Random-Number Generation

Weiss, chapter 9 (pages 393–417)
Random Numbers

• Test a sorting algorithms
  ▪ Write a routine that generates many inputs
  ▪ Generate random permutations of the first $N$ integers.
    ▪ A permutation of 1, 2, ..., $N$ is a sequence of $N$ integers, where each integer appears exactly one.
    ▪ The result of sort is always 1, 2, ..., $N$.

• Randomness depends on algorithms that generate random numbers.
  • Generate a sequence of pseudorandom numbers with uniform distribution.
  • In uniform distribution, all numbers in a specified range are equally likely to occur.
  • java.util.Random class provides methods for generating uniformly distributed numbers. For example, nextInt(), nextInt(int n)
Random Numbers

• Assume random numbers are uniformly distributed by adding 1 to the previous random number. The first number is the number of milliseconds for a system clock. 
  3, 4, 5, 6, ....

• This random number generators fails two statistical tests:
  ▪ The sum of two consecutive random numbers is even or odd with equal likelihood.
  ▪ There are duplicates in the resulting random numbers.

• Good random number generators fail fewer tests than the bad ones.
Linear Congruential Generator

• An algorithm for generating uniformly distributed numbers: \( X_1, X_2, ... \)

\[
X_{i+1} = AX_i \pmod{M} \rightarrow x[i+1] = A \times x[i] \% M
\]

The first random number \( (X_0) \) is called a seed. \( 1 \leq X_0 < M \)

\[
M = 9, A = 2, X_0 = 1 \\
2, 4, 8, 7, 5, 1, 2, 4, ...
\]

• The sequence repeats after a period (6).

• Good random number generators provide greater periods. The longest period is called the full period (period of length \( M - 1 \)).

• For example \( M = 11 \) and \( A = 7 \) gives a full period of 10.
Linear Congruential Generator

- Libraries have a random number generator based on:
  \[ X_{i+1} = (Ax_i + C) \mod 2^B \]
  
  B is the number of bits for doing integer-based arithmetic.

  Java library uses B = 48, but returns high 32 bits for a random integer.

```java
long A = 25214903917L;
long M = (1L << B); // 2^B : 1000....0
long MASK = M - 1; // 01111...1 (preserves low 48 bits)
public int nextInt( ) { return next( 32 ); } 

private int next( int bits ) {
    long state = ( A * state + C ) & MASK; //
    return (int) (state >>> ( B -bits ) ); // High 32 bits generate more random numbers.
} 
```
Random Permutation

- Generate random permutations of 1, 2, ..., N. Each permutation is equally likely.
- A random permutation is generated in linear time.

```java
public static final void permute ( Object [ ] a ) {
    Random r = new Random( );
    for( int i = 1; i < a.length; i++ ) {
        swapReferences( a, i, nextInt( 0, i ) );
    }
}
```

- All permutations are not equally likely.
- A permutation depends on a generated random integer (index of the array in the above example).
- This may not be enough for generating a many permutations.
Randomized Algorithm

- A randomized algorithm perform computations on the permutations of an input.
- Particular bad inputs are not important. These inputs are not generated frequently.
- Measure *expected* running time for randomized algorithms. It is an average over all possible permutations of random numbers for a specific input.