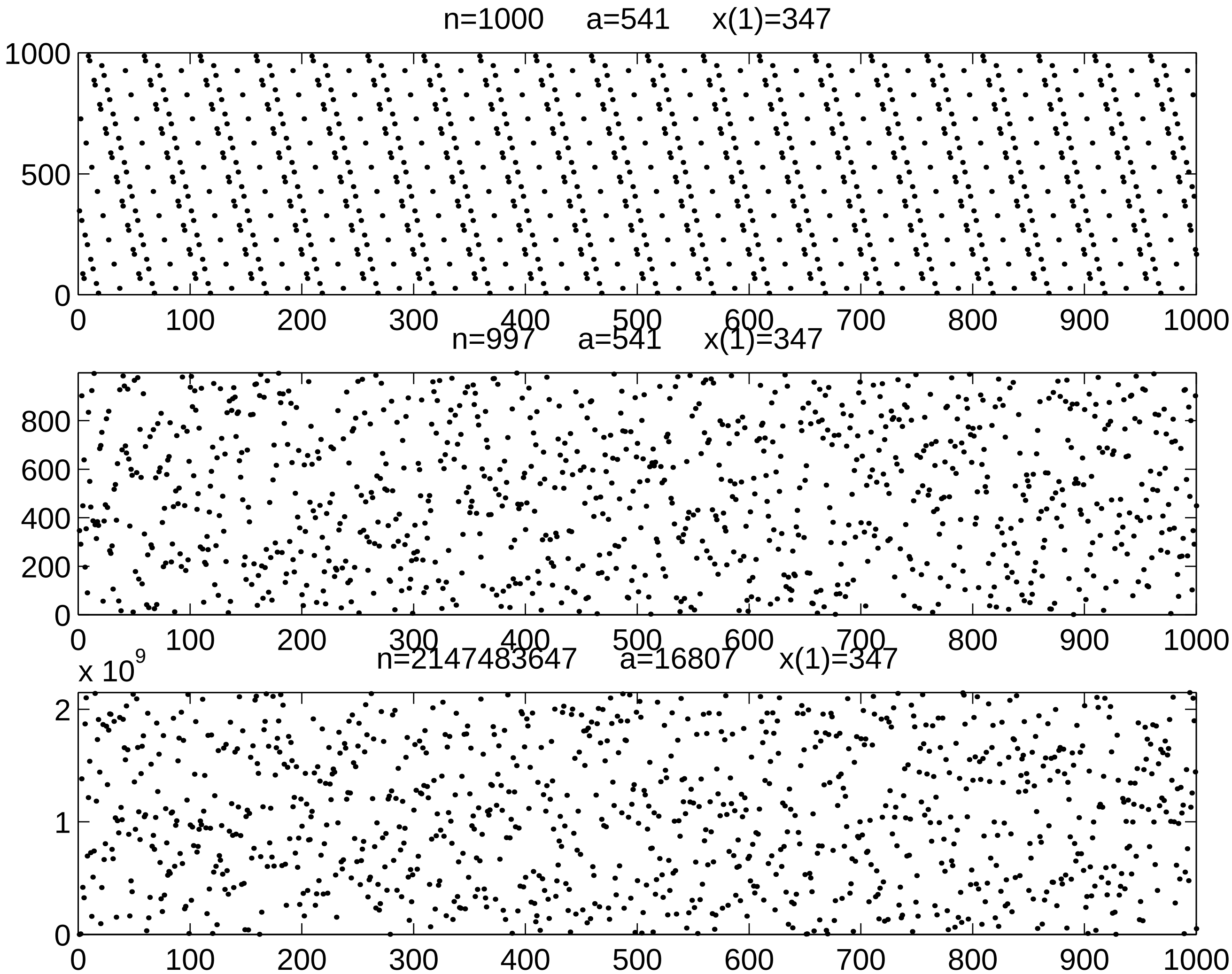
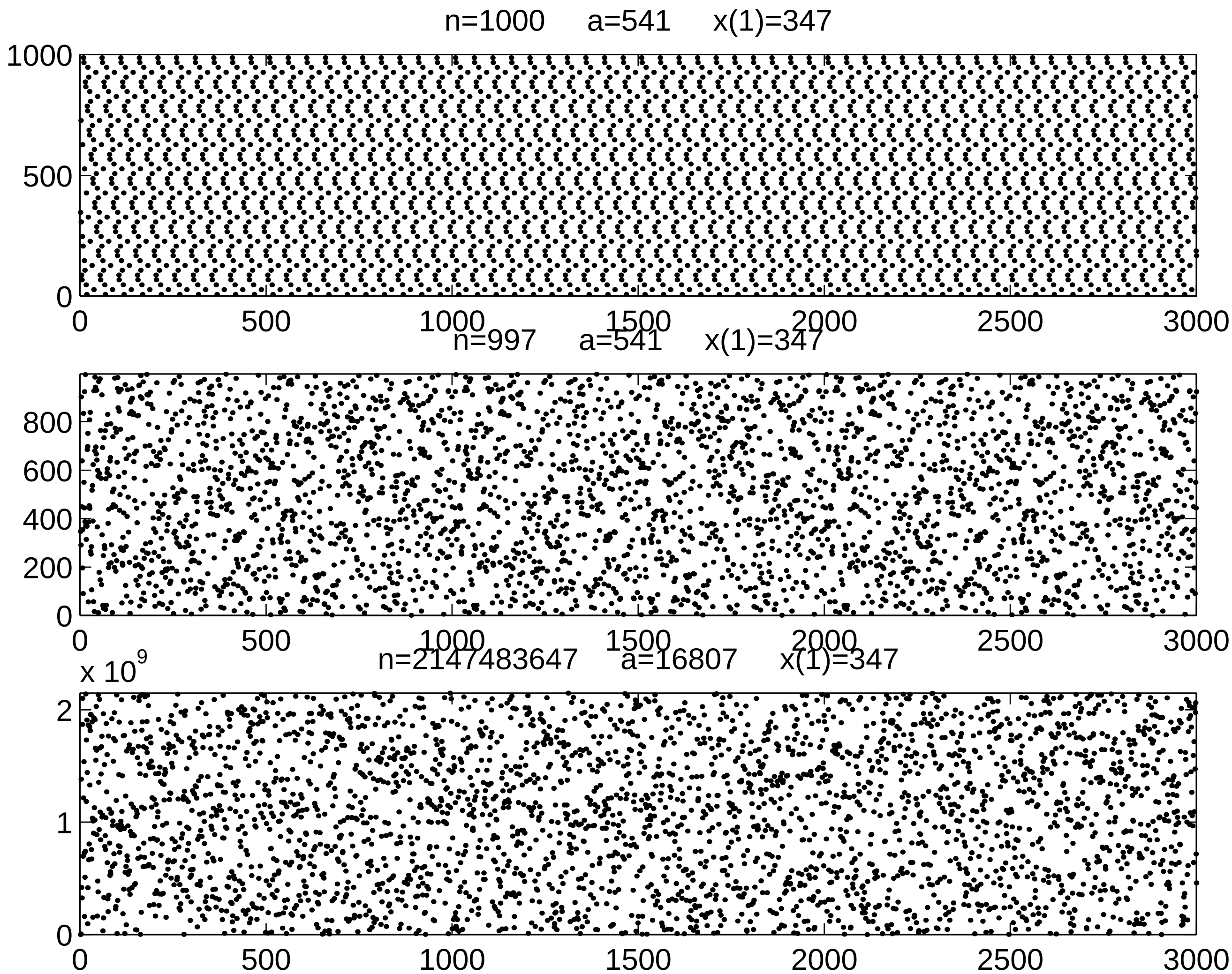
```
Math 55 - Spring 2004 - Lecture notes # 6 - Feb 5 (Thursday)

   Keep reading Chapter 2:

   Goals of chapter 2: Integer algorithms and number theory, basis for:
                       how to generate hash tables
                       how to generate random numbers
                       how computer arithmetic works (hardware or software)
                       how to do encryption/decryption

   Goal for today: Basic properties of primes,
                   greatest common divisor gcd(a,b)
                   division algorithm
                   hash tables
                   random numbers

   Primes
   DEF: if a and b are integers, a != 0, say a|b if a divides b, ie.
        exists integer f such that b=a*f; else say a /| b
    EX: 2|2, 2|100, 1| anything, anything | 0, 2 /| 1001, 3 | 111111, 9 | 72252
        (reminder of rules for whether 3|a, 9|b, proofs later)
    Thm: a|b and a|c => a|(b+c)
         Proof: a|b <=> b=a*f1 for some f1, a|c <=> c=a*f2 for some f2,
                so a|b and a|c -> b+c=a*(f1+f2) -> a|b+c
    Thm: a|b => a|b*c; a|b and b|c => a|c
 ASK&WAIT: why?
    DEF a positive integer p is a prime if the only positive integers which
        divide it are 1 and p; else composite
    EX: 2,3,5,7,11,13,... are prime
    Theorem (Fundamental Theoremm of Arithmetic): every positive integer
      has a unique prime factorization, where the factors are written in
      increasing order.
    Proof: wait till we learn induction in Chapter 3
    EX: 100 = 2*2*5*5 = 2^2 * 5^2; 1024 = 2^10
 ASK&WAIT: how many primes are there? Why?

    DEF a,b, integer, not both 0.
        gcd(a,b) = greatest common divisor of a and b, is largest integer d
                   such that d|a, d|b
 ASK&WAIT:  why exclude a=b=0?
 ASK&WAIT:  gcd(6,9)?, gcd(1,101)?, gcd(0,234)?,
     DEF if gcd(a,b)=1, we say a and b are relatively prime.
```

```
ASK&WAIT suppose a = 2^5  * 3^2  * 5^1
                 b = 2^4  * 3^3  * 5^2
            what is gcd(a,b)?
         suppose a = 2^n2 * 3^n3 * 5^n5
                 b = 2^m2 * 3^m3 * 5^m5
            what is gcd(a,b) =?

    First algorithm for computing gcd(a,b):
       1) factor a = 2^n1 * 3^n2 * 5^n3 * ...
       2) factor b = 2^m1 * 3^m2 * 5^m3 * ...
       3) let  gcd = 2^min(m1,n1) * 3^min(m2,n2) * 5^min(m3,n3) * ...
    Later: a (much faster!) algorithm to compute gcd(a,b),
           without factoring a and b

   DEF a,b, postive integer, lcm(a,b) = least common multiple
       = smallest positive integer divisible by a and b
ASK&WAIT   EX: lcm(6,9)?, lcm(1,101)? lcm(0,234)?
ASK&WAIT   Suppose a = 2^5  * 3^2  * 5^1
                   b = 2^4  * 3^3  * 5^2 - what is lcm(a,b)?

    Algorithm for computing lcm(a,b):
       1) factor a = 2^n1 * 3^n2 * 5^n3 * ...
       2) factor b = 2^m1 * 3^m2 * 5^m3 * ...
       3) let  gcd = 2^max(m1,n1) * 3^max(m2,n2) * 5^max(m3,n3) * ...

ASK&WAIT   What is gcd(a,b)*lcm(a,b)?

   Theorem (division algorithm) given integers a, d>0 (divisor), there is a
       unique q (quotient) and r (remainder) such that 0<=r<d, a=q*d+r
ASK&WAIT:  is this an algorithm?

   DEF a integer, d>0, then a mod d = r, remainder after dividing a by d
  Note: in C,C++, this is written a%d
  EX:  7 mod 3 = 1, since 7=1+2*3; 3 mod 7 = 3; anything mod 1 = 0.
ASK&WAIT:  what is 87813134 mod 1000
ASK&WAIT:  what is 27 mod 8 = 11011_2 mod 2^3

Application of Division Algorithm: Hashing functions.
   A "hash table" is a data structure
   where you can store data and search for it (usually) very quickly  (CS61B)
   EX: You want to store records (student ID#, name, grades) in a data base,
       and look up records quickly given the student ID#, an 8-digit integer.
```

```
      We could have a table of length 10^8,
        array Student[100000000]
      where Student(i) contains the record with the name, grades of student i
      and just look at entry i to find data for student i.
      But this is too large a table, since it is too large to fit in memory,
      and since there are many fewer students than 10^8. Instead we use a
      smaller table (say size 10^5 for Berkeley, enough to hold all students,
      and a little more) and do the following:

        array Student[100000]
        a = f(i)    ... compute address a in table of record of student i
        record = Student[a]

      Hash function f(i) needs to map 8 digit integers to 5 digit integers,
      to look up in table of length 10^5. It should spread the data
      out across the table evenly, to use whole table.

      Simple function to use: f(i) = i mod 10^5
        (generally, i mod m, m = length of table)
      EX: data for student i = 87654321 stored at address a=f(i)=54321
ASK&WAIT:   what happens if two students i and j have same a=f(i)=f(j)?

 Application: Random number generation, or how "rand" function works in C,C++

    Random number generation means producing a sequence of integers
    x(1), x(2), ... all in the range [0,N-1], where each x(i) is chosen
    ``at random''. For example, if N=6, we could roll a die to get each x(i):
    each value from 0 to N-1 is equally likely, and each x(i) is
    "independent" or unrelated to all other x(j). We want an algorithm
    to produce such a sequence efficiently.

    Uses: 1) game programs (so game different each time)
          2) many fast algorithms (quicksort)
          3) programs to simulate real world events which occur at random:
               simulate data traffic in new network design
               simulate elevator traffic in new building design
               simulate bits of fluid in a turbulent air flow between
                  a disk head and a disk surface in a new disk design or
                  over an airplane wing in a new airplane design
          4) related idea used for hash functions

    We will use a simple algorithm (based on division algorithm)
```

```
    to generate x(i+1) from x(i), so x(i) is not random in sense of
    rolling dice (since it is easy to predict x(i+1) from x(i), if you
    know the formula used in the algorithm, but it will "look random",
    and be good enough for purposes described above.
ASK&WAIT:  Why not use a really random function to generate x(i),
           e.g. counting ticks on a Geiger counter, or looking at
           certain stock exchange data (eg 3rd lowest digit of volume)
    The formula is x(n+1) = a*x(n) mod n,
    This is called a linear congruential method, because the formula
    involves a linear function a*x(n) and a congruence (or mod)
    EX: x(n+1) = 3*x(n) mod 7 yields 1 3 2 6 4 5 1 3 2 6 4 5 ...
    Thm: Any linear congruential random number generator generates
         a periodic sequence, i.e. it eventually repeats the
         same sequence over and over
ASK&WAIT: Why? How long can the random sequence be before it repeats?
    EX: x(n+1) = 4*x(n) mod 7 yields 1 4 2 1 4 2 ...
        So choice of a, n important
        (see Knuth, "Art of Computer Programming", vol 2)
    EX: Following 3 figures illustrate good and bad random sequences
    Comments on first 3 figures (1000 samples in each one)
        Top: n=1000, a=541, x(1)=347; doesn't even look random
             only 49 different values of x(i), so period only 49
        Middle: n=997, a=541, x(1)=347; much better, period 997
        Bottom: n=2^15-1, a=7^5, x(1)=347; much better, period 2^15-1
    Comments on next 3 figures (3000 samples in each one)
        Periodic behavior of top obvious
        Periodic behavior of middle less obvious
        No periodic behavior of bottom yet
```


n=1000 a=541 x(1)=347
n=997 a=541 x(1)=347
x 10^9
n=2147483647 a=16807 x(1)=347


n=1000 a=541 x(1)=347
1000
500
0
0
500
1000
1500
2000
2500
3000
n=997 a=541 x(1)=347
800
600
400
200
0
0
500
1000
1500
2000
2500
3000
x 10^9
n=2147483647 a=16807 x(1)=347
2
1
0
0
500
1000
1500
2000
2500
3000