Question 1

Bloom filter (BF) is a memory efficient data structure to store information where the downside is that you cannot delete information and you cannot retrieve information because it is not a key and value pair structure. You can check if an data is stored in the BF, but you may encounter false positive, meaning that it may indicate certain data is in the BF when it actually not. The false negative is impossible in a BF, so it is applicable in all cases where false positive is tolerable while false negative is not.

A simple example of a bloom filter is to decide if you want to access a file over a system. It would be much more time saving if you could know before hand if the file exists on that system before you access it. Bloom Filters can help us do that.

Another example that is popular and common is a username check system on a website. Whenever you try to resiter on a website, you usually need to create an username. But often you might get a notification that the username you want is already taken. That is possible thorough a bloom filter.

A regualr Bloom Filter supports addtion of an element and query/search of an element. It doesn't support deletion as it would require unsetting one or more of the bits array. This could easily lead us to accidentally delete other elements in the set as well as every stored element has the value of 1.

Here is how a typical Bloom Filter functions: We have two hash functions, giving the word “Professor Sterne”, one returned 8, another one returned 5. BF array is binary array that only contains 1 and 0. Initially every item in BF is set to 0 (off state). The current value of items No. 5 and 8 become changed into 1 (on state) in the array. Again, we give it the word "CS110". Now it returns 1 and 3. Since 1 and 3 positions are 0(off state), they get changed to 1 (on state). Now, if we want to check whether “Elon Musk” is in the list, so requesting it gives 5 and 3. Since both 5 and 3 are in on state, so the program will say “Elon Musk” is in the list. This is where false positive occurs.

Question 2

In [1]:
```
from __future__ import division
import numpy as np
import math
import random

name_list = set()
with open("randomnames.txt", "r") as data:
    for line in data:
        name_list.add(line.rstrip(" \xc2\xa0\n"))
print "Random Names Count:", len(name_list)
name_list = list(name_list)
training = name_list[:int(len(name_list)/2)]
testing = name_list[int(len(name_list)/2):]
```

```
Random Names Count: 200
```

In [2]:
```
class BloomFilter(object):

    def __init__(self, length):
        self.length = length
        self.k = 2 #no. of hash func. is pre-determined
        self.arr = np.zeros(length, dtype=np.bool)
        self.added = 0

#Following hash functions are modified versions of 7.2 pre-work#

    def hash_1(self, string):
        answer = 0
        random.seed(ord(string[0]))
        for letter in string:
            answer += random.getrandbits(32)
        return answer % self.length

    def hash_2(self, string):
        answer = 0
        for letter in string:
            answer = answer * 128 + ord(letter)
        return answer % self.length

    def add(self, string):
        self.arr[self.hash_1(string)] = self.arr[self.hash_2(string)] = True
        self.added += 1

    def checking(self, string):
        if self.arr[self.hash_1(string)] and self.arr[self.hash_2(string)]:
            return True
        else:
            return False

    def testing_rate(self, name_list):
        counter = 0
        for name in name_list:
            if self.checking(name):
                counter += 1
        return counter / len(name_list)

    def theory_rate(self):
#Formula is taken from Wikipedia#
        return (1 - math.exp((-self.k * self.added) / self.length)) ** self.k

    def __str__(self):
        return str(self.arr)
```

Question 3

I used two hash functions that are slight modifications from the hash functions used in class 7.2 preclass work.

hash_1 uses a random approach, the seed will be the first letter of each string, and then randomly generate numbers. This function may end up with collisions especially if the length of strings are the same. But this hash function is really useful as it uses less space and can store more amount of unique values.

hash_2 breaks down the string into characters and then gets the unicode value of each character, the multiplies the answer by 128 and then adds the next character.

(I perform “answer % self.length” for both hash functions so that it will be in the range of the BF.)

Question 4

According to Wikipedia (has low credibility, but works), the theoretical false positive rate:

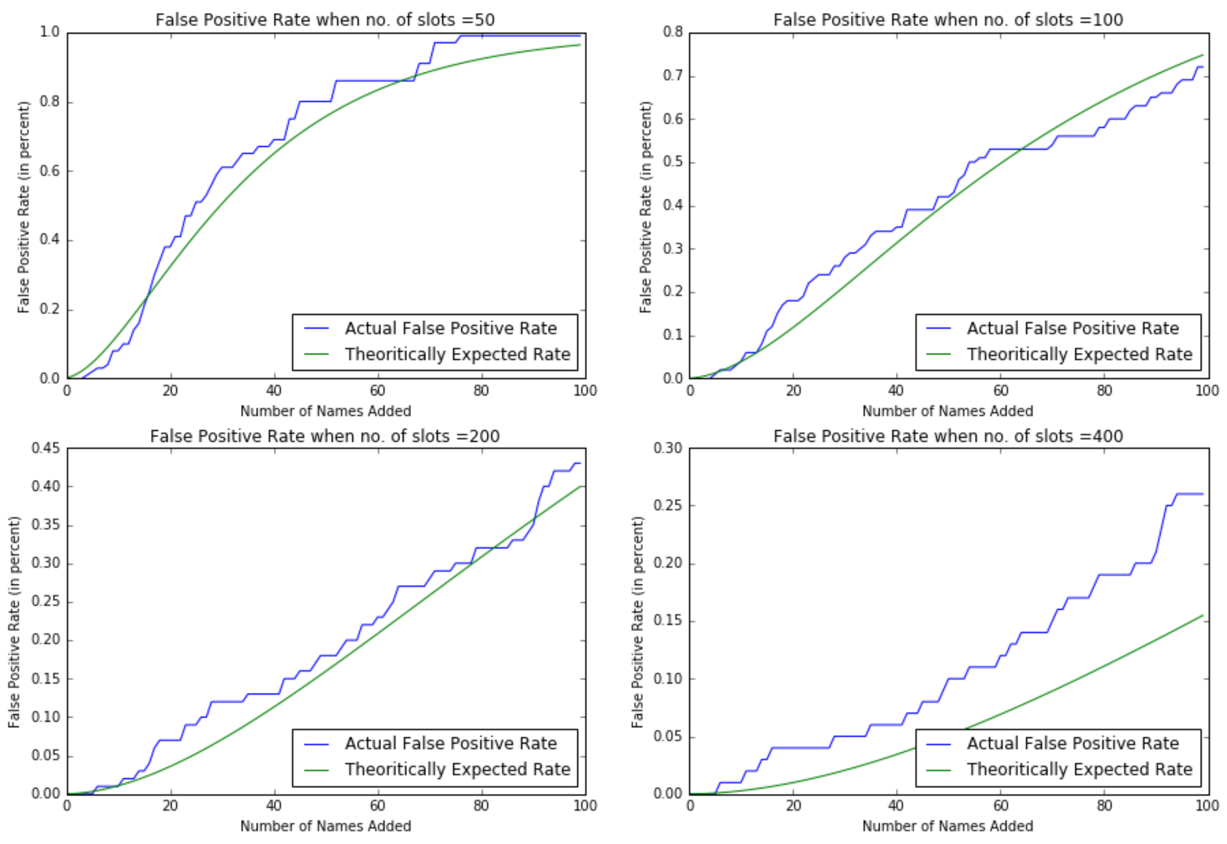
[1-{1-(1/m)}^kn]^k

1. The memory size is m in the formula, so you can see that the bigger m is, the smaller the result will be, so the false positive rate declines as memory increases. It makes sense because the more memory you add for the variables to be stored, the less likely the program will generate a false positive (this also depends on the distribution of the hash function. We need to have a hash function that gives out a uniform distribution to support this decrease in the false positive rate with the increase in memory.)
2. If BF uses a static array, the memory size should be predetermined by the time the array being created, so it should be a fixed number regardless how many item has been stored, but if that is the case, the false positive rate will be different (will increase with more items as more slots will be switched on). If we want the same average false positive rate, we need more memory size with the increase of number of items stored.
3. The number of the hash functions being used is k. The access time is related to k. The time complexity is O(k). Increase in access time means that k will increase. Based on the formula, if k increases the false positive rate should decrease.
4. If k does not change, the access time has nothing to do with the number of item stored, but if we want to keep false positive rate at an average, then when the number of items stored increases, we need to increase k and therefore the access time to keep it balanced.

Question 5

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
data, ((ax1,ax2), (ax3, ax4)) = plt.subplots (2, 2, figsize=(15,10))
axis = [ax1, ax2, ax3, ax4]

numbers = [50, 100, 200, 400]
for i, nums in enumerate(numbers):
    bloomfil = BloomFilter(nums)
    practical_rates = []
    theoritical_rates = []
    for name in training:
        bloomfil.add(name)
        practical_rates.append(bloomfil.testing_rate(testing))
        theoritical_rates.append(bloomfil.theory_rate())
    legend1, = axis[i].plot(xrange(len(training)), practical_rates, label = "Act
    legend2, = axis[i].plot(xrange(len(training)), theoritical_rates, label = "T
    axis[i].legend(handles = [legend1, legend2], loc=4)
    axis[i].set_title("False Positive Rate when no. of slots =" + str(nums))
    axis[i].set_xlabel("Number of Names Added")
    axis[i].set_ylabel("False Positive Rate (in percent)" )
```



References

1. Pagel, M. (2016, November 09). What are Bloom filters, and why are they useful? Retrieved November 05, 2017, from https://sc5.io/posts/what-are-bloom-filters-and-why-are-they-useful/#gref (https://sc5.io/posts/what-are-bloom-filters-and-why-are-they-useful/#gref)
2. Talbot, J. (2015, July 15). What are Bloom filters? – 3 min read. Retrieved November 05, 2017, from https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff (https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff)
3. Nielsen, M. (2012, September 26). Why Bloom filters work the way they do. Retrieved November 05, 2017, from http://www.michaelnielsen.org/ddi/why-bloom-filters-work-the-way-they-do/ (http://www.michaelnielsen.org/ddi/why-bloom-filters-work-the-way-they-do/)
4. Bloom Filters by Example. (n.d.). Retrieved November 05, 2017, from https://llimllib.github.io/bloomfilter-tutorial/ (https://llimllib.github.io/bloomfilter-tutorial/)
5. Davies, J. (n.d.). Bloom Filters. Retrieved November 06, 2017, from https://www.jasondavies.com/bloomfilter/ (https://www.jasondavies.com/bloomfilter/)
6. Bloom filter. (2017, November 02). Retrieved November 06, 2017, from https://en.wikipedia.org/wiki/Bloom_filter (https://en.wikipedia.org/wiki/Bloom_filter)
7. List of Random Names. (n.d.). Retrieved November 05, 2017, from http://listofrandomnames.com/ (http://listofrandomnames.com/)

Appendix

1. Random list of names .txt file - https://file.io/RKLzPU (https://file.io/RKLzPU)