David Bozarth
CES 514
Fall 2005

Homework #1

2. a. and b.

The objective is to characterize the distribution of the first digit of a very long integer that results from repeated multiplications of very long integers. We agree that an N-digit integer may not have zero as its most significant digit.

A Java program (see "Ex_1_2.java") generates 100,000 sets of random numbers. Each set contains K random numbers of arbitrary length N. (The value of N is limited by the size of the Java integer.) Each random number represents the most significant N digits of a very long integer of unspecified length. The elements of within each set are multiplied, and the 100,000 resulting products are used to form the distribution of most-significant digits.

A potential problem can be thought of as a type of overflow. If the downstream digits in one or more factor integers are large, then the value of the most significant digit of the K-product is uncertain. Using the factors 19999 by 29999 causes the 2-product MSB to be 3 – not the 2 suggested by the MSB's of the 2 factors. Decrementing either of the two factors once, removes this difficulty.

On the basis of this observation, I decided to let N be a lower bound on the length of each integer, and to have the actual length increment by one for each unit of K. A trial run using K = 2 and N = 5 yielded 61 occurrences of unstable MSB's. (The way to detect these was discussed in class – incrementing the factors and multiplying a second time, then comparing that product with the original product.) After increasing N to 10, I found no more unstable MSB's.

Using N = 10, I found and graphed the distributions for K = 2 and K = 20. In addition I calculated and graphed the theoretical distribution discussed in class, where the normalized value of the random variable is the base 10 log(1 + 1/a), where a is the MSB. As expected, using the higher value of K resulted in a close match with the theoretical distribution.

Getting K = 20 to work was a bit of a trick that required the Java LongInteger class. The centerpiece of this program turned out to be the function getRandom(n), which returns a String representing a random number whose length is limited by the size of the Java integer, and whose first digit is not zero. This function was not thoroughly tested, but it seemed to work fine for this problem.

The graphs of the 3 distributions are shown on the following page.

3.

For this problem I used a Basic routine that simulates a series of coin tosses, and reports the average number of coin tosses it takes to turn up the sequence “TTH”. The output from a run of that program is shown below, and the upper figure is for “TTH”.

```
:date
September 28, 2005; Wednesday   internal: 13786   julian: 271
:coin

Number  games =   100000

First  player
   games won         =    66413
   avg. tosses to win =  6.39

Second  player
   games won         =    33587
   avg. tosses to win =  5.39
```

The reported expectation is that by the seventh toss, the sequence “TTH” would have turned up. Since we expect 1 occurrence for every 6.39 tosses, the expected number of occurrences in 50 throws is (50 / 6.39) = 7.82.

The 50-character sequence given in the problem contains 5 occurrences. We could establish a confidence interval and determine the probability that Mr. Hill is telling the truth. I would be inclined to accept his claim. If he had reported a 500-character sequence with only 50 occurrences, then I would be suspicious.