Lecture #13: Derandomization

This is a huge topic and we will only scratch the surface.

Many directions to explore (see courses online).

Big Q:

if ∃ a randomised algo for a problem (say decision problem)
that takes poly time and is correct w.p. 3/4 (on all instances)

⇒ does ∃ a det poly time algorithm as well?

(is BPP = P ?)

P ← poly time (deterministic)

BPP ← Bounded error Probabilistic Poly time

Many directions include

- Randomness from hardness
  (if a function is hard ⇒ its outputs can be used to get pseudorandom generators for bounded machines)
  ("fooling" computationally restricted algorithms)
- Limited Independence
- Randomness Amplification using Expanders. (See notes from Expander lectures)
- Other ways of getting pseudorandom generators (Hash functions)
- Randomness Extractors (information-theoretic pseudorandomness)

Pseudorandom Generators

(for algorithms in class $\mathcal{A}$)

$G(r) \in \{0,1\}^M$ where $r \in \{0,1\}^m$ is a "random seed"

such that $G(r)$ looks indistinguishable from truly random $\{0,1\}^M$

to any test algorithm $T \in \mathcal{A}$.

i.e. $\left| \Pr_{r \in \{0,1\}^m}[T(G(r)) = 1] - \Pr_{R \in \{0,1\}^M}[T(R) = 1] \right| \leq \varepsilon. \quad \forall T \in \mathcal{A}.$

Note:

This means that if we have an algorithm that uses M bits of (true) randomness and belongs to a class $\mathcal{A}$. and succeeds w.p. $3/4$

Then we can use only m bits of randomness, feed into G,

and use it in our algo, succeed w.p. $3/4 - \varepsilon$.

And hence get a deterministic algorithm that runs in time 2^m.

(Indeed, try all 2^m choices of random seeds,

$3/4$ of them will succeed).

Good if seed length is small (say $m = O(\log n)$).

—— X ——

In fact may even consider specific algos. (A = your favorite algo).

E.g. want to solve the max-cut problem. (approximately)

Given $G=(V,E)$ undirected weighted graph. ($w_e \geq 0$)

want a subset S so that Edges from S to $V \setminus S$ are max.

i.e. max $w(E(S, V\setminus S))$.

maximize #edges cut

Thm: $\exists$ a cut s.t. $w(E(S,\bar{S})) \geq \frac{1}{2}\sum_{e\in E} w_e \geq \frac{1}{2}\,OPTimum$.

Pf: $\forall$ vertex $v \in V$, pick a random sign $\chi_v \in \{-1, 1\}$. u.a.r. indep.

Let $S = \{v \in V \mid \chi_v = +1\}$.

Claim: $\mathbb{E}(w(E(S,\bar{S}))) \geq \frac{1}{2}\cdot\sum w_e$.

Pf: Consider edge $e = \{u, v\}$.

$$Pr(\chi_u \neq \chi_v) = \frac{1}{2}.$$

$\Rightarrow \mathbb{E}[\text{contribution of } e \text{ to cut}] = \frac{1}{2}\cdot w_e$.

Now use linearity of expectations!

☺

$\Rightarrow$ A random cut has weight $\frac{1}{2}\sum w_e$ in expectation

$\Rightarrow \exists$ cut of weight $\geq \frac{1}{2}\sum w_e$.

☺

Exercise: repeat multiple times (say $O(\frac{1}{\varepsilon}\log n)$ times) and return best of all these — will get a cut with $(\frac{1}{2}-\varepsilon)m$ edges w.p. $1-1/\mathrm{poly}(n)$.

Here is one way to derandomize this algo.

Define a distribution over n elements $(X_1, X_2 \ldots X_n)$ to be pairwise independent (PI) if:-

$$Pr(X_i = a \wedge X_j = b) = Pr(X_i = a)\, Pr(X_j = b) \quad \forall i \neq j, \forall a, b.$$

E.g. if the distribution is a "product" distribution
ie if all $X_1, X_2 \ldots X_n$ are independent
$\Rightarrow$ clearly PI.

Supported on all 8 strings in $\{0,1\}^3$

(Eg. flip n coins indep. then resulting bit string is Product distrib $\Rightarrow$ PI)

vs.

But: another example:—

X_1	X_2	$X_3 = X_1 \oplus X_2$
0	0	0
0	1	1
1	0	1
1	1	0

supported on 4 strings from $\{0,1\}^3$ = 8 elements

Not a product distribution (X_3 clearly known if X_1, X_2 known)

but any 2 columns look independent!

Q1: how to generate PI distributions?

Q2: What can we do with PI distributions?

One Construction for PI bit strings

Consider the finite field $\mathbb{F}_{2^b}$

it has a natural correspondence with b-bit strings. (has 2^b elts)

Now: pick random $\alpha, \beta \in \mathbb{F}(2^b)$ ← denote this as $\mathbb{F}$ for compactness.

Define $X_z = \alpha z + \beta$ (all arithmetic in $\mathbb{F}(2^b)$)

this produces $X = (X_z)_{z \in \mathbb{F}}$

which is joint distribution over 2^b elements of $\mathbb{F}$.

———— x ————

Claim: $\Pr(X_{z_1} = a_1 \wedge X_{z_2} = a_2) = \Pr(X_{z_1} = a_1) \cdot \Pr(X_{z_2} = a_2)$ $\forall z_1 \neq z_2$ a_1, a_2.

Pf: LHS = event $\left\{ \begin{pmatrix} z_1 & 1 \\ z_2 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \right\}$

but if $z_1 \neq z_2$ the matrix is invertible and

$$= \left\{ \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} z_1 & 1 \\ z_2 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \right\}$$

but α, β chosen u.a.r. from $\mathbb{F}$.

⇒ the prob they take the values on RHS $= \frac{1}{|\mathbb{F}|^2}$.

$= \Pr(X_{z_1} = a_1) \Pr(X_{z_2} = a_2)$. ▨

trivial Vandermonde matrix ⇒ suggests might get k-wise indep

———— x ————

So distribution is P.I.

Any pair of columns looks independent. (unif over $\mathbb{F}(2^b)$).

Could just take the 1st bit (or last bit or any bit) of X_z's to get PI distribution over $\{0,1\}^{2^b} = \{0,1\}^n$ if $n = 2^b$ $\Leftrightarrow \log_2 n = b$

But size of distribution is $2^{2^b}! = 2^{2\log n}$

$= n^2$.

$\Rightarrow$ Distribution supported over n^2 bit strings of length n which is PI.

$\Updownarrow$

Algorithm to convert $2\log n$ bits into n bits that are PI

$\uparrow$

truly random "seed"

Why useful?

Analysis for max cut works even if colors of vertices are pairwise independent

(Indeed — all we did was look at $\chi(v_1) = \chi(v_2)$ event)

$\Rightarrow$ take all possible seeds (enumerate over them)

One must give a cut of size $\geq \frac{1}{2} w(E) = \frac{1}{2} \sum_{e \in E} w_e$.

$\Rightarrow$ can derandomize algo in $O(n^2)$ iterations!

Aside: another way of derandomizing the max-cut algorithm is called "Method of Conditional Expectations".

Uses the fact that best outcome $\geq$ expectation. ☺

This is too vague, let me explain.

Suppose we assign colors χ to vertices one by one. (completely independently)

Know that

$$\underset{\chi(1)}{\mathbb{E}}\,\underset{\chi(2)}{\mathbb{E}}\cdots\underset{\chi(n)}{\mathbb{E}}[\text{weight of edges cut}] \geq m/2.$$

$$= Y_{2:n} \text{ (say)}$$

Look at first choice.

$$\underset{\chi(1)}{\mathbb{E}}\, Y_{2:n} \geq m/2.$$

but LHS $= \frac{1}{2}\cdot Y_{2:n}\Big|_{\chi(1)=+1} + \frac{1}{2} Y_{2:n}\Big|_{\chi(2)=-1} \geq m/2$

At least one of these choices must be $\geq m/2$. (may be both)

So set $\chi(1)$ to be that choice and continue.

Need: to be able to compute $Y_{2:n}$ conditioned on some value of $\chi(1)$.

↑ compute conditional expectations

↑ hence the name!

In general. Suppose fixed colors of $1, 2 \ldots i$

Want to compute $E[$weight of edges cut if rest of vertices picked uniformly at random$]$

$= \sum_{e \text{ edges within } \{1 \ldots i\}}$ edges cut $+ \sum_{e: \text{ one end in } \{1 \ldots i\}, \text{ another end in } \{i+1 \ldots n\}}$ Pr(free end gets opposite color) $\cdot$ wt $\leftarrow \frac{1}{2}$

$+ \sum_{e: \text{ both ends in } \{i+1 \ldots n\}}$ Pr(get different colors) $\cdot$ wt $\leftarrow$ also $\frac{1}{2}$

$=$ weight of edges already cut for sure

$+ \frac{1}{2} \cdot$ weight of edges that have one end in $\{i+1 \ldots n\}$ or both ends.

Can use this to choose color of vertices one by one,

ensure that current expectation $\geq m/2$.

$\Rightarrow$ Finally end at solution with cut value $\geq m/2$.

Actually results in greedy algorithm ☺

In this case resulting algo was easy (in retrospect) ← makes for good example in class ☺

But other cases can be quite non-trivial

Rest of Lecture

Get randomness from "hardness"

if a function is hard (in some sense)

maybe its output looks difficult to compute (for some class of algos)

And maybe we can use it to look "random".

Again, vague. Let's be concrete

Hardness from Cryptography

Suppose a function is hard to invert ("one way" function) for some class of algos

then we show that can use to get pseudorandom bit strings (for that class of algos) !

Technically: will show [Håstad Impagliazzo Levin Luby]

"one way permutations exist" $\Rightarrow$ "pseudorandom generators exist" (PRGs)

can do more:

"one way functions" $\Rightarrow$ PRGs

[see crypto class or textbook]

One Way Permutations:

Let $f: \{0,1\}^* \to \{0,1\}^*$ be a function such that

$$|f(x)| = |x|$$

and f restricted to $\{0,1\}^n$ be a permutation on $\{0,1\}^n$.

- Define the success probability of A be

$$\delta(n) = \Pr_{x \in \{0,1\}^n}[A(f(x)) = x]$$

- Say that f is $s(n)$ secure if for any algorithm A,

$$\frac{\text{runtime}(A(n))}{\delta(n)} \geq s(n). \qquad \forall n.$$

Loosely: any efficient algo (s.t. runtime is small) must have small probability of inverting f.

We'll show that: if you can distinguish the output of the PRG from a truly uniform distribution (efficiently)

$\Rightarrow$ can invert a one-way $\overset{\text{perm}}{\uparrow}$ efficiently with good prob

(which is not possible)

Basic construction of PRGs from OWPs

Pick random seed x

random string r

} both of length n.

f: one way perm.

output $f(x), r, \langle x \cdot r \rangle$

$\langle x \cdot r \rangle = \sum_i x_i r_i \bmod 2$.

$2n+1$ bits

Main Technical Statement:

Intuition: if after seeing $f(x)$ and r, A can predict $\langle x, r \rangle$ w.p. $\frac{1}{2}+\varepsilon$

$\Rightarrow$ can invert f with reasonable prob $\mathcal{R}(\varepsilon)$.

$\Rightarrow \langle x, r \rangle$ "looks random" to algo A ← See Luca's notes or other textbooks for formal proof

---

Only stretched by 1 bit so far.

Can extend the idea:-

Output:

$$r, \langle x, r \rangle, \langle f(x), r \rangle, \langle f^{(2)}(x), r \rangle, \langle f^{(3)}(x), r \rangle \ldots \langle f^{(t)}(x), r \rangle$$

$n + t + 1$ bits.

$f^{(t)}(x) = f(f(f \cdots f(x)))$

The Goldreich-Levin Hidden Bit Theorem.

$f: \{0,1\}^n \to \{0,1\}^n$ permutation

Suppose we have an algo. A that, given (a) $f(x) \in \{0,1\}^n$ and (b) "random" bitstring $r \in \{0,1\}^n$.

$\langle a, b \rangle = \sum_i a_i b_i \bmod 2$

achieves

$$\Pr_{x,r}(A(f(x), r) = \langle x, r \rangle) \geq \tfrac{1}{2} + \varepsilon$$

then there exists an algorithm B ("breaker") that can achieve

$$\Pr_{x}(B(f(x)) = x) \geq \Omega(\varepsilon).$$

In other words, if we given $f(x)$ and r, we can guess $\langle x, r \rangle$ slightly better than random (on random x, r)

then we can invert $f(x)$ on some non-trivial fraction of bit strings x.

In other other words, if we assume that f is hard to invert (efficiently) permutation ("one-way perm.")

then given $f(x)$ and r, the bit string $\langle x, r \rangle$ still looks random! (to comp. bdd. algos)

So: pick $x, r \in \{0,1\}^n$. 2n random bits

output $(f(x), r, \langle x, r \rangle) \in \{0,1\}^{2n+1}$

pseudorandom string.

looks indistinguishable from a random $2n+1$ bit string to any comp.-bounded algorithm

GL Version 0:

instead of $\frac{1}{2}+\varepsilon$

- if $\exists A$ s.t. $\Pr_{x,r}[A(f(x), r) = \langle x, r\rangle] = 1$.

  $\Rightarrow$ can invert f (i.e. $\exists B$ s.t. $B(f(x)) = x$ w.p. 1).

  using n queries.

  & poly time.

Pf: By assumption

$A(f(x), r) = \langle x, r\rangle \quad \forall x, r.$

$\Rightarrow$ ask $r = e_i \ \forall i$ (n queries)

$\Rightarrow A(f(x), e_i) = \langle x, e_i\rangle = x_i \quad \forall i$

$\Rightarrow$ output $x = (x_1, x_2 \dots x_n)$.

$\square$

GL Version 1:

instead of $\frac{1}{2}+\varepsilon$ or 1.

- if $\exists A$ s.t. $\Pr_{x,r}(A(f(x), r) = \langle x, r\rangle) \geq \frac{15}{16}$

  $\Rightarrow \exists B$ s.t. $\Pr_x(B(f(x)) = x) \geq \frac{1}{2} - \frac{1}{\text{poly}(n)}$

  using poly n queries.

  runs in poly time

Pf: cannot do the same thing. ☹

But want to simulate asking $\langle x, e_i\rangle$ again.

OK: call x good if $\Pr_r(A(f(x), r) = \langle x, r\rangle) \geq \frac{7}{8}$ ← for that fixed x.

Claim: $\Pr_x(x \text{ good}) \geq \frac{1}{2}$.

Pf: if x bad $\Rightarrow \Pr_r(\text{—}) < \frac{7}{8}$. if x good $\Rightarrow \Pr_r(\text{—}) \leq 1$.

$\Rightarrow$ if $\Pr(x \text{ good}) = p \Rightarrow p \cdot 1 + (1-p)\cdot\frac{7}{8} \geq \frac{15}{16} \Rightarrow p \geq \frac{1}{2}$. $\square$

Great. so we'll try to invert f for good x's. (which ~~are~~ are $\frac{1}{2}$ of space).

for such an x: $\Pr_r\big(\underbrace{A(f(x),r)}_{H(r)} = \langle x, r\rangle\big) \geq 7/8$.

Want to learn x.

Invert "good" x.

OK: note: if $H(r_1) = \langle x, r_1\rangle$, $H(r_2) = \langle x, r_2\rangle$ $\Rightarrow$ can compute $\langle x, r_1 + r_2\rangle$!

So pick a random $r \in \{0,1\}^n$.

Ask: $H(r)$ and $H(r+e_i)$

and hope that $\hat{H}_i(r) = H(r+e_i) - H(r)$.

$$\Pr_r\big(\hat{H}_i(r) = \langle x, e_i\rangle\big) \geq \Pr_r\big(H(r+e_i) = \langle x, r+e_i\rangle \wedge H(r) = \langle x, r\rangle\big)$$

$$\geq 1 - \Pr_r\big(H(r+e_i) \neq \langle x, r+e_i\rangle\big) + \Pr_r\big(H(r) \neq \langle x, r\rangle\big)$$

$$\geq 1 - \left(\tfrac{1}{8} + \tfrac{1}{8}\right) = 3/4.$$

$\Rightarrow$ correct value of x_i w.p. $3/4$!

$\Rightarrow$ now take $k = O(\log n)$ repetitions (random values r). $r_1, r_2 \ldots r_k$

And take majority vote $\mathrm{maj}\big(\hat{H}_i(r_1) \ldots \hat{H}_i(r_k)\big)$.

Exercise: correct w.p. $1 - 1/\mathrm{poly}(n)$.

Do this for all n bits of x. $\Rightarrow$ recover x !

Great: GL version 1 looks great.

How hard can it be to get down to $\frac{1}{2}+\varepsilon$? ☺.

Barrier: $\exists$ functions H s.t.

$$\Pr_r(H(r) = \langle x_1, r\rangle) = 3/4$$

and $$\Pr_r(H(r) = \langle x_2, r\rangle) = 3/4$$

for $x_1 \neq x_2$!

So hopeless to reduce $15/16$ to below $3/4$ ☹.

↳ how does this work?

say $x_1 = 0$ (one bit)

$x_2 = 1$.

Can't be sure if $x = 0$ or 1. ☹

Thm: define

$H(r) = 0$ if $r = 0$

and $\text{Ber}(\frac{1}{2})$ if $r = 1$

———— X ————

What next?

- How to avoid this problem?
- Idea: "List decoding"

GL (Decoding) Lemma: Suppose we have function H such that for unknown $x \in \{0,1\}^n$:

$$\Pr_r(H(r) = \langle x, r\rangle) \geq \frac{1}{2}+\varepsilon$$

$\Rightarrow$ can create a list L of $O(1/\varepsilon^2)$ strings $\in \{0,1\}^n$ such that one of these strings is x. w.p. $\frac{1}{2}$.

(does so by log. poly $(1/\varepsilon)$ queries and runs in poly time)

Claim: GL Decoding Lemma $\Rightarrow$ GL Theorem.

Pf sketch: if $\Pr_{x,r}(A(f(x),r) = \langle x,r\rangle) \geq \frac{1}{2}+\varepsilon$.

$\Rightarrow$ define x "good" if $\Pr_r(\underbrace{A(f(x),r)}_{H(r)} = \langle x,r\rangle) \geq \frac{1}{2}+\frac{\varepsilon}{2}$.

can show (same idea as before) that $\Pr_x(x \text{ is good}) \geq \varepsilon/2$.

$\Rightarrow$ now focus on good x's and decode them w.p. $\frac{1}{2}$. $\Rightarrow \Pr(\text{invert } f) \geq \varepsilon/4$. ▨

Great: Now want to prove Lemma.

(if $\Pr_r(H(r) = \langle x,r\rangle) \geq \frac{1}{2}+\varepsilon$
$\Rightarrow$ can List decode x w.p. $\frac{1}{2}$).
small List of candidates.
pick mine.

Where did previous proof fail?

- we said $\Pr(H(r+e_i) - H(r) = \langle x, e_i\rangle) \geq 1 - \Pr(\text{mistake on } r+e_i) - \Pr(\text{mistake on } r)$
$\geq 1 - (\frac{1}{8}+\frac{1}{8}) = 3/4$.

So got x_i correctly w.p. $> \frac{1}{2}$. repeat & take majority.

- But now: $\Pr(\quad) \geq 1 - (\frac{1}{2}-\varepsilon + \frac{1}{2}-\varepsilon) = 2\varepsilon$. ☹.

Not high enough! Better to just flip a coin!

one query is always correct!

But idea: suppose know the value of $\langle x,r\rangle$ correctly without asking $H(r)$.

then $\Pr(H(r+e_i) - \langle x,r\rangle = \langle x, e_i\rangle) \geq 1 - (\frac{1}{2}-\varepsilon) = \frac{1}{2}+\varepsilon$. ☺

Can repeat and get higher confidence. But: need different r's, need value of $\langle x,r\rangle$

how do we know $\langle x,r\rangle$?? 😐

Attempt 1 (using these free queries that are always correct!)

- Pick $k = O(1/\varepsilon^2)$ random strings $r_1, r_2 \ldots r_k$.
- Suppose know: $b_i = \langle r_i, x \rangle$.

big assumption!

- Define

$$G_i(r) = H(r + r_i) - b_i$$

- Observe: $G_i(r) = \langle x, r \rangle$ wp $\frac{1}{2} + \varepsilon$.

Indeed: if $H(r + r_i) = \langle x, r + r_i \rangle \Rightarrow G_i(r) = \langle x, r + r_i \rangle - \langle x, r_i \rangle$
$= \langle x, r \rangle$.

- **Claim:** Let $G(r) = \underset{i \in [k]}{\text{majority}}\ G_i(r)$.

then $\Pr_{r, r_1, r_2 \ldots r_k} [G(r) = \langle x, r \rangle] \geq \frac{31}{32}$.

Pf: Let $Z_i = \mathbb{1}(G_i(r) = \langle x, r \rangle) \Rightarrow EZ_i = \frac{1}{2} + \varepsilon$.

$$\Pr\left(\textstyle\sum_i Z_i < k/2\right) \leq \Pr\left(\left|\sum_i Z_i - E\sum_i Z_i\right| \geq \varepsilon k\right) \leq \frac{\mathrm{Var}(\sum_i Z_i)}{\varepsilon^2 k^2}$$

But $\mathrm{Var}(\sum_i Z_i) = \sum_i \mathrm{Var}(Z_i) \leq \sum_i E(Z_i) \leq (\frac{1}{2} + \varepsilon) k \leq k$
(↑ indep.)

$\Rightarrow \Pr(\text{mistake}) \leq \frac{k}{\varepsilon^2 k^2} \leq \frac{31}{32}$ if $k \geq$ constant $\cdot 1/\varepsilon^2$.

▨

We'll come back to this claim soon!

(Hint, Hint...)

Now: usual trick

$$\Pr_{r_1, r_2 \ldots r_k}\left[G(r) = \langle x, r\rangle \right] \geq \frac{31}{32}$$

$$\Rightarrow \Pr_{r_1, \ldots r_k}\left[\Pr_{r}\left[G(r) = \langle x, r\rangle\right] \geq 7/8 \right] \geq \frac{3}{4}.$$

$\Rightarrow$ if we choose $r_1, r_2 \ldots r_k$ u.a.r. then with good prob the estimator we have satisfies the conditions of GL v1's main Lemma.

$\Rightarrow$ can recover x wp $1 - 1/\text{poly}(n)$. !

— x —

All this is great, but what about the "know $b_1, b_2 \ldots b_k$" part ?!?

This is why it is attempt #1. ☺

We'll "guess" them. Basically enumerate over all choices of $b_1 \ldots b_k$.

"Only" $2^k = 2^{O(1/\varepsilon^2)}$ choices.

One is good. Will return the right x.

But List has size $2^{O(1/\varepsilon^2)}$. ☹

Algo has time $2^{O(1/\varepsilon^2)}$ ☹

Not what we promised!

On to v2.

Q: why did we lose so much?

Because of guessing. all these choices.

Brilliant Idea:

Use pairwise independence!

Only used Chebyshev for the main claim.

But that works even when r_i, r_j are pairwise independent. !

—— X ——

Fix: (~~Assume~~ 2)

Pick $k' = \log \frac{1}{\varepsilon^2}$ different $r_1, r_2 \ldots r_{k'}$

Guess what $\langle x, r_i \rangle = b_i$ values are.

Now: define for each set S of $2^{k'}$ indices of $\{1, 2 \ldots k'\}$.

$$r_S = \sum_{i \in S}' r_i \qquad b_S = \sum_{i \in S}' b_i$$

Facts: . if $\langle x, r_i \rangle = b_i \ \forall i \Rightarrow \langle x, r_S \rangle = b_S$.

. Also if $S \neq T$ then r_S, r_T independent of each other.

Do above algo with these random choices $r_1, r_2 \ldots r_{k'}$

and enumerate over all guesses

$b_1, b_2 \ldots b_{k'}$

Runtime is now $2^{k'} = O(1/\varepsilon^2)$.

And so is list.

Yay! ☺

To summarize

- PRGs : pseudorandom generators.

  take truly random seed

  produce strings that "look" random to any computationally bounded adversary/algo.

- Pairwise Independent distributions. (and k-wise Indep)

———X———

(1) If Algo only uses pairwise indep arguments,

can use a PI distribution to derandomize.

(2) Aside: some algos can be derandomized using the method of Conditional Expectations

(3) But a general approach is: "hardness" ⇒ "randomness"

E.g. if assume one way functions / permutations

⇒ PRGs.

via the Hidden Bit Theorem (a.k.a. Learning Algo) Goldreich Levin

———X———