HW #3 – Darin Ellis – 9/24

I developed my algorithms in Python using Wing IDE. Any python interpreter should run the program given the .py file. Let me know if there is any difficulties so I can resolve them.

I generated 16-bit random numbers by using Python's random library to give me integers of either 1 or 0 and then appending those into a list. Of those 16-bit numbers that were presumed prime (as they passed a Fermat method) and passed into the 100, I found that only 1 was actually not a prime, by using the brute force method. It surprises me a little, but not too much. The chance of that particular algorithm failing is incredibly small (something on the order of 10^(-20)). The probability of a failure is much higher in this sample, but I believe it is consistent with what we discussed in class, given the small sample size. I am almost certain that if I tried this more times I would get more "0" false primes than anything else. I must have been relatively unlucky!

16 bit tries: 3

32 bit tries: 3

64 bit tries: 79

128 bit tries: 48

256 bit tries: 323

In theory, the chance to hit a prime number is approximately 3/n. My results are essentially consistent with this theory, given that there is a sharp upwards trend as the number of bits (n) increases (with a notable exception at 128 bits, but that is not unexpected given the randomness and small sample size). I measured these tries in my code using a simple accumulator that incremented after each number was generated but not considered prime.

Note: 256-bit numbers generated are in a separate txt file.

16-bit time: 0.10300588607788086 seconds

32-bit time: 0.5440309047698975 seconds

64-bit time: 2.0971200466156006 seconds

128-bit time: 13.711785078048706 seconds

256-bit time: 60.903483867645264 seconds

To obtain these values, I used a time library and measured the time it took to execute main. It is definitely not a linear growth, as it is not proportional to n. I would estimate it to be θ(n^2), since it is sharply increasing (but not at a ridiculous rate). I argue that it is big theta and

not big oh since there is no real case that there would be a noticeably smaller big omega, given that there will always be many iterations. This is rather consistent with what we discussed in class, since we knew that there would be many more iterations to change primality as n increased.