed{286}$ pizzas

f) 0: 1 pizza
1: 10 pizzas
2: 55 pizzas
3: 220 pizzas
4: $\left(\!\binom{10}{4}\!\right) = \binom{13}{4} = 715$ pizzas
5: $\left(\!\binom{10}{5}\!\right) = \binom{14}{5} = 2002$ pizzas

$\boxed{3003}$ pizzas

3a) $\binom{100}{5} = \frac{100!}{5!\,95!} = \frac{100\times 99\times 98\times 97\times 96}{5\times 4\times 3\times 2} = \boxed{75287520}$ total ways to choose 5 of 100 (no replacement)

b) Ways to choose 5 unmarked

$\binom{90}{5} = \frac{90!}{5!\,85!} = \boxed{43949268}$ ways

$P(\neg go) = \frac{\binom{90}{5}}{\binom{100}{5}} = \frac{\#\text{ ways 5 unmarked}}{\#\text{ ways to pick 5}} = \frac{\frac{90!}{5!\,85!}}{\frac{100!}{5!\,95!}} = \boxed{\frac{90!\,95!}{85!\,100!} \approx .58375}$

c) ways to get 1 marked, 4 unmarked

$\binom{90}{4}\times\binom{10}{1} = \frac{90!}{4!\,86!}\,\frac{10!}{1!\,9!} = 25551900$ ways

$P(\text{1 marked, 4 unmarked}) = \frac{25551900}{75287520} = \boxed{.33939}$

$\binom{90}{3}\times\binom{10}{2} = \frac{90!}{3!\,87!}\,\frac{10!}{2!\,8!} = 117480\times 45 = 5286600$ ways

$P(\text{2 marked}) = \frac{5286600}{75287520} = \boxed{.07022}$

d) $P(\text{3 marked}) = \frac{\binom{90}{2}\binom{10}{3}}{\binom{100}{5}} = \frac{4005\times 120}{75287520} = \frac{480600\text{ ways}}{75287520} = \boxed{.00638}$

$P(\text{4 marked}) = \frac{\binom{90}{1}\binom{10}{4}}{\binom{100}{5}} = \frac{90\times 210}{75287520} = \frac{1890\text{ ways}}{75287520} = \boxed{.000251}$

$P(\text{5 marked}) = \frac{\binom{10}{5}}{\binom{100}{5}} = \frac{\frac{10!}{5!\,5!}}{\frac{100!}{5!\,95!}} = \frac{10!\,95!}{5!\,100!} = \boxed{.0000003347}$

$P(go) = P(1)+P(2)+P(3)+P(4)+P(5) = \boxed{.4162}$

The probability is the same as $1 - P(\neg go) = .41625$

e) This leaves 9 marked, 86 unmarked, 95 total

$\binom{86}{5} = 34826302$ ways to pick 5 unmarked

$\binom{95}{5} = 57940519$ ways to pick 5 of 95

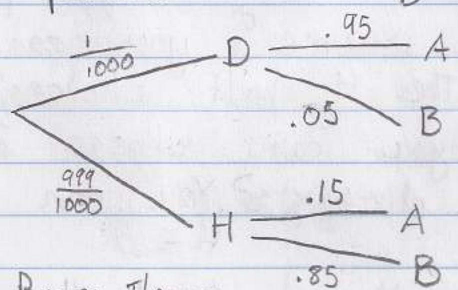
$P(\text{Tommy } \neg go) = \frac{\binom{86}{5}}{\binom{95}{5}} = \frac{\frac{86!}{5!\,81!}}{\frac{95!}{5!\,90!}} = \frac{86!\,90!}{81!\,95!} = .60107$

$P(\text{Tommy goes}) = 1 - .60107 = \boxed{.399}$

4. 1 in 1000 have disease. 95% chance of testing positive if a person has disease. 85% chance negative for person without

a) D = have disease H = healthy

A = positive B = negative



Using Bayes Theorem

$$P(D|A) = \frac{P(A|D)P(D)}{P(A)} = \frac{P(A|D)P(D)}{P(D\cap A)+P(H\cap A)} = \frac{P(A\cap D)}{P(D\cap A)+P(H\cap A)}$$

$P(A|D) = .95$ $P(D) = \frac{1}{1000}$ $P(A) = (\frac{1}{1000})(.95) + (\frac{999}{1000})(.15)$

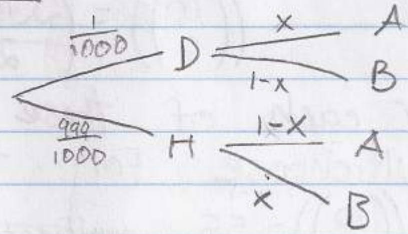
$$P(D|A) = \frac{(.95)(\frac{1}{1000})}{(\frac{1}{1000})(.95)+(\frac{999}{1000})(.15)} = \boxed{.0063}$$

b) $$P(H|B) = \frac{P(B|H)P(H)}{P(B)} = \frac{P(B|H)P(H)}{P(D\cap B)+P(H\cap B)} = \frac{P(B\cap H)}{P(D\cap H)+P(H\cap B)}$$

$P(B|H) = .85$ $P(H) = \frac{999}{1000}$ $P(B) = (\frac{1}{1000})(.05) + (\frac{999}{1000})(.85)$

$$P(H|B) = \frac{(.85)(\frac{999}{1000})}{(\frac{1}{1000})(.05)+(\frac{999}{1000})(.85)} = \boxed{.99994}$$

c) $P(D|A) = .9$



$$P(D|A) = \frac{P(A|D)P(D)}{P(A)}$$

$$.9 = \frac{(x)(\frac{1}{1000})}{(\frac{1}{1000})(x)+(\frac{999}{1000})(1-x)}$$ where x = accuracy

$$.9 = \frac{x}{1000(\frac{x}{1000}+\frac{999}{1000}-\frac{999x}{1000})} = \frac{x}{x+999-999x} = \frac{x}{999-998x}$$

$.9(999) - .9(998)x = x$

$(1+.9(998))x = .9(999)$

$$x = \frac{.9(999)}{1+.9(998)} = \boxed{.998888}$$ accuracy

Problem 5

Question: Crazy Pizza is a pizzaria with 10 toppings. However, you must specify 4 things to build your pizza: a topping for the bottom half, a topping for the top half, a topping for the left half, and a topping for the right half. How many unique pizzos can you make? Now assume you can choose no topping for any of the 4 half choices; how many pizzas? What if you can choose 2 toppings for each of the 4 divisions? You can repeat toppings for all parts.

Solution: First, we assume a unique pizza refers to the combinations in each quadrant - the order of the quadrants doesn't matter. So if I switch the top and bottom half, I have the same pizza if the left and right stay the same. Each quadrant has 2 toppings.

First, let's look at the top and bottom half. We have 10 toppings, and we can choose 2 toppings with replacement, and the order doesn't matter. So we use multichoose.

$$\left(\!\binom{10}{2}\!\right) = \binom{11}{2} = 55 \text{ multisets for top and bottom.}$$

For each of these multisets I can do another multichoose for the left and right

$$\left(\!\binom{10}{2}\!\right) = 55 \text{ multisets for left and right}$$

Then we have $\left(\!\binom{10}{2}\!\right)^2 = 55^2 = \boxed{3025 \text{ pizzas}}$

Now let's look at choosing no topping for any of the halves. This really is just an 11th topping:

$$\left(\!\!\binom{11}{2}\!\!\right)^2 = \binom{12}{2}^2 = 66^2 = \boxed{4356 \text{ pizzas}}$$

2 toppings on each half:

Again, we look at the first toppings for the top and bottom:

$$\left(\!\!\binom{11}{2}\!\!\right) = 66$$

and multiply by the number of multisets for the second top and bottom toppings

$$\left(\!\!\binom{11}{2}\!\!\right)^2 = \binom{12}{2}^2 = 4356$$

And multiply by the same number for left and right

$$\left(\!\!\binom{11}{2}\!\!\right)^4 = 4356^2 = \boxed{18974736 \text{ pizzas}}$$

In general, if you can choose n toppings for each half, you have

$$\boxed{\left(\!\!\binom{11}{2}\!\!\right)^{2n} \text{ pizzas}}$$

Problem 6 - Midterm Question 3

(1,0) (2,6) (3,0) (4,0) (6,0) GF(7)

$\Delta_1(x) = \frac{(x-2)(x-3)(x-4)(x-6)}{(1-2)(1-3)(1-4)(1-6)} =$

$\Delta_2(x) = \frac{(x-1)(x-3)(x-4)(x-6)}{(2-1)(2-3)(2-4)(2-6)}$ $1 \cdot -1 \cdot -2 \cdot -4 = -8 \equiv 6 \text{ mod } 7$

$\Delta_3(x) = \frac{(x-1)(x-2)(x-4)(x-6)}{(3-1)(3-2)(3-4)(3-6)}$

$\Delta_4(x) = \frac{(x-1)(x-2)(x-3)(x-6)}{(4-1)(4-2)(4-3)(4-6)}$

$\Delta_6(x) = \frac{(x-1)(x-2)(x-3)(x-4)}{(6-1)(6-2)(6-3)(6-4)}$

$P(x) = 6(\Delta_2(x)) = \frac{6(x^2-4x+3)(x^2-10x+24)}{6}$

$6^{-1} \text{ mod } 7 \equiv 6$

$P(x) = 36(x^4 - 10x^3 + 24x^2 - 4x^3 + 4x^2 - 96x + 3x^2 - 30x + 72)$

$= 6(6x^4 + 6(-14)x^3 + 6(31)x^2 + 6(-126)x + 6(72))$

$\equiv x^4 + 6(4)x^2 + 6(5) \quad \text{mod } 7$

$\equiv x^4 + 3x^2 + 2 \quad \text{mod } 7$

$P(0) = 0 + 0 + 2 = 2$

$\boxed{\text{secret: } 2 = \text{Pidgeot}}$

7. Midterm Q4:

$300^{300} \text{ mod } 35$

$\Rightarrow 300^{300} \text{ mod } 7 \equiv (300 \text{ mod } 7)^{300 \text{ mod } 6} \text{ mod } 7 \equiv (300 \text{ mod } 7)^0 \text{ mod } 7$

$\Rightarrow 300^{300} \text{ mod } 7 \equiv 1 \text{ mod } 7$

$\Rightarrow 300^{300} \text{ mod } 5 \equiv (300 \text{ mod } 5)^{300 \text{ mod } 4} \equiv 0 \text{ mod } 5$

$1 \text{ mod } 7 \equiv \{1, 8, 15, 22, 29, \ldots\}$

$0 \text{ mod } 5 \equiv \{0, 5, 10, 15 \ldots\}$

$300^{300} \text{ mod } 35 \equiv \boxed{15 \text{ mod } 35}$

8. MTQ5

$N = 77$ $e = 3 \Rightarrow$ public key $(3, 77)$

$d = 26 \Rightarrow$ private key $(26, 77)$

N is a small number, so we can prime factor it easily:

$p = 7$ $q = 11$

e needs to be coprime with $(p-1)(q-1)$

so $\gcd(e, 6 \cdot 10) = \gcd(e, 60) = 1$

Bob first goes wrong when he chooses e, because 3 is not co-prime with 60. So this RSA scheme will not work for encoding and decoding. Since Bob did not use a public key e that is coprime with $(p-1)(q-1)$ (an RSA requirement), the RSA functions will not be bijections, so the scheme wouldn't work.

9. MTQ6

Prove: If two $d \leq n-1$ polynomials agree at n distinct points ($P(x_i) = Q(x_i)$ for $1 \leq i \leq n$), then they are the same elsewhere.

Proof: From note 7, property 2 says that given n distinct points (distinct x_i), there is a unique polynomial of degree d at most $n-1$, so $d \leq n-1$. Therefore P is uniquely determined by the n points it shares with Q, and similarly Q is uniquely determined by the same n points it shares with P. Since n points give a unique $d \leq n-1$ polynomial, P and Q must be the same polynomials. Hence they agree at all other points.

We can also prove by contradiction. Suppose P and Q agreed at those n points, but were different everywhere else, so they are different

polynomials. Now consider $P - Q = R(x)$.
$R(x)$ has n roots, since P and Q agree at n points.
but $R(x)$ is a degree at $n-1$ polynomial, (since a degree $n-1$ polynomial minus a degree $n-1$ polynomial is still at most $n-1$)
which contradicts the property that a degree d polynomial has d roots. So P and Q must be the same, and therefore they agree everywhere else (than the n distinct points).

10. MT Q 7

a) A) $K=100$ B) $K=1000$ C) $K=10000$

As the number of coin flips increases, the spread in absolute coin tosses that were heads increases. When plotted on all the same scales, we expect $k=10000$ to have the widest histogram

b) A) $K=100$ B) $K=1000$ C) $K=10000$

As the number of coin flips increases, we expect the the relative fraction of heads to get narrower around $.5-.8$, since its easier to flip a very small number of coins all heads than a large number. So $K=10000$ should have the narrowest graph

c) A) 40%   B) 50%   C) 60%

We expect that the 40% bag would be the left most curve since almost all the trials would have at most .5 fractions of heads. It would also be likely that almost all the 50% coins would be contained by at most 60% of them heads, so it makes sense for C (the furthest right) to be 60%.

11. MTQ8

We need to account for k erasure errors, where $k = n$. So $k \geq (n+k)F$, where $0 \leq F \leq \frac{1}{4}$ is the fraction of packets lost.

Then $k - kF \geq nF$

$$\Rightarrow k \geq \frac{nF}{1-F} \Rightarrow k \geq \frac{n(\frac{1}{4})}{1-\frac{1}{4}} = \frac{\frac{n}{4}}{\frac{3}{4}} = \frac{n}{3}$$

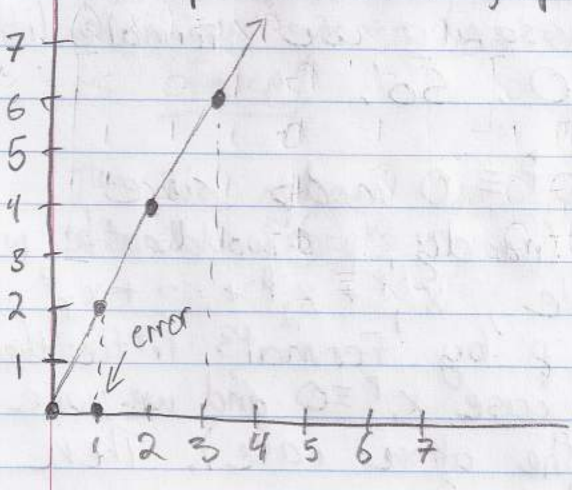
so we need to send $k = \lfloor \frac{n}{3} \rfloor$ additional packets at least in order to account for the erasure errors.

12. MT Q9

$A=0 \quad B=1 \quad C=2 \quad D=3 \quad E=4 \quad F=5 \quad G=6$.

$AAEG = \{0, 0, 4, 6\} \quad n=2 \quad k=1$

Plot points on a graph



Since Alice is sending a message of length 2 with $2k = 2$ redundancies, she needs a degree 1 polynomial from interpolating the points. Clearly, the polynomial she used was $P(x) = 2x$, and the error is at $x = 1$.

$P(1) = 2 \cdot 1 = 2 = C$

Alice tried to send Bob $\boxed{ACEG}$

13. MT Q10

$n \geq 1$, $x_1, x_2, x_3, \ldots, x_n \geq 1$, p is prime

$(x_1 + x_2 + \cdots + x_n)^p \equiv (x_1 + x_2 + \cdots + x_n) \bmod p$

since Fermat's little theorem says $a^p \equiv a \bmod p$

but then $x_1 \equiv x_1^p \bmod p$ and $x_i \equiv x_i^p \bmod p$ for $1 \leq i \leq n$.

Then $(x_1 + x_2 + \cdots + x_n) \equiv x_1^p + x_2^p + \cdots + x_n^p \bmod p$

so transitively

$(x_1 + x_2 + \cdots + x_n)^p \equiv x_1^p + x_2^p + \cdots + x_n^p \bmod p$

Thus we have directly proven the statement is $\boxed{\text{true}}$

However, we need to consider the case when some of the x_i are 0. Then we have

$$(x_1 + x_2 + \cdots + 0 + 0 + \cdots + x_n)^p \equiv (x_1 + x_2 + \cdots + x_n)^p$$
$$\equiv (x_1^p + x_2^p + \cdots x_n^p) \bmod p.$$

So it reduces down to the first proof without the x_i's $= 0$. If all the x's sum to 0, then we can't necessarily use Fermat's little theorem. Now $\sum_{i=1}^{n} x_i = 0$. So:

$(x_1 + x_2 + \cdots x_n)^p \equiv 0^p \equiv 0 \bmod p$ since

0 to any prime is zero (namely, $p = 0$ would not be prime)

Evaluating the right side, $x_1^p + x_2^p + \cdots + x_n^p$, we get $x_i^p = x_i \bmod p$ by Fermat's little theorem (unless $x_i = 0$, in that case $x_i^p \equiv 0$ and we use arguments similar to the above case). Then

$$x_1^p + x_2^p + \cdots + x_n^p \equiv x_1 + x_2 + \cdots + x_n$$

but we already know $\sum_{i=1}^{n} x_i = 0$, so

$x_1^p + x_2^p + \cdots + x_n^p = 0$

Then both sides equal zero, so the proof is always true, even in this special case.

14. MT Q11 $n \geq 1$, $r \geq 1$

n characters in a polynomial degree $\leq n-1$

evaluate at $n+r$ points

False: counter example

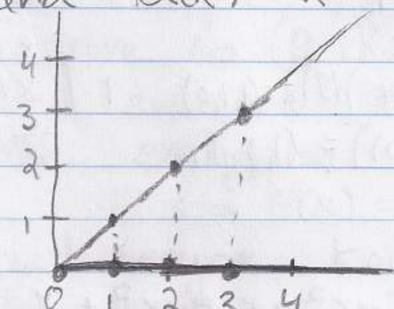
Let $n=2$ $r=2$ and the field $GF(5)$

so we have the letters A, B, C, D, E

Then the messages are encoded into 1 degree polynomials (which are just lines).

Suppose one message is "AA" and the other is "AB."

Then the polynomials that encode them are $P(x)=0$ and $Q(x)=x$



We evaluate both of these polynomials at $n+r=4$ points. Clearly they differ by $3=r+1$ points. So these two messages differ in fewer than $r+2$, hence the claim that two codewords of different messages must differ in at least $r+2$ places is $\boxed{\text{False}}$.

The two different codewords are $\{0,1,2,3\}$ and $\{0,0,0,0\}$

15. MT Q 12 Let $P(x) = x^3+x+1$
$Q(x) = x+1$

EGCD(P(x), Q(x)):
  if Q(x) == 0:
    return (P(x), 1, 0)
  else:
    $(D(x), A(x), B(x)) = EGCD(Q(x), P(x) - \lfloor \frac{P(x)}{Q(x)} \rfloor Q(x))$
    return $(D, B, A - \lfloor \frac{P(x)}{Q(x)} \rfloor B(x))$

$EGCD(x^3+x+1, x+1)$
$(D, A, B) = EGCD(x+1, x^3+x+1 - (x^2+x)(x+1)) = (1,0,1)$
$\boxed{\text{return } (1, 1, 0-(x^2+x)(1))}$

$\cancel{x^3}+\cancel{x}+1 - \cancel{x^3} - x^2 - x^2 - \cancel{x} = -\cancel{2x^2}^{0} + 1 = 1$

$$\begin{array}{r} x^2 + x \quad r\ 1 \\ x+1 \overline{) x^3 + x + 1} \\ \underline{x^3+x^2} \\ x^2+x \\ \underline{x^2+x} \\ 1 \end{array}$$

$EGCD(x+1, 1)$
$(D, A, B) = EGCD(1, x+1 - (x+1)(1)) = (1,1,0)$
return $(1, 0, 1 - (x+1)(0)) = (1, 0, 1)$

$$\begin{array}{r} x+1 \quad r\ 0 \\ 1 \overline{) x+1} \\ \underline{x} \\ 0 \end{array}$$

EGCD(1, 0)
return (1, 1, 0)

so $A(x) = 1$ and $B(x) = -x^2 - x \equiv x^2 + x \mod x^3+x+1$

we want $A(x)P(x) + B(x)Q(x) = 1 \mod x^3+x+1$

$\cancel{(x^3+x+1)}^{0} + (x^2+x)(x+1) \mod x^3+x+1$
$\equiv (x^3 + \cancel{x^2} + \cancel{x^2} + x) \pmod{x^3+x+1}$
$\equiv x^3 + x \pmod{x^3+x+1}$
$\equiv -1 \pmod{x^3+x+1}$
$\equiv 1 \pmod{x^3+x+1}$ in GF(2)

so the inverse of $x+1$ in mod x^3+x+1
is $\boxed{x^2+x}$

16. Midterm Q13

a) Everyone knows the public decryption module, which takes an input number y and has the known public key (d, N). So everyone knows that their input y will be interpreted by the computer as

$$x = D(y) = y^d \bmod N$$

Now let N be the product of two primes P, Q (so $N = PQ$), and d be the inverse to e, a private encryption key only known to the computer, in mod $(p-1)(q-1)$, so e is coprime to $(P-1)(Q-1)$.

Now everyone knows the computer applies the encryption function:

$$s = E(x) = x^e \bmod N$$

and everyone knows that when $s = s_0$ = the magic # the computer will blow up. Even though everyone knows how the decryptor works, so they can figure out what x the computer encrypts when they input whatever y, they have no idea what x_0 such that $E(x_0) = s_0$ since only the computer knows e.

Given that the computer chooses a large enough P and Q, N will be hard to factorize and therefore the RSA scheme ensures that it is very difficult for any one to figure out the private key (e, N) despite knowing (d, N). Essentially, this is just the reverse of the normal RSA scheme, so it works since the RSA functions are bijections

b) Suppose now we make a secret sharing scheme such that when the necessary parties agree, the secret input y_0 is revealed, so the people can blow up the computer.
First, make a 1 degree called $P_{all}(x)$ such that $P_{all}(0) = y_0$. Then evaluate $P_{all}(x)$ at 2 other points, say $P_{all}(1)=y_1$ and $P_{all}(2)=y_2$.
Now make 2 faction polynomials of degree 1 $P_{blue}(x)$ and $P_{gold}(x)$ such that $P_{blue}(0)=y_1$ and $P_{gold}(0)=y_2$.
Now evaluate P_{blue} at 4 other points, say $x_1=1, \dots, x_4=4$ and give each $P_{blue}(x_i)$ to one of the blue families, and do the same for P_{gold} and each of the gold families.
Since P_{blue} is a 1 degree polynomial, if at least 2 blue families combine their shares, they can interpolate P_{blue} and figure out $P_{blue}(0)=y_1$. If 2 gold families agree, they can interpolate for $P_{gold}(0)=y_2$. Now only if both 2 blue families and 2 gold families agree can they know both y_1 and y_2 at the same time. Once they do, they can interpolate for $P_{all}(x)$ and hence find $P_{all}(0)=y_0$. However, any less than 2 families from each will give them less than 2 points on $P_{all}(x)$, so it would fail.
We can make 2 new polynomials $Q_{blue}(x)$ and $Q_{gold}(x)$ that are each of degree 3. Let $Q_{blue}(0) = Q_{gold}(0) = y_0$

Now evaluate Q_{blue} and Q_{gold} at $x = 1, 2, 3, 4$ and give each of the respective evaluations to a blue family and a gold family. If 4 blue families come together, they won't need the golds and can interpolate to find $Q_{blue}(x)$ and therefore immediately know the secret input $Q_{blue}(0) = y_0$. Similarly, if 4 golds agree, they know $Q_{gold}(0) = y_0$. Under any other conditions, Q_{blue} and Q_{gold} won't be useful, and hence they still need 2 blues and 2 golds to agree to find y_0 with P_{blue}, P_{gold}, and P_{all}

17. MT Q14

Let n be the number of elements in each input list, so one list is $a = (a_1, \ldots, a_n)$ and the other is $b = (b_1, \ldots, b_n)$. Now we can assume the machine makes general corruption errors, so we use a $(n, 2k)$-Reed Solomon code for each list and give the machine $a' = (a'_1, \ldots, a'_{n+2k})$ $b' = (b'_1, \ldots, b'_{n+2k})$. Berlekamp-Welch lets us get all $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ back correctly, so we are gauranteed the sums $(a_1+b_1, \ldots, a_n+b_n)$, provided we choose the right k.

The machine makes at most $\frac{1}{3}$ of the sums wrong. So we must have

$k \geq (n+2k)F$ extra packets, where F is the fraction corrupted

$$\Rightarrow k \geq \frac{nF}{1-2F}$$

then $k \geq \frac{n\frac{1}{3}}{1-2\left(\frac{1}{3}\right)} = \frac{\frac{n}{3}}{\frac{1}{3}} = n$

so then we need to send $\boxed{2n} = 2k$ additional packets on each list to make sure the machine always works.