CS61B Lecture #32

om Numbers (Chapter 11)

e random sequences?

ndom sequences"?

om sequences.

ne.

a library classes and methods.

nutations.

Why Random Sequences?

stical samples

rithms

y:

random keys

g streams of random bits (e.g., SSL xor's your data with atable, pseudo-random bit stream that only you and the can generate).

se, games

What Is a "Random Sequence"?

"a sequence where all numbers occur with equal fre-

3, 4, ...?

ow about: "an unpredictable sequence where all numbers qual frequency?"

0, 1, 1, 2, 2, 2, 2, 2, 3, 4, 4, 0, 1, 1, 1,...?

t is wrong with 0, 0, 0, 0, ...anyway? Can't that occur election?

Pseudo-Random Sequences

able, a "truly" random sequence is difficult for a com- nan) to produce.

poses, need only a sequence that satisfies certain sta- erties, even if deterministic.

e.g., cryptography) need sequence that is *hard* or *im-* predict.

om sequence: deterministic sequence that passes some statistical tests.

, look at lengths of *runs:* increasing or decreasing con- equences.

ly, statistical criteria to be used are quite involved. For Knuth.

nerating Pseudo-Random Sequences

as you might think.

mplex jumbling methods can give rise to bad sequences.

uential method is a simple method used by Java:

$$X_0 = \text{arbitrary seed}$$
$$X_i = (aX_{i-1} + c) \bmod m, \quad i > 0$$

large power of 2.

sults, want $a \equiv 5 \bmod 8$, and $a,\ c,\ m$ with no common

nerator with a *period of m* (length of sequence before nd reasonable *potency* (measures certain dependencies ent X_i.)

ts of a to "have no obvious pattern" and pass certain (see Knuth).

$= 25214903917$, $c = 11$, $m = 2^{48}$, to compute 48-bit om numbers. It's good enough for many purposes, but *aphically secure.*

What Can Go Wrong (I)?

ls, many impossible values: E.g., $a,\ c,\ m$ even.

erns. E.g., just using lower 3 bits of X_i in Java's 48-bit get integers in range 0 to 7. By properties of modular

$$\begin{aligned} \bmod 8 &= (25214903917X_{i-1} + 11 \bmod 2^{48}) \bmod 8 \\ &= (5(X_{i-1} \bmod 8) + 3) \bmod 8 \end{aligned}$$

period of 8 on this generator; sequences like

$$0, 1, 3, 7, 1, 2, 7, 1, 4, \ldots$$

le. This is why Java doesn't give you the raw 48 bits.

Additive Generators

erator:

$$X_n = \begin{cases} \textit{arbitary value}, & n < 55 \\ (X_{n-24} + X_{n-55}) \bmod 2^e, & n \geq 55 \end{cases}$$

es than 24 and 55 possible.

period of $2^f(2^{55}-1)$, for some $f < e$.

mentation with circular buffer:

```
55;
+31) % 55];  // Why +31 (55-24) instead of -24?
   /* modulo 2^32 */
```

. 54] is initialized to some "random" initial seed val-

aphic Pseudo-Random Number Generator Example

good *block cipher*—an encryption algorithm that en-
s of N bits (not just one byte at a time as for Enigma).
ample.

rovide a key, K, and an initialization value I.

udo-random number is now $E(K, I + j)$, where $E(x, y)$ is
on of message y using key x.

Adjusting Range (II)

bias problems when n does not evenly divide 2^{48}, Java
alues after the largest multiple of n that is less than

```
 integer in the range 0 .. n-1, n>0. */
(int n) {
 next random long (0 ≤ X < 2^48);
k for some k)
n  top k bits of X;

 largest multiple of n that is < 2^48;
i >= MAX)
 next random long (0 ≤ X < 2^48);
i / (MAX/n);
```

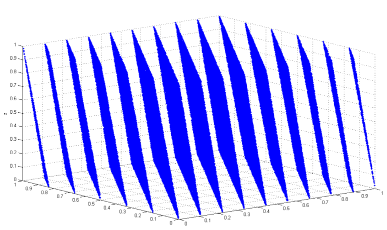

What Can Go Wrong (II)?

ds to bad correlations.

s IBM generator RANDU: $c = 0$, $a = 65539$, $m = 2^{31}$.

U is used to make 3D points: $(X_i/S, X_{i+1}/S, X_{i+2}/S)$,
es to a unit cube, ...

be arranged in parallel planes with voids between. So
ts" won't ever get near many points in the cube:



is Sanchez at English Wikipedia - Transferred from en.wikipedia to Commons
., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3832343]

aphic Pseudo-Random Number Generators

orm of linear congruential generators means that one
future values after seeing relatively few outputs.

you want *unpredictable* output (think on-line games in-
y or randomly generated keys for encrypting your web

phic pseudo-random number generator (CPRNG) has the
hat

ts of a sequence, no polynomial-time algorithm can guess
bit with better than 50% accuracy.

current state of the generator, it is also infeasible to
ct the bits it generated in getting to that state.

Adjusting Range and Distribution

quence of numbers, X_i, from above methods in range
^{48}, how to get uniform random integers in range 0 to

easy: use top k bits of next X_i (bottom k bits not as

be careful of slight biases at the ends. For example, if
$X_i/(2^{48}/n)$ using all integer division, and if $(2^{48}/n)$ gets
n, then you can get n as a result (which you don't want).

fix that by computing $(2^{48}/(n-1))$ instead, the proba-
ing $n-1$ will be wrong.

Arbitrary Bounds

- rbitrary range of integers (L to U)?
- m `float`, x in range $0 \le x < d$, compute

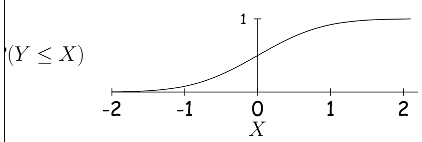
```
xtInt(1<<24) / (1<<24);
```

- ble a bit more complicated: need two integers to get

```
nd = ((long) nextInt(1<<26) << 27) + (long) nextInt(1<<27);
 bigRand / (1L << 53);
```


ieneralizing: Other Distributions

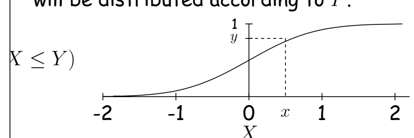
- iave some desired probability distribution function, and random numbers that are distributed according to that How can we do this?
- e normal distribution:



- desired probability distribution. $P(Y \le X)$ is the prob- random variable Y is $\le X$.

Other Distributions

- e y uniformly between 0 and 1, and the corresponding x will be distributed according to P.



Java Classes

- `()`: random double in $[0..1)$.
- `til.Random`: a random number generator with construc-
  - ierator with "random" seed (based on time).
  - **d)** generator with given starting value (reproducible).
  - r random integer
  - nt in range $[0..n)$.
  - andom 64-bit integer.
  - **(), nextFloat(), nextDouble()** Next random values of other types.
  - **n()** normal distribution with mean 0 and standard devia- ell curve").
- `.shuffle`(L, R) for list R and `Random` R permutes L ing R).

Shuffling

- a random permutation of some sequence.
- b technique for sorting N-element list:
  - N random numbers
  - ch to one of the list elements
  - ist using random numbers as keys.
- a bit better:

```
ist L, Random R) {
= L.size(); i > 0; i -= 1)
ement i-1 of L with element R.nextInt(i) of L;
```

	1	2	3	4	5	Swap items	0	1	2	3	4	5
	2♣	3♣	A♡	2♡	3♡	$3 \Longleftrightarrow 3$	A♣	3♡	2♡	A♡	3♣	2♣
	3♡	3♣	A♡	2♡	2♣	$2 \Longleftrightarrow 0$	2♡	3♡	A♣	A♡	3♣	2♣
	3♡	2♡	A♡	3♣	2♣	$1 \Longleftrightarrow 0$	3♡	2♡	A♣	A♡	3♣	2♣

Random Selection

- que would allow us to select N items from list:

```
 and return sublist of K>=0 randomly
ments of L, using R as random source. */
st L, int k, Random R) {
 L.size(); i+k > L.size(); i -= 1)
nt i-1 of L with element
t(i) of L;
list(L.size()-k, L.size());
```

- efficient for selecting random sequence of K distinct m $[0..N)$, with $K \ll N$.

ernative Selection Algorithm (Floyd)

```
nce of K distinct integers
 0<=K<=N. */
ts(int N, int K, Random R)

w IntList();

-K; i < N; i += 1) {
s in S are < i
ndInt(i+1); // 0 <= s <= i < N
et(j)  for some j)
 value i (which can't be there
ter the s (i.e., at a random
ther than the front)
 i);

 random value s at front
);
```

Example

i	s	S
5	4	[4]
6	2	[2, 4]
7	5	[5, 2, 4]
8	5	[5, 8, 2, 4]
9	4	[5, 8, 2, 4, 9]

`selectRandomIntegers(10, 5, R)`