PSEUDO-RANDOMNESS :

"Randomness is in the eye of the beholder"

Randomness is very useful in computer science

- Eg. randomized algos can be simpler & more powerful than deterministic counterparts
  For some tasks, det. solns of similar complexity not known. [Eg. Find an n-bit prime number.]
- Crucial for cryptography
  - necessary for picking keys
  - One-time pad: $Enc_K(m) = m \oplus K$ (secret key)
  - Length of K = length of m (list of random bits)
    Can't reuse K to send another message.

Where do we get randomness?

Pseudo-randomness : Create strings that are random enough for the purpose at hand.

↳ What does this mean?

↓

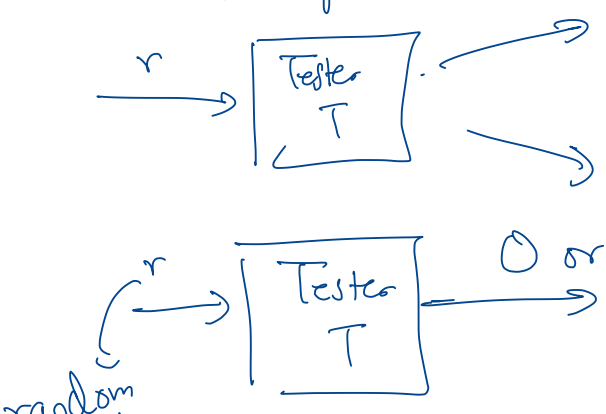
- Give precise definitions
- Constructions that realize such pseudorandomness

Distribution $D \sim \{0,1\}^n$

Eg. U_n = uniform dist on $\{0,1\}^n$.

Define notion of ε-closeness to U_n.

How? Think of a "test" that should tell apart D from U_n based on a sample.



Distinguishing probability:

$$\left| \Pr_{r \sim D}[T(r)=1] - \Pr_{r \sim U_n}[T(r)=1] \right|$$

Example: D – uniform distribution on even parity strings (with even # 1's)

Natural tester:

$T(r) = 1$ if r has even # 1's

0 otherwise

Distinguishing prob: $1 - 1/2 = 1/2$.

(Above dist. is $\frac{1}{2}$-close to U_n.)

Exercise: $1/2$ is the best distinguishing prob.

Defn. Dist D on $\{0,1\}^n$ is ϵ-close to U_n if $\forall$ Tests $T : \{0,1\}^n \to \{0,1\}$

$$\left| \Pr_{r \sim D}[T(r)=1] - \Pr_{r \sim U_n}[T(r)=1] \right| \leq \epsilon.$$

Statistical or Total variation distance between D and U_n is $\leq \epsilon$.

$$\Delta(D, U_n) = \max_{\text{tests } T} \left| \Pr_{r \sim D}[T(r)=1] - \Pr_{r \sim U_n}[T(r)=1] \right|$$

stat. dist

Exercise: $\Delta(D, U_n) = \frac{1}{2} \sum_{x \in \{0,1\}^n} \left| D(x) - \frac{1}{2^n} \right|$

$D(x)$: Prob. of sampling x under D.

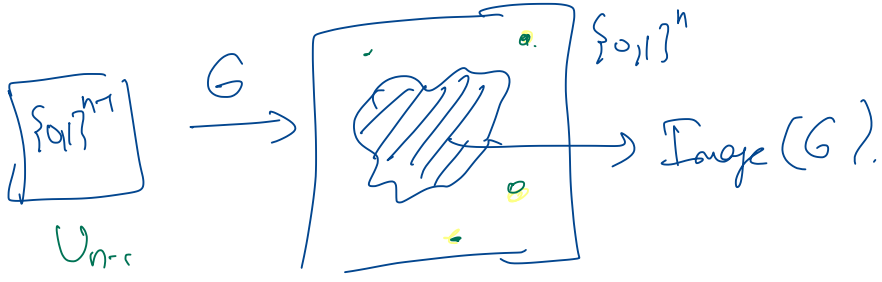
How to produce a pseudorandom string distributed according to D which is close to U_n?

Deterministic Generator $G: \{0,1\}^{n-1} \to \{0,1\}^n$

Feed G $(n-1)$ random bits,

output of G should "be like" n-random bits (ε-close to U_n?)

☹ No such G can exist!



Check: $\Delta(G(U_{n-1}), U_n) \geq \frac{1}{4}$

Computational distinguishability

A distribution D on $\{0,1\}^n$ is $(C(n), \varepsilon(n))$-pseudorandom if for every machine A of "complexity" $C(n)$

$$\left| \Pr_{r \sim D}[A(r)=1] - \Pr_{r \sim U_n}[A(r)=1] \right| \leq \varepsilon$$

That is, restrict tests to low complexity "machines"

$C(n)$ = runtime of TM on n-bit inputs

or $C(n)$ = size of a circuit that takes n inputs

Defn: A pseudorandom generator

$G: \{0,1\}^s \longrightarrow \{0,1\}^n$ ($s = s(n)$ "seed length")

↳ (G stretches its input) $s < n$

is an $(C(n), \varepsilon(n))$-PRG if

$G(U_s)$ is $(C(n), \varepsilon(n))$-pseudorandom.

Food for thought: If $G: \{0,1\}^{s(n)} \to \{0,1\}^n$ is a $(n^3, \frac{1}{10})$-PRG, and G can be computed in $2^{O(s(n))}$ time, then a randomized alg A with runtime $O(n^2)$ (needing n random bits) and success prob. $\geq \frac{2}{3}$ can be derandomized into a deterministic alg. that runs in time $2^{O(s(n))} \cdot O(n^2)$.

How?

- Run A on $G(a)$ for every $a \in \{0,1\}^{s(n)}$ as its "random" string.
- Output majority answer.

Exercise: Prove that the above is a correct derandomization. $\left(\frac{2}{3} - \frac{1}{10} > \frac{1}{2}\right)$

In particular, if $s(n) = O(\log n)$ then get a deterministic polytime algorithm!

Pseudorandomness from hardness:

PRG $G: \{0,1\}^s \longrightarrow \{0,1\}^{s+1}$ $(n = s+1)$

$G(x) = \langle x, g(x) \rangle$ $\quad g: \{0,1\}^s \to \{0,1\}$

$\langle x, b \rangle$ $\quad b$ = random bit

If a test T fooled by $G(U_s)$ then T should not be able to predict $g(x)$ given random input x.

$$\Pr_{x \sim \{0,1\}^n}\left[T(x) = g(x)\right] \leq \frac{1}{2} + \varepsilon$$

[Pick a function $g(x)$ s.t no complexity $C(n)$ test T can predict $g(x)$ with accuracy $> \frac{1}{2} + \varepsilon$.

$\Longrightarrow$ G is a $(C(n), \varepsilon)$- PRG.

(require a small proof)

[Link between hardness & randomness]

"Something really hard to predict looks random"

Great, but there's a problem:

$G(x) = (x, g(x))$ ← hard to compute

Hmm, so how can G compute the last bit efficiently?

Two possible avenues:

1. Allow G more time than the tests it "fools". (Ok for derandomization application mentioned above, but NOT ok for cryptography)

② Twist the PRG construction to create some asymmetry that gives G an edge compared to the tests it fools

Few words about ②:

$h: \{0,1\}^s \to \{0,1\}$ (easy to compute)

~~$G(x) = (x, h(x))$~~

$G(x) = (\pi(x), h(x))$

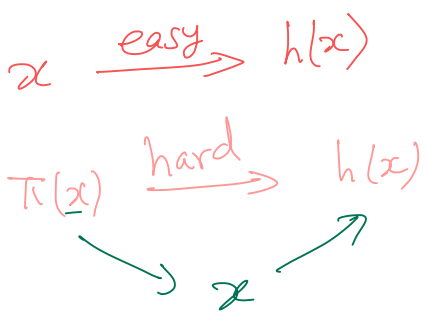
π : permutation on $\{0,1\}^s$

Easy to compute $x \mapsto \pi(x)$

Hard to invert.

(π: One-way permutation)

$h(x)$: hardcore predicate for one-way permutation π



Examples: (of one-way permutation & hardcore predicate)

(Conjectured)

- $\pi(x) = x^e \bmod N$ (RSA exponentiation)

  $x \in \mathbb{Z}_N^*$ $\quad h(x) =$ least significant bit of x $= lsb(x)$

Assumption: From $x^e \bmod N$, not only is it hard to decrypt x, but in fact can't even predict $lsb(x)$ better than 50-50.

- $\pi(x) = g^x \bmod p$ (p prime)

  (Inverting = discrete logarithm)

  $h(x) = 1$ if $x \in [0, \frac{p-1}{2}]$

  $\quad\quad 0$ if $x > \frac{p-1}{2}$ (like most sig. bit)

$x \in \{0, 1, \ldots p-2\}$

Comment: $lsb(x)$ is not hardcore for $\pi(x)$.

To get longer stretch, can iterate above hardcore predicate based construction repeatedly.

$x_0 \mapsto x_1 = \pi(x_0) \longrightarrow x_2 = \pi(x_1) \longrightarrow x_3 = \pi(x_2) \longrightarrow \ldots$

output $\longrightarrow$ $b_1 = h(x_0)$ $\quad b_2 = h(x_1)$ $\quad b_3 = h(x_2)$