Definition (Recall)

Turing machine model:

- Finite set of states
- Finite set of symbols
- Finite set of 1-dimensional tapes
- Finite set of transition rules

Universal model of computation:

1. In two senses:
   a. There exists a universal TM, \( U \), such that for any TM, \( M \), and input \( x \),
      
      \[ U(M, x) = M(x) \]

   and runtime of \( U(M, x) \) is \( O(r(M, x)) \),

      where \( r(M, x) \) is runtime of \( M(x) \).

   b. For any "reasonable" model of computation,
      
      a program \( f \),
      
      \[ U(f, x) = f(x) \]

      and runtime of \( U(f, x) \) is \( O(r) \), where \( r \) says:

      \[ O(r) \]

Hence finding another way to simulate a TM is as good as standard

models such as Random Access Machines (RAM) are equivalent up to polynomial of transformations.

Also, changing alphabet size / number of tapes makes a difference in runtime of at most a constant factor / logarithmic factor.

E.g., going from 1 to 2 tapes or alphabet size makes a factor of \( 2 \log \) factor. [Intuition: one in 0/1's to encode one symbol, twice as many steps.]

Slightly harder: converse holds too: If \( M \) has alphabet size \( m = 2^{11} \),

then using an alphabet size \( m = 2^{\frac{11}{2}} \), gives a speedup of factor \( \frac{2^{11}}{2^{\frac{11}{2}}} \).
$N = \{0, 1, 2, \ldots \}$

**Definitions**

Let $\mathbb{N} \to \mathbb{N}$. 

$\text{DTIME}(T) = \{ L : \text{for some TM } M, M \text{ decides } L \text{ and} \\
\text{runs in time } O(T(n)) \}$. 

**Where:** $L \subseteq$ language is any subset of $\{0, 1\}^*$. 

$M \text{ decides } L \iff \forall s, M\text{ accepts } s \iff s \in L, \text{ and} \\
M \text{ rejects } s \iff s \notin L.$

In particular, $M$ halts on all inputs or (!) 

$M \text{ runs in time } T(n) \iff \exists c > 0 \text{ s.t. for all } s,$ 

$M(s), \text{ halts in time } \leq c \cdot T(|s|).$

**Comments:**

- $M$ can be a $k$-tape TM, for any $k \geq 1$.
- $O(T(n))$ is used, so constant factor speedups/slowdowns are irrelevant
- Letting number of $k$ of tape may vary is optional, but standard.
- We use “languages” or “decision problems” as our central notion. Sometimes we are interested in the DTIME complexity of functions too — this is defined in the obvious fashion.
- Runtime bounded by a function of the length of the input. (<=)

**Define** $P = \bigcup \text{DTIME}(n^k)$, “Polynomial time”

**Remark:** This is how it is usually stated, but it really means

$\bigcup \text{DTIME}(\text{max}(n^k, 1)).$

Small mistake at length $n=0, n=1$ should be tolerated without comment.
Remark: By earlier comments, P is the same class whether defined by Turing machine, or a more realistic model such as RAMs.

Remark A function \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) is polynomial time computable if:

(a) \( f \) has polynomial growth rate, \( \forall n, \; |f(n)| \leq n^c + c \) for some \( c \)

(b) The product \( \{<i, s> \text{ s.t. } i \text{th bit of } f(i) \text{ is symbol } s\} \) is in P.

The class of polynomial time computable functions is denoted \( \text{P} \), or sometimes just \( P \).

History Cobham, 1964 & Edmonds, 1965 - gave original definitions of \( P \) in P and highlighted the importance of \( P \) corresponding to feasible computability.

Other remarks: We typically use well-defined \( T(n) \)'s to bound runtime.

E.g. \( T(n) \geq n \)

\( T(n) \) is nondecreasing

Also, technical condition: \( T(n) \) is time-constructible.
Space bounded computation

Interestingly, \( \text{Space} = \# \text{of tape squares visited during the computation} \).

A language \( L \) is in \( \text{SPACE}(s(n)) \) if there is a \( \text{TM} \) \( M \) that accepts \( L \) and uses space \( \text{Space}(n) \leq O(s(n)) \).

Small catch, we want to allow \( s(n) = \log n \) so as to define \( \text{LOGSPACE} \),
but the input uses space \( n \neq O(\log n) \).

Solution: Ignore the size of the input.

Definition: For space-bounded TM's: the input tape is read-only, work tape (and optionally the output tape) are read-write, only space on read-write tapes is counted in the space used.

Definition: Let \( S: N \rightarrow N \), \( S(n) \geq \log n \), \( s(n) \) "well-behaved" (non-decreasing, space-computable).

\[ \text{SPACE}(S) = \{ L \subseteq \{0,1\}^* : \text{for some TM } M, M \text{ decides } L \text{ and } M(s) \text{ uses space } \Theta(S(101)) \text{ for all inputs} \} \]

Comments: Page 2 comments all apply again.

- If \( s(n) \geq n \), the input tape/work tape distinction is unnecessary.
- If \( s(n) < \log n \), weird things can happen.

Usual definition: \( f: \{0,1\}^* \rightarrow \{0,1\}^* \) being computable in space \( S(n) \) is:

\[ \forall n, |f(n)| \leq 2 \]

and \( \{ (i, f(i), s) : \text{ith bit of } f(i) \text{ is } s \} \in \text{SPACE}(S(n)) \).

Note here: \( |i| = O(S(101)) \).

- It is usual to use \( \text{SPACE}(S) \) but \( \text{DTIME}(T) \).

Definition: \( \text{PSPACE} = \bigcup_{c \geq 1} \text{SPACE}(n^c) \) \( \text{LOGSPACE} = \text{SPACE}(\log n) \) \( \text{Poly Log-Space} = \bigcup_{c \geq 1} \text{SPACE}(\log^c n) \).
Theorem: \( \text{DTIME}(T(n)) \subseteq \text{SPACE}(T(n)) \)

\[ \text{Pf: This is obvious.} \]

Theorem: \( \text{SPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))}) \left(= U_c, \text{DTIME}(2^{c \cdot S(n)}) \right) \)

\[ \text{Pf: A configuration of a T.M. is the a complete state of its} \]

- Input tape head position: \( O(\log n) \) bits
- Input tape contents: \( O(\log n) \) bits
- Work tape contents: \( O(c \cdot S(n)) \) bits
- Current tape head position of 1st work type: \( O(\log(S(n)) + \log c) \) bits
- Same for 2nd type \( \ldots \) \( k \) th \( \ldots \) \( \text{fixed} \)
- Current state: \( O(1) \) bits

Total description of configuration is: \( O(\log n) + O(\log S(n)) + O(c \cdot S(n)) + O(1) \)

\[ \text{In particular, this is } < 2^{O(\log)} \]

since each is described with only \( O(\log) \) many bits.

Consider \( M \) that decides \( L \) in \( \text{SPACE}(S(n)) \), and consider a fixed input \( x \).

Claim: \( M(x) \) enters each configuration at most once.

\[ \text{Pf: } x_L \text{, since } M \text{ is deterministic, it would be in a loop and never halt.} \]

Hence \( M(x) \) runs for time \( < (\text{# of configurations}) = 2^{O(S(n))} \).

\[ \text{QED.} \]
\[ \text{Culley:} \quad \text{LOGSPACE} \subseteq P \subseteq \text{PSPACE} \subseteq \text{\#P} \subseteq \text{EXP} \subseteq \text{\#EXPTIME} = \text{\#EXP} \subseteq \text{EXPTIME} \subseteq \text{EXP} \subseteq \text{\#P} \subset \text{PSPACE} \subset \text{LOGSPACE} \]

Non-deterministic Computation

Definition of NP #1

Here we view P as the class of feasibly computable problems.

NP is the class of feasibly verifiable problems.

Definition: NP = \{ L \subseteq \{0,1\}^* : \exists \text{ a predicate (language) } M \in \text{ P such that}

\exists p(n), s.t.

\forall x, \text{ s.t. } L \subseteq \{0,1\}^n \Rightarrow \exists u \in \{0,1\}^p(|x|) \text{ s.t. } M(x,u) \}

M can be thought of as a language in P or equivalently as a polynomial time bounded TM.

When M(x,u), 1\leq p(|x|) holds, u is called a certificate or witness for x \in L.

Example: Calculating problems with known length solutions.

P \iff computing a solution.

NP \iff searching a solution is harder.

Examples:

1. Factoring: \[ L = \{ x \in \mathbb{N}, \langle x, l, u \rangle, x \leq 2^u, \text{ s.t. } x \text{ has a prime factor } p \leq 2^u \} \]

If you could solve this decision problem efficiently, then you could factor integers efficiently by using binary search.

Note: \( L \in \text{NP} \).

Open: Is \( L \in \text{NP} \)? (Is Factoring \( \in \text{P} \)?)

Is \( L \) \( \in \text{NP} \) complete?
(2) Graph Isomorphism: \( L_2 = \{ (G_1, G_2) : G_1 \cong G_2 \} \)

Inputs \( G_1, G_2 \) are coded as binary strings in some straight-line way, e.g., by incidence matrix or 10,11. \( G_1, G_2 \) -undirected graphs.

\( L_2 \in NP \) (Take \( u \) to be an explicit isomorphism.)

Open: is \( L_2 \in P \). Is \( L_2 \) \( NP \)-complete?

(3) Traveling Salesman Problem:

\( L_3 = \{ \text{Graph } G = (V, E), f : E \rightarrow \mathbb{N}, k \in \mathbb{N} : \text{a circuit visiting each vertex exactly once with total weight } \leq k \} \)

\( L_3 \in NP \) \( L_3 \)-NP-complete.

(4) Theorem of Set Theory ZF:

\( L_4 = \{ \langle \psi, \bar{0}^k \rangle : \psi \text{ is a ZF-formula} \) and has a proof of \( \leq k \) symbols \}

\( L_4 \in NP \) \( L_4 \)-NP-complete.

Open: is \( L_4 \in P \).

Theorem: \( P \leq NP \).

Pf: \( M \) ignores \( u \), and just solves \( L \).

Theorem: \( NP \leq PSPACE \).

Pf Algorithm for \( L \):

For each \( u \in \{0,1\}^n \)

if \( M(\bar{u}, u) \) accepts, \( \text{halt and accept} \)

else \( \text{halt and reject} \).

Space used: \( \text{Space to store } u \), i.e., \( O(p(n)) \)

+ Space to run \( M(\bar{u}, u) \), which is \( \leq \text{Time to run } M(\bar{u}, u) \).

Conclude: \( NP \leq \text{EXPTIME} \).
Non-deterministic computation

NTM (Non-deterministic TM)

Like a TM but has two transition functions, i.e., at each configuration, there are two possible moves based on currently read symbol.

\[ S_0 \left( q_0, \delta, \ldots \delta \right) = \left( q_1, \alpha, \ldots \alpha \right) = \delta_{i} - state \]

\[ S_1 \left( \ldots \ldots \right) = \left( \ldots \ldots \right) \]

Comment: Can allow an option in some states, by setting \( S_0 = S_1 \).

Can simulate any number of options, by repeated choices among two (any number of local changes to the configuration).

Define: \( M(x) \) accepts, \( M \) "accepts" \( x \), or \( M(x) = 1 \), means, there is some possible computation of \( M(x) \) that leads to the accept state.

If all possible computation paths lead to rejection, \( M \) rejects \( x \), \( M(x) = 0 \).

\( M \) runs in time \( T(n) \) if \( \forall \delta, M(\delta) \) halts in \( \leq T(n) \) steps, \( n = |\delta| \), on all possible computation paths.

Wlog, \( M(\delta) \) always halts, since we can just cut-off it's computation after \( T(n) \) steps. (Provided \( T(n) \) is time constructible)

Define: \( \text{NTIME}(T(n)) = \{ L \subseteq \{0,1\}^* : \exists c, \text{NTM M, } M \text{ runs in } c \cdot T(n) \text{ steps on all inputs, and } L \subseteq \{0,1\} \} \)

Define (2nd alternative)

\( N \) = \( \cup \text{NTIME}(n^p) \).

Discussion: \( \text{NTIME} \) machines never need to have a definite "reject" condition.
The two definitions are equivalent.

**Pf:** Suppose \( L \) satisfies the first definition. NTM algorithm \( M \):

```
Input \( w \):
"Guess" \( u \) by writing out string \( \rho(10) \) many 0/1's on \( \text{tape} \).
Run \( M(x,u) \).
Accept if \( M(x,u) \) accepts.
```

Suppose \( L \) satisfies the second definition.

Accepted by \( M \) runs in \( \text{poly}(n) \) steps.

Let \( u \in \{0,1\}^{\text{poly}(n)} \) indicate \( M \)'s nondeterministic choices.

\( \text{i} \)-th bit = \( \{0,1\} \) union \( \{0,1\} \) where \( \text{i} \)-th step \( M(0) \) uses \( \{0,1\} \).

\( M'(x,u) \) runs \( M \) deterministically choosing at each step to use \( \{0,1\} \) or \( \{0,1\} \), depending on \( \text{i} \)-th bit of \( M \).

\( \text{g.e.d.} \)

Then \( \text{NTIME}(T(n)) \subseteq \text{SPACE}(T(n)) \).

**Pf:** Let \( L \in \text{NTIME}(T(n)) \), run time \( \leq c \cdot T(n) \) on NTM \( M \).

SPACE\( (T(n)) \) algorithm \( M' \) on \( L \):

```
Input \( w \), \( 10/w \)
Loop, for each \( u \in \{0,1\}^{c \cdot T(n)} \):

Run \( M(0) \) deterministically for \( c \cdot T(n) \), zero bits \( \forall u \).
Select \( \{0,1\} \) transitional rule.
If \( M(0) \) accepts, halt and accept.
Else: continue w/ next \( u \).
End loop.
```

Halt or reject.
Defn. $\text{NSPACE}(S(n)) = \{ L \subseteq \{0,1\}^* : \text{For some NTM } M, M \text{ runs in space } \leq S(n) \text{ for all input } x, \text{ halt, } L = L(M) \}$. 

Defn. $L(M) = \{ \sigma : M \text{ accepts } \sigma \} = \{ \sigma : M(\sigma) = 1 \}$.

Defn. $\text{NL} \subseteq \text{NSPACE}(\log n) \subseteq \text{NSPACE}(n)$, nondet. in log space.

Thm. $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})(= \cup_{c \geq 1} \text{DTIME}(2^{c \cdot S(n)}))$.

PF: Let $L \in \text{NSPACE}(S(n))$, via NTM $M$.

Algorithm for $L$: Input $x$, halt in $S(n)$

Loop, for all $u \in \{0,1\}^{O(S(n))}$

- Run $M(x,u)$, for $2^{O(S(n))}$ steps (using alg. 6.6a on page 5)

- If $M(x,u)$ accept, halt & accept.

- Else, reject.

Runtime: $2^{O(S(n)) \cdot 2^{O(S(n))}} = 2^{O(S(n)^2)}$.

QED.

Thm. $\text{SPACE}(S(n)) \subseteq \text{NSPACE}(S(n))$

PF: Obvious!
Savage's Theorem: Let \( S(n) \geq \log n \) be "well-behaved" (space-constructible).

Then \( \text{NSPACE}(S(n)) \subseteq \text{SPACE}(S(n)^2) \).

[Waltz Savage '70]

**Pf:** Let \( L \in \text{NSPACE}(S(n)) \) be accepted by NTM \( M \) in space \( c \cdot S(n) \).

For a fixed input \( \sigma \), let \( n = |\sigma| \).

\( M \) has \( 2^{2 \cdot d \cdot S(n)} \) configurations, each describable with \( d \cdot S(n) \) bits.

We shall show how to compute

\[
R_m(\sigma, C_0, C_1, t) := \begin{cases} C_0, C_1 \text{ if } t \leq n \\ C_0, C_1, t \text{ if } t > n \end{cases}
\]

and configurations of \( M \) on input \( \sigma \),

\( t \in \{1, \ldots, 2^n\} \), and

\( t \) is computed of \( M(\sigma) \) that starts at \( C_0 \)

and reaches \( C_1 \) in \( \leq 2^t \) steps.

Then \( \sigma \in L \iff \exists \text{ accepting configuration } C_i \).

If \( t = 0 \),

Also, \( R_m(\sigma, C_0, C_1, t) \Rightarrow \exists C_0 \leq \sum \forall \leq C_{t+1} \).

And \( R_m(\sigma, C_0, C_1, 0) \) is easy to check if true or false.

**Algorithm:**

Input \( \sigma \).

Loop over all \( C_i \in \{0, 1\}^{d \cdot S(n)} \).

If \( C_i \) enters an accepting configuration of \( M(\sigma) \)

Compute \( R_m(\sigma, \text{initial config.}, C_i, d \cdot S(n)) \)

If it accepts, halt and accept.

Exit loop.

Reject.

**Algorithm for \( R_m \):**

Return True/False.

If \( t = 0 \), accept/reject, based on \( M \)'s \( S(n) \) states.

Else:

Loop over all \( C_{t+1} \in \{0, 1\}^{d \cdot S(n)} \).

Compute \( R_m(\sigma, C_0, C_{t+1}, t+1) \).

If halt accept, then return True.

End loop.

Return False.

Space \( \leq 2^{O(S(n)^4)} \) — Pf. Recursion depth \( O(S(n)) \), local vars on \( O(S(n)) \) bit.
**Proof** Next time.

**Theorem** [Immudossy, Szepesvari; 1988]

Let \( S(n) \) be a space constructible.

Then \( \text{NSPACE}(S(n)) = \text{coNSPACE}(S(n)) \).

**Proof** Next time.

**Theorem** \( P = \text{NP} \Rightarrow \text{coNP} = \text{NP} \) and \( P = \text{coNP} \).

**Proof** \( P = \text{NP} \).

---

**Theorem** [Hartmanis, Stearns; 1964] [Cook; 1971]

If \( T, T' \) are time constructible, \( T(n), T'(n) \), and \( \lim_{n \to \infty} \frac{T(n)}{T'(n)} = 0 \), then \( \text{DTIME}(T(n)) = \text{DTIME}(T'(n)) \).

If \( \lim_{n \to \infty} \frac{T(n)}{T'(n)} \neq 0 \), then \( \text{NTIME}(T(n)) \neq \text{NTIME}(T'(n)) \).

**Theorem** [Stearns, Hartmanis, Lewis; 1965]

If \( S, S' \) are space constructible and \( \lim_{n \to \infty} \frac{S(n)}{S'(n)} \), then \( \text{SPACE}(S(n)) = \text{SPACE}(S'(n)) \).

**Proof** Next time.

\[ L \subseteq L \subseteq P \subseteq \text{NP} \subseteq \text{PSpace} = \text{NSpace} \subseteq \text{EXP} \}

\[ \text{coNP} \subseteq \text{coPSpace} \neq \text{coNP} \neq \text{coPSpace} \]
Algorithm $G$: $M_i$ will do essentially the same, except:

1. $M_i$ accepts $S_i$: $\text{NSPACE}(\Sigma^* \cdot \text{poly}(n))$.
2. $M_i$ rejects $S_i$: $\text{NSPACE}(\Sigma^* \cdot \text{poly}(n))$.

Note: $m=1$.

C: $M_i$ is $c$.

Return ans.

If any of above costs $\geq \text{poly}(n)$, halt and reject. (opposite of yes.)

End.
Algorithm to compute $n!$ for $n \leq k$.

Enter $k = 0$

For $C_i \leq 0 \ldots d, \ldots n$,

If "$C_i \leq R_i$ with $C_i \leq 1R_i$", keep

If "$C_i \leq R_i$ with $C_i \leq 1\text{sum}$", keep

else reject.

Algorithm to compute if $A_{C_1} \ldots A_{C_n}$ are all compatible, given $n \in \mathbb{Z}$.

keep, given config $g_i$, fill $(n-1)$

verify $C_{n-1} > C_n$, else reject

with $C_{n-1} > R_i$, also reject

if $g_i$ not acceptable, reject.

end loop

Accept

QED

Note: Lookup: Hopcroft-Paul-VanLiant: $DTIME(T(n)) \leq SPACE(T(n)/\log T(n))$.

"On Time versus Space", J. ACM 24(2) 332-337, 1977

prelim version in FOCS '75.
**Definition**: \( \text{coTIME}(T(n)) \subseteq \text{coSPACE}(T(n)) \)
\( \text{coNTIME}(T(n)) \subseteq \text{coNSPACE}(T(n)) \).

**Theorem**: \( \text{coTIME}(T(n)) = \text{TIME}(T(n)) \) and \( \text{coSPACE}(T(n)) = \text{SPACE}(T(n)) \).

**Proof.**

**Corollary to Theorem**: \( \text{coNSPACE}(S(n)) = \text{SPACE}(S(n)) \).

**Open Problem**: \( \text{NP} = \text{coNP} \)?

**Theorem.** Immersing Szilard's Theorem

**Corollary**: \( \text{NL} = \text{coNL} \).

**Proof.** Sufficient to show \( \text{coNSPACE}(S(n)) = \text{NSPACE}(S(n)) \).

Let \( M \) be a NTM that uses \( S(n) \)-space bounded.

Why: \( M \) runs for exactly \( S(n) \) steps for all inputs of length \( n \).

Let \( L = \{ w : M(w) \text{ accepts } \text{ on every computation path} \} \).

We want to recognize \( L \) in an NTM \( M' \), i.e., an NTM that runs in \( \text{SPACE}(S(n)) \), and such that \( M'(w) \) accepts on some computation path.

All \( M'(w) \) needs is one accepting path, so it works if some path rejects.

**General idea**: Let \( n_i \): # of configurations that can be reached by some computation path of \( M(w) \) in exactly \( i \) steps.

**Ri**: Create exactly \( n_i \) states.

**Ri**: \( \{ n_i \} \).

**Ri**: \( M'(0) \) will compute \( n_0, n_1, n_2, \ldots \) and has \( n_0 \) to help compute \( n_1 \).

Thus, using \( n_0 \) to help compute \( n_1 \), \( \ldots \), \( n_2 \), \ldots \), \( n_1 \) will decide if every state makes at least \( i \) steps.
Claim: $M$ can do any of the following tests, in the sense that

There is some nonnegativity path that succeeds,

ever fails.

(let $R_i = \{C : C$ is a configuration reachable by $M(\delta)\}$ exactly $i$ steps)

Task 1: Given a configuration $C \in R_i$

Alg: "guess" the computation

Task 2: Given a correct value for $n_i$, determine $C \in R_i$

Alg: Input $n_i$, $C_i$, $i$

Introduce: FoundC = false

Loop, successively guessing candidates $C_0$, $C_1$, ..., $C_{n_i}$

and checking that $C_0 < C_1 < \ldots < C_{n_i}$ and $C_j \in R_i$, if not reject

if every $C_j = C$, set FoundC = true

Endloop

Accept if FoundC = false

Memory usage: index $j$, $n_i$, $i$ - all $O(S(n))$ bits

$C_j$, $C_{j+1}$, $C$ - each $O(S(n))$ bits

Task 3: Given a correct value for $n_i$, determine $C \in R_{n_i}$

Alg: Same as above, but replace test "$C_j = C$" with

"$C_j$ is reachable in 1 step from $C_j$".

Task 4: Given $n_i$, compute $n_{i+1}$

Alg: For all configurations $C_0$, $C_{n_i}$ $\in R_i$

Test that all

nonterminally unity $C_i \in R_i$ or reject $C_i \in R_i$

Keep count of how many $C_i \in R_i$

Task 5: Given $n_i$, determine all $C \in R_{n_i}$ are accepting.

Alg as above, but test "$C_i$ rejects"
Alternating Turing Machines.

Two examples: An NTM has existential states, i.e. two transition functions $S_0, S_1$:

Accepts iff it accepts a series of choices of all halting moves that leads to acceptance.

A co-NTM machine is defined exactly the same, but the states are "universal". It accepts if any sequence of choices of moves, the machine enters an accepting state.

Formally an Alternating TM has:

finite list of states pre-ordered into

$$\text{Accepting} \quad \text{Existential (E)}$$

$$\text{Universal (U)}$$

one "initial" state.

finite # of tapes, alphabet symbols.

Two transition functions $S_0, S_1$.

Let ATM $M$ have input $x$. Define the set of configurations as before. Define $S_i(c)$: current read by $S_i$'s rule.

A configuration $C$ of $M(x)$ is accepting (and accepting if it is an $E$-state, both $S_0(c)$ and $S_1(c)$ are accept if $C$ is a $U$-state, both $S_0(c)$ and $S_1(c)$ are accept if $C$ is in an accepting state and accept $M(x)$ is accepting if its initial configuration accepting.
Defn. \( \text{ATIME}(T(n)) \), \( \text{ASPACE}(S(n)) \) [CKS]

Alternating Polynomial Time = \( \bigcup \text{ATIME}(n) \).

Thm. \( \text{NSPACE}(S(n)) \leq \text{ATIME}(S(n)^2) \)
\( S(n) \geq n \)

Proof. Use Savitch's Theorem's Proof. Actually \( \text{ATISP}(S(n)^2, S(n)) \)

Thm. \( \text{ATIME}(T(n)) \leq \text{SPACE}(T(n)) \).

Proof. Obvious? Try all possible choices in lex order.

Need to save current choice.

For each node along current computation path:

- if they second choice, whether the state configurtion reached by the first choice \( S_0 \) was accepting.

Comments: Wlog. the graph graph with nodes the set of configurations, and edges is defined by \( S_0, S_1 \) is acyclic. Recon. With a time space bound, one can just run "clock" and make any configuration enter a reject state if the clock runs over the allotted time.

Corollary. Alternating \( \text{PTIME} = \text{PSPACE} \).

Open: Can \( S(n)^2 \) be replaced by \( S(n) \) in theorem, as in Savitch's Theorem.

Thm. \( \text{ASPACE}(S(n)) \leq \text{DTIME}(2^{O(S(n))}) \)

Proof. Essentially same as proof that \( \text{NSPACE}(S(n)) \leq \text{DTIME}(2^{O(S(n))}) \)

Thm. \( \text{DTIME}(2^{O(S(n))}) \leq \text{ASPACE}(S(n)) \).

Proof. An fort can be justified by checking model anticipates - an shrik to le -
A $\Sigma_i$-alternating TM is a alternating TM which switches for exactly $i$ alternated
a total of $i$ times.

$\Pi_i$-alternating TM defined dually.

$\Sigma_i: TIME(T_i(n)) = \{ L \subseteq \{0,1\}^* : L \text{ accepted by a } \Sigma_i \text{-TM
with } n \text{ steps } \leq T_i(n) \}

\Pi_i: TIME(T_i(n)) \text{ defined dually.}

$\Sigma_i^p = \bigcup_{c_0} \Sigma_i: TIME(n^c)$

Collectively called the
$\Pi_i^p = \bigcup_{c_0} \Pi_i: TIME(n^c)$

Polynomial time hierarchy

$NP = \Sigma_i^p \text{ coNP = } \Pi_i^p \text{ \Sigma_i = co } \Pi_i^p

NP = \Sigma_i^p \text{ coNP = } \Pi_i^p

\Pi_i^p = coNP \text{ $NP = \Sigma_i^p$}

$NP = \Sigma_i^p \text{ coNP = } \Pi_i^p

\text{Open: Is } P = NP \text{ coNP.}

Alternate Def. of $\Sigma_i^p / \Pi_i^p

Then $L \subseteq \Sigma_i^p \text{ if } \exists p_1, p_2 \text{ and a polynomial time } R(\cdot)$

$s.t. \forall \sigma \in L \iff \forall u_1, u_2, \ldots, u_i \in \{0,1\}^i \text{ or } \{0,1\}^i \ldots \text{ or } \{0,1\}^i \ldots \text{ or } \{0,1\}^i, R(u_1, u_2, \ldots, u_i)" 

$P$-duality for $\Pi_i^p$.

Pf. Like before, for $NP = \Sigma_i^p: TIME(n^c)$. 

Recall a language $L$ is a subset of $\{0,1\}^*$.  

**Def**: A oracle $\mathcal{O}$ is a subset of $\{0,1\}^*$.  

An oracle Turing Machine, written $M^{\mathcal{O}}$, is a TM augmented with a special "query" state and a designated "oracle query type" $\mathcal{O}$ and the designated oracle answer states $\mathcal{O}_0$, $\mathcal{O}_1$.  Whenever $M$ enters state $\mathcal{O}_0$ the oracle tape is sealed and is over some string $\omega \in \{0,1\}^*$ (delimited by non-0,1's).  

The next step of $M$ places $M$ in either  

$$\text{if } \omega \in \mathcal{O}_0$$  

$$\omega \notin \mathcal{O}_0$$  

and $M$'s tape contents and tape head positions unchanged.  

**Meth0de**: All our earlier def's were extend to oracle Turing machines, so do all prior theorems so far!

The notations extend to complexity classes as:  

For example,  

$$P^{\text{NP}} = \{L : \text{for some OTM } M \in \text{DTIME}(n^c), \text{ and some } L \in \text{NP}, L = M^{\text{L'}} = \{\sigma \in \{0,1\}^* : M^{\text{L'}}(\sigma) \text{ accepts}\}\}.$$ 

Thus $NP \subseteq P^{\text{NP}}$, so $NP \subseteq P^{\text{NP}}$  

Thus $P^{\text{NP}} = P^{\text{NP}}$  

Thus $P^{\text{NP}} = \Sigma_2^p \cap \Pi_2^p$
If it suffices to prove $P^{NP} \subseteq \Sigma_2^p$ (since $P^{NP}$ is closed under complementation and $\Sigma_2^p \subseteq \Sigma_2^{12}$).

Let $L \subseteq P^{NP}$, say accepted by P-time $M^2$ with $L \in NP$. Let $\Sigma = \{\sigma \in \{0,1\}^* : \exists u, |u| \leq 10^6 \cdot c, N(<\sigma,u>) \}$ for some $N \in \text{Ptime}$.

Consider $\sigma$ input to $M^2$ (to decide if $\sigma \in L$).

The computation of $M^2(\sigma)$ is a series of configurations:

$$C_0, C_1, \ldots, C_i, C_{n_0}$$

where $p$ is an polynomial in $n = 10^6$.

At certain steps, the machine makes an oracle call, i.e., a string $\sigma_i$ on the oracle query tape and $C_{i+1}$ is in state $S_2$, $S_3$ depends on whether $\sigma \in L$,

i.e., whether there is a $u_i$ such that $|u_i| < 10^6 \cdot c$ and $N(<\sigma,u>)$.

---

$\Sigma_2$-algorithm for $L$:

Input $\sigma$. Goal: decide if $\sigma \in L$.

Existentially guess all configurations $C_0, \ldots, C_{p(n)}$.

And for each oracle query $u_i$ in config $C_i$ that has $C_i$, in state $S_2$, guess a value $u_i$, $|u_i| \leq p(n)$.

Determine whether $u_i$ satisfies $N(<\sigma,u>)$.

Reject if any one fails.

Universally pick $u_i$'s for the remaining oracle queries, i.e., for each $C_i, C_{i+1}$ query with $C_i$, in $S_2$,

verify that $N(<\sigma,u_i>)$ is false.

If any fails to be true, reject.

Determine whether $C_0, \ldots, C_{p(n)}$ is a correct computation based on the oracle and query queries.

Remark: Total runtime is $O(p(n))$ for some program $p$. 

Then if $P = \text{NP}$, then $\forall i, E_i^P = P \neq P = \text{PH}$.

Then if $\text{NP} = \text{coNP}$, the $\text{PH} \neq \text{NP}$.

Proof: Any predicate $L \in \text{PH}$ is in some $\Sigma_i^P$ and can be written as

$$L \iff \exists E_{i_1}, E_{i_2}, \ldots, E_{i_n} \in \text{coNP}, (\forall E_k, \forall u_{k+1} \in \text{coNP}(b)) N(\sigma, u, u_{k+1})$$

If $\text{NP} = \text{coNP}$, the $\text{NP}$ predicate

$$R(\sigma, u_1, u_{k+1}) := (\exists E_{i_k}, \forall u_{k+1} \in \text{coNP}(b)) N(\sigma, u, u_{k+1})$$

can be rewritten as a $\text{coNP}$ predicate

$$\forall E_{i_k}, \forall u_{k+1} \in \text{coNP}(b) \exists \forall v_k, N(\sigma, u, u_{k+1}, u_{k+1})$$

Then

$$L \iff \exists E_1 \ldots (\forall <u_{k+1}, v_k>, k_{k+1}, v_k \geq P \text{Red}\text{ed}(101) N(\sigma, u_{k+1}, v_k))$$

Then can use induction on $i$. 
Complete Problems:

1. Canonical example:

\[ SAT \text{ is } NP \text{-complete.} \]

**Defn:** SAT is the following decision problem:

**Input:** \((\{s_0, 1\}^*, \text{encoding a set of clauses})\)

- A literal is \(x_i \) or \(\overline{x_i} \).
- A clause is a set of literals.
- A truth assignment \(t: \{x_i\} \rightarrow \{T, F\}\) \(t(x_i) = \overline{t(\overline{x_i})}\).
- \(T \land C \iff \text{true iff } C\) if \(t(x) = T\) for all \(x \in C\).
- \(T \lor T', T ~ a ~ \text{set of clauses}, \iff T \lor \neg T', \text{VCE}\).

**Inc.:** \(F \leq C N F \text{ formula, } (\lor \text{ of } V \text{'s of literals})\)

**Then:** For all \(L \in NP, \exists \text{ a polynomial time f,}\)

\[ s.t.: \forall W (\text{W is satisfiable } fsat). \]

**Pf:** See Garey-Johnson

**Select ideas:** Use variables \(x_i, p, t, s\)

To assign \(\text{ "Top } #1, \text{ point } p, \text{ contains symbol } r \text{ at } t + t\):

\[ Z_{p \land s} \iff \text{M is state } p \text{ at } t + t \]

\[ Y_{p, t, r} \iff \text{Top } i \text{ is head is at part } p \text{ and } r \]

\[ X_{i, p, t, s} \iff \text{Each } X_{i, p, t, s} \text{ depends on any } \]

\[ \{\text{finite many } Y_{i, p, t, s}, Z_{p, t, s}, Y_{i, p, t, s}'s\} \]

\[ \exists_{X_{i, p, t, s}, Y_{i, p, t, s}, Z_{p, t, s}} \text{ have definite values under each}\]

\[ \text{assign } X_{i, p, t, s}, Y_{i, p, t, s}'s. \]

\[ X_{i, p, t, s}, Y_{i, p, t, s}, Z_{p, t, s} \text{ are definite values depend}\]

\[ \text{on only finitely many but } f \text{ of } \text{input } x \text{ to } M. \]

\[ W \equiv M \text{ accepts at } t, \quad W \equiv M \]

**Accept**
Then \( u \in L \iff \overline{f(u)} \in \text{SAT} \).

A.E.D. (Handwriting)

Comments: It can actually be log-space computable.

(Explain what this means).

This means "many-one complete" or "Karp-complete".

Comment: Also 3-SAT.

Def. CVP, Circuit Value Problem. Inputs 0/1 (Not Variable).

Then CVP \( \in \text{P} \).

Arora-Barak call it "Circuit Eval".

Then \( \forall L \in \text{P} \exists f, \text{computable in log-space such that} \)

\[ \forall u (u \in L \iff f(u) \in \text{CVP}). \]

Pf: Like above, but \( x, y, p, t \) are given by fixed-size circuits of definite may of the values at time \( t \). Put these together to compute

Accept, Too.

With care, \( f \) is log-space computable by some reduction.

Def. \( L \) is \( C \)-complete (many-one Complete) if \( \forall L \in \text{P} \) and \( \forall L \in \text{C} \exists f, f \text{logs} \forall u (u \in L \iff f(u) \in L) \).
**Defn** \( L \subseteq \{0,1\}^* \) has circuits of size \( S(n) \), provided:

- for all \( n \geq 1 \), \( \exists \) Boolean circuit \( C_n \):
  1. with variable \( x_1, \ldots, x_n \);
  2. with gates \( \land, \lor, \neg \);
  3. a single output signal
  4. For all \( x_1, \ldots, x_n \), \( x_1, \ldots, x_n \in L \iff C_n(x_1, \ldots, x_n) = 1 \)
  5. For all \( n \), size \((C_n) \leq S(n)\).

**Thm.** \( P \) has polynomial size circuits.

**Pf.** Immediate from the above construction.

**Defn** \( P/poly \) = class of languages \( L \) with polynomial size circuits.
A complete problem for $\text{NL}$

Path: Directed graph $G = (V, E)$ and grammar $S, \tau$.

Then Path is $\text{NL-complete}$. 

NL algorithm: Start at $S$, nondeterministically choose a move $s_i$, set $S^t = S$.

Repeat, until $S = \tau$ the halt and accept.

Thus Path is $\text{NL-complete}$. 

If $L \leq \text{NL}$, on input $w$,

let $G = \text{configuration graph of } M(w), \quad \delta^2 G = 2$

$s = \text{initial configuration}$
$t = \text{final accepting configuration}$

Note $(G, s, t) \in \text{PATH}$ iff $M(w)$ accept.

Define $f: 0 \rightarrow (G, s, t)$.

Q.E.D.

A complete problem for $\text{PSPACE}$

Define $\text{QBF} = \{ \text{quantified Boolean formulas} \}$

$= \{ \text{formulas} \}: Q_1 x_1, Q_2 x_2, \ldots, Q_k x_k, \varphi(x_1, x_2, \ldots, x_k) \}

\text{with quantifiers } \forall, \exists, x_i, \forall, \exists, \neg, \vee, \wedge, \oplus, \otimes, \text{ true, false}.

Then $\text{QBF} \in \text{PSPACE}$.

Then: If $L \leq \text{(N)PSPACE}$, then $\exists \text{ logspace } f, \forall f \in \text{(logL) \cup P} (\text{QBF}), \exists \text{ Minic proof of Savitch's Theorem. Note quadratic blow up seen.}$
A probabilistic TM has two transition functions $S_0, S_1$. At each step, it chooses to use $S_0$ or $S_1$, with probability $\frac{1}{2}$ each.

The machine $(M, W)$ always halts and accepts $w$ if $|W| \leq \text{poly}(w)$, runs in time $O(n^k)$, space $S(n)$, etc., defined as before.

**Defn:** \( \text{PP} = \{ L : \#0, \#1 \in L \iff \Pr[M(0) = 1] \geq \frac{2}{3} \text{ for some } M \text{ that runs in polynomial time} \} \)

**Equivalent defn:**

\( L \in \text{PP} \iff \exists \text{ poly-time } TM \text{ s.t. } M \text{ and } q \text{ poly-time } p(n) \text{ s.t. } \forall \sigma \in L \iff |\{u \in \{0,1\}^{p(n)} : M(u, \sigma) \text{ accepts}\}| \geq 2 \}

**Pf:** Easy.

**Defn:** \( \text{BPTIME}(T(n)) = \text{ the set of languages } L \text{ such that } \exists \text{ prob. TM } M \text{ that runs in time } O(T(n)) \text{ s.t.} \)

\[ \forall \sigma : \sigma \in L \implies \Pr[M(0) = 1] \geq \frac{2}{3} . \]

\[ \sigma \notin L \implies \Pr[M(0) = 0] \geq \frac{2}{3} . \]

**Defn:** \( \text{BPP} = \bigcup_n \text{BPTIME}(n^c) \)

**Defn:** \( \text{RTIME}(T(n)) = \{ \text{languages } L \text{ s.t. } \exists \text{ prob. TM } M \text{ that runs in time } O(T(n)) \text{ s.t.} \)

\[ \forall \sigma : \sigma \in L \implies \Pr[M(0) = 1] \geq \frac{2}{3} . \]

\[ \sigma \notin L \implies \Pr[M(0) = 0] = 0 \text{ (so } \Pr[M(0) = 0] = 1) . \]

\( \text{RP} = \bigcup_n \text{RTIME}(n^c). \)
Theorem: \( \text{ZPP} \subseteq \text{RP} \cap \text{coRP} \).

Proof: Run \( \text{RP} \cap \text{coRP} \) repeatedly until a result is obtained. 

Expected runtime \( = \frac{2}{3} T(n) \left( 1 + \frac{1}{3} + \frac{1}{3^2} + \ldots \right) \)

\( \geq 3 T(n) \)
Examples of Probabilistic Algorithms (see Aho-Cormen pp. 126-128)

1. Find the median element.

Algorithm: Input array A = a_0, a_1, ..., a_{n-1}, and k ∈ [0, n-1].
Find k-th (A, k):

- Select i ∈ [0, n-1] at random.
- Scan A, count # of q's with q ≤ a_i.
  - If i < k,
    - Scan again, creating new list of n elements < a_i.
    - Return Find-k-th (B, k).
  - Else
    - Scan again, create new list j of n elements > a_i.
    - Return Find-k-th (B, k-N).

This is the most efficient algorithm known.

Claim: Expected running time is O(n).

Proof: Intuition is that |B| ≤ \frac{3}{4}|A| with probability \frac{1}{2} (say).

If this happens every other time, we can bound after

\[ \text{Recurrence is } O(1A) + \text{time to vacuum calls} \]

\[ \text{i.e., } O(1A + \frac{3}{4}1A + (\frac{3}{4})^21A + ...) = O(A). \]

More generally, if you do $\sum_{i=0}^{\infty} \frac{3}{4}^i 1A$ iterations, you have $\sum_{i=0}^{\infty} \frac{3}{4}^i 1A = 1$ with high probability (close to 1).

So running time is

\[ \leq \left[ \sum_{i=0}^{\infty} \frac{3}{4}^i 1A \right] (1 - \epsilon) + \epsilon \left[ \sum_{i=0}^{\infty} \frac{3}{4}^i 1A + (1 - \epsilon) + \epsilon \left[ \ldots \right] \right] \]

\[ = O(1A). \]
Problem

Input: \( N \)

Output: Accept if \( N \) is a composite \((\text{non-prime})\), with high probability.

\[
QR_N(A) = \begin{cases} 
0 & \text{if } \gcd(A,N) \neq 1 \\
+1 & \text{if } A = B(x,y) \text{ for } x,y \in \mathbb{Z} \\
-1 & \text{otherwise}
\end{cases}
\]

Fact: \( QR_N(A) = A^{(n+1)/2} \) for odd prime \( N \) and hence is computable in polynomial time.

Fact: The Jacobi symbol \( \left( \frac{N}{A} \right) = \prod_{p|N} \left( \frac{N}{p} \right) \) where \( p \) runs over the prime factors (with repetitions) of \( N \).

Fact: \( \left( \frac{N}{A} \right) \) is computable in polynomial time (by simple recursion).

Fact: If \( N \) is prime, \( \left( \frac{N}{A} \right) = QR_N(A) \) for all \( A \in \{2, N-1\} \)

If \( N \) is not prime, at most half of \( A \)'s satisfy this.

Algorithm: \( A \) inputs \( N \).

Pick \( A \in \{2, N-1\} \) at random.

If \( \left( \frac{N}{A} \right) = QR_N(A) \), reject.

If \( \left( \frac{N}{A} \right) \neq QR_N(A) \), accept.

Repeat several times, probability success is \( \Theta \left( \frac{1}{\log N} \right) \).

So probability of error shrinks to \( \left( \frac{1}{2} \right)^{\log N} \), i.e. \( 2^{-n^2} \) for any desired fixed \( c \).

Agarwal-Kayal-Saxena

Other algorithms for primality testing.
Polynomial Identity Testing

Input \( p(x) \in \mathbb{F}[x] \) - univariate coefficient.

\[ \Rightarrow \text{specified by an algebraic circuit or straight-line program.} \]

with inputs \( x_1, \ldots, x_n \), output a single output.

Goal: Decide if \( p(x) = 0 \) for all \( x \in \mathbb{F} \).

Schwartz-Zippel Lemma: Let \( p(x) \) be a non-zero polynomial in \( n \) variables of total degree \( d \).

Let \( S \subseteq \mathbb{F} \) be finite. Then

\[ \Pr_{\mathbf{a}, \mathbf{c} \in S} \left[ p(\mathbf{a}) = 0 \right] \leq \frac{d}{|S|^n}. \]

Rk: Independent of \( n \).

Pf: By induction on \( n \).

Base case: \( n = 1 \). A degree \( d \) polynomial has \( \leq d \) roots (in \( \mathbb{F} \) or \( \mathbb{C} \)).

Induction Step: Let \( d \) degree be the number of variables be \( n+1 \).

Write \( p(x) = \sum_{k=0}^{d} x_1^k \cdot p_k(x_1, \ldots, x_n) \), each \( p_k \) of degree \( \leq d-k \).

Choose \( \text{max } k \) such \( \forall x_1, \ldots, x_n \) is non-zero.

By inductive hyp, \( \Pr_{\mathbf{q} \in S} \left[ p_k(\mathbf{q}) = 0 \right] \leq \frac{d-k}{|S|^{n-1}} \).

For each \( \mathbf{a} = \mathbf{q} \in S \), with \( p_\mathbf{a}(\mathbf{x}) = 0 \), we get a \( d-k \) polynomial of degree \( k \) in \( x_1 \).

If \( q_{n+1} \in S \) is chosen at random, \( \Pr_{\mathbf{q}_{n+1} \in S} \left[ P_{\mathbf{q}_{n+1}}(x_{n+1}) = 0 \right] \leq \frac{k}{|S|} \).

Thus,

\[ \Pr_{\mathbf{q}, \mathbf{a} \in S} \left[ p(\mathbf{q}, \mathbf{a}) = 0 \right] \leq \frac{d-k}{|S|^{n-1}} + \left( 1 - \frac{d-k}{|S|^{n-1}} \right) \frac{k}{|S|} \leq \frac{d}{|S|^n}. \]

Schwartz-Zippel also holds over \( GF(q) \), with \( S \subseteq GF(q) \), \( d < q \).

\[ \Rightarrow \text{Some proof works.} \]
Algorithm: If \( p \) computed by a circuit of size \( M \), then it has degree

\[ \text{degree} \leq 2^m \quad (\text{at most doubled at each gate}) \]

Pick \( S = \{1, \ldots, 2^m\} \). Idea: Choose \( a_1, a_2 \in S \) at random

For values \( a_1, a_2 \in S \), \( p(a_1, a_2) \) can be bounded by \( (2^m)^2 \). These values cannot uniformly cover \( 2^m \) bits just to recognize one.

To fix this, we choose \( a_1, \ldots, a_n \in 2^m \) at random, and

\[ a \in \text{random } \subseteq [0, 2^m] \]

Evaluate \( p(a) \) and \( q \): if non-zero, reject. If zero accept.

\( q \) is prime and \( q \parallel y = p(a) \), then \( p(a) \) and \( q \neq 0 \).

What is the probability \( q \) is prime? \( \Omega\left(\frac{1}{n}\right) \) by the prime number theorem. \( \# \text{ of primes } \leq 2^m \Rightarrow \Omega\left(\frac{1}{n}\right) = \Omega\left(\frac{1}{2^m}\right) \).

What is the probability \( 1/y \)? \( \frac{1}{n} \leq 2^{-m \log_2 n} \) since \( n \leq 2^m \).

So \( \Omega\left(\frac{1}{2^m}\right) \rightarrow \Omega\left(\frac{1}{n}\right) \).

Prove: \( (8 \text{ is prime } \iff 8 \parallel y) = \Omega\left(\frac{1}{2^m}\right) \) \( p(x) \) is not identically zero, then for \( \bar{a} \in [0, 2^m] \), \( q < 2^m \),

\[
\Pr[p(\bar{a}) \text{ and } q \neq 0] = \frac{n}{2^m} \cdot \Omega\left(\frac{1}{2^m}\right) = \Omega\left(\frac{1}{2^m}\right)
\]

Thus the \( \{p(x): p(x) \text{ is identically zero}, f \text{ given by a circuit/straight-line program}\} \) is in \( \text{coR} \).

I.e., \( \text{ZEK} \in \text{coR} \).

Imply \( \Pr[p = \text{accy}^{\text{R}}] = 1 \)

If \( \Pr[p \neq \text{accy}^{\text{R}}] < 2^{-m \log_2 n} \),

Thus the \( \{p(x): p(x) \text{ is identically zero}, f \text{ given by a circuit/straight-line program}\} \) is in \( \text{coR} \).

I.e., \( \text{ZEK} \in \text{coR} \).

Imply \( \Pr[p = \text{accy}^{\text{R}}] = 1 \)

If \( \Pr[p \neq \text{accy}^{\text{R}}] < 2^{-m \log_2 n} \),
Probability amplification

Obviously the \( \frac{n}{4} \) to (1) above are not optimal, e.g. new the same start in time (5g), reduces the probability of an error for \( \frac{c}{3} \) to \( \left( \frac{1}{3} \right)^n \), i.e. exponentially small probability of error.

Basic Probability Results

(1) Markov's Inequality:

a) For \( X \) a random variable, they have non-negative values only,

\[
Pr \left[ X \geq k\mu \right] \leq \frac{1}{k}
\]

when \( \mu \equiv \text{mean}(X) \), so \( Pr[X \geq x] \leq \frac{x}{\mu} \).

(2) Chebyshev Inequality: Let \( X \) have mean \( \mu \), Std dev. \( \sigma \)

\[
Pr \left[ |X - E(X)| \geq k\sigma^2 \right] \leq \frac{1}{k^2}.
\]

So \( Pr[X - E(X) \geq k\sigma^2] \leq \left( \frac{1}{k} \right)^2 \)

Proof: Recall

\[
\text{Variance} \quad Var(X) = E((X - \mu)^2)
\]

where

\[
\text{Std Dev.} = \sqrt{Var(X)}.
\]

So \( \sigma^2 = Var(X) = E((X - \mu)^2) \).

By Markov's Inequality,

\[
Pr \left[ |X - E(X)| \geq k\sigma^2 \right] \leq \frac{1}{k}
\]

So

\[
Pr \left[ |X - E(X)| \geq \frac{1}{k} \right] \leq \frac{1}{k}
\]

Now do a change of variable, replacing \( \sqrt{\mu} \) with \( k \).

(3) Chernoff Bounds: If \( X_1, X_2, \ldots, X_n \) are mutually independent random variables, they values in \( \{0,1\} \), then, for \( S \geq 0 \),

\[
Pr \left[ \sum_{i=1}^{n} X_i \geq (1+S)\mu \right] \leq \left( \frac{e^S}{(1+S)^{\mu}} \right)^n \leq \left( e^{-S/4} \right)^n \quad \text{for} \quad S \geq 0.
\]

\[
Pr \left[ \sum_{i=1}^{n} X_i \leq (1-S)\mu \right] \leq \left( \frac{e^S}{(1-S)^{\mu}} \right)^n \leq \left( e^{-S/4} \right)^n
\]

where \( \mu = \sum_{i=1}^{n} \mu_i \), \( \mu_i \equiv E(X) \).
Let \( \text{RP} = \mathcal{C}_L \): for some polynomial \( T(n) \),

\[
\mathcal{C}_L \Rightarrow \Pr[M(\mathcal{L}) \text{ accepts}] > \frac{2}{3}
\]

\[
\mathcal{C}_L \Rightarrow \Pr[M(\mathcal{L}) \text{ accepts}] = 0
\]

Then \( \text{RP}[\frac{1}{2^{2n^c}}] \subseteq \text{RP} \) for all \( c > 0 \).

**Proof**

2: Let \( \mathcal{C} \in \text{RP} + \mathcal{M}_0 \) accept with \( \Pr > \frac{2}{3} \) if \( \mathcal{C} \in \mathcal{C}_L \).

Run \( M(\mathcal{C}) \) \( k \) times, accept if any one instance accepts.

Probability, for \( \mathcal{C} \in \mathcal{C}_L \), that \( \mathcal{C} \) does not accept \( \leq \frac{1}{2^k} \).

Take \( k = \log_2 \left( \frac{1}{1/2} \right) \leq \log_2 \left( \frac{1}{1/3} \right) \leq n^c \).

Then \( \forall c \in [0, 1] \), \( \mathcal{R}_R[c] = \text{RP} \). **Proof** by the same proof.

Then: \( \mathcal{R}_R[\frac{1}{n^c}] = \text{RP} \) for all \( c > 0 \).

It is obvious.

\[ \Rightarrow \]

Take \( \mathcal{C} \in \text{RP}[\frac{1}{n^c}] \) accepted by \( \mathcal{R}_R \) machine \( M \) with probability \( \frac{1}{n^c} \).

Iterate \( M(\mathcal{C}) \) \( n^c \) times.

Prob of (eventually) never acceptance is:

\[
\leq (1 - \frac{1}{n^c})^{n^c} \leq \frac{1}{e}
\]

\[ \leq \frac{(1 - \frac{1}{a})^a < e^{-1}}{a \ln(1 - \frac{1}{a}) < -1}
\]

\[ a \ln(1 - \frac{1}{a}) = a(-\frac{1}{a}) = -1 \]
Defining \( \mathsf{BPP}[p] \) similarly, now \( p > \frac{1}{2} \) is required.

Then 1. \( \mathsf{BPP}[\frac{1}{2} + \frac{1}{n^c}] = \mathsf{BPP} \).

Proof 2. Obvious

2. Use Chebyshev bounds. (2)

Proof of Thm. 1: Given \( L \in \mathsf{BPP}[\frac{1}{2} + \frac{1}{n^c}] \) with M.S.T. \( \sigma(L) \Rightarrow \Pr[M(u, c) = 1] \geq \frac{1}{2} + \frac{1}{n^c} \),

\[ \sigma(L) \Rightarrow \Pr[M(u, c) = 0] \leq \frac{1}{2} + \frac{1}{n^c}. \]

Algorithm: Run \( M \) \( k \) times, take majority answer.

We want to determine \( k \).

For \( w \in L \), consider random variable \( X_i = \begin{cases} 1 & \text{if } M(u_i, c) = 1 \\ 0 & \text{otherwise} \end{cases} \)

So \( \Pr[X_i = 1] = p \geq \frac{1}{2} + \frac{1}{n^c} \), \( \Pr[X_i = 0] = 1 - p = \frac{1}{2} - \frac{1}{n^c} \).

\[ \mu_X = \mathbb{E}[X] = p \quad \text{Var}(X) = \mathbb{V}[X - \mu] = np \]

Let \( X = \sum_{i=1}^{k} x_i \). \( \mathbb{E}[X] = kp \quad \text{Var}(X) = kp^2 \quad \text{(by pairwise independence)} \)

\[ \sigma_X = \sqrt{\text{Var}(X)} = \sqrt{k} \] \[ \leq \sqrt{k} \frac{p^2}{p^2 - (1 - p)^2} \]

By Chebyshev's inequality, \( \Pr[X \geq \frac{k}{2}] \Rightarrow \Pr[X - kp \geq \frac{k}{2} - kp = \frac{k}{2} - p] \Rightarrow \Pr[X - kp \geq \frac{k}{2}] \)

\[ \leq \Pr[|X - \mu| \geq \frac{k}{2}] \leq \frac{\sigma_X^2}{k} \leq \frac{\frac{p^2}{k(1 - p)^2}}{k} \]

Taking \( k = 4n^c \), this becomes \( \frac{1}{8} < \frac{1}{3} \).
Proof of Thm 2:

Let \( L, M, K, \rho, \varphi \) be close, and \( \rho > 2/3 \) as desired.

By Chebyshev bounds, (since \( X_i, \xi \) are mutually independent)

\[
Pr[X < \frac{1}{2}] \leq Pr[X \leq (1 - \frac{1}{4})^{\frac{3}{2}K}] = Pr[X \leq (1 - \frac{1}{8}) E(X)]
\]

when \( S = \frac{1}{4} \).

\[
\leq \left( e^{-\frac{3}{8}K} \right)^{E(X)} = \left( e^{-\frac{3}{8}K} \right)^{K} = \left( e^{\frac{\varphi}{K}} \right)^{K} = \left( e^{\varphi} \right)^{K} = \left( e^{\varphi} \right)^{K},
\]

chose \( k \) so that \( d^k < \frac{1}{2^{n^c}} \), i.e., \( k = \frac{\log(d)}{c} \cdot n^c \)

\[
d^k < \frac{1}{2^{n^c}}
\]

\[
(k \log d) < -n^c
\]

\[
k > \left( \frac{-1}{k \log d} \right) n^c
\]

(Note \( \log d < 0 \))

\[
k > (log d, n^c)
\]
Corollary. \( \text{BPP} \subseteq \text{P/poly}. \)

\[ \text{Pf:} \] Let \( L \in \text{BPP}[1 - \frac{1}{2^k}] \), accepted by machine \( M \).

For any \( u \in \{0,1\}^n \), call \( v \in \{0,1\} \) "good" if \( u \in M(v) \).

For any \( u \), \( \Pr[v \text{ is good}] \geq 1 - \frac{1}{2^k} \). \( \Pr[v \text{ is not good}] \leq \frac{1}{2^k} \).

\[ \text{Lemma.} \] \( \forall u \in \{0,1\}^n \), \( \Pr\left[ E(u,v) = 1 \right] \geq \frac{1}{2^{k+1}} \).

\[ \text{Pf.} \] Given \( \forall u \in \{0,1\}^n \), \( \Pr\left[ E(u,v) = 1 \right] \geq \frac{1}{2^{k+1}} \).

Thus: \( \sum_{v \in \{0,1\}^n} \Pr\left[ E(u,v) = 1 \right] \geq \frac{1}{2^{k+1}} \).

\[ \text{i.e.} \; \sum_{v \in \{0,1\}^n} \sum_{u \in \{0,1\}^n} \Pr\left[ E(u,v) = 1 \right] \geq \frac{1}{2^{k+1}} \).

So, for any \( v \), \( \sum_{u \in \{0,1\}^n} \Pr\left[ E(u,v) = 1 \right] \geq \frac{1}{2^{k+1}} \). \( \therefore \) \( \Pr\left[ E(v,u) \right] \geq 1 - \frac{1}{2^{k+1}} \).

\[ \text{Then,} \; \exists v \in \{0,1\}^n \; \Pr\left[ v \text{ is good for all } u \right] = 1 - \frac{1}{2^{k+1}}. \]

Since, then on only \( 2^{-k} \) \( v \), this means, \( v \) is good for all \( v \).

\[ \text{P/poly algorithm} \]

Let \( f(n) = \sum_v \text{good for all } u \in \{0,1\}^n \).

Then \( \forall u \in \{0,1\}^n \Rightarrow M(h, f(u)) = 1. \)

\[ \text{get.} \]

Note in fact (Markov's inequality) that the fraction of \( v \) which are good for all \( u \) is \( \geq 1 - \frac{2^k}{2^{2k}} = 1 - 2^{-2k} \).

\( \therefore \) a random \( v \in \{0,1\}^n \) is good for all \( u \in \{0,1\}^n \) with high probability.
Def: \( \#P \) is the set of functions of the form
\[
P(x) = \#u \in \{0,1\}^x \text{ s.t. } M(u,x) \text{ accepts.}
\]

Thm: \( \#P \leq \#P \).

**Def:** A \( \#P \) machine can query a single \( \#P \) function.

A \( \#P \) machine can query the a single (pronouncedly) \( \#P \) function i.e., a \#P function \( L \),

\[
L = \{ \sigma : \Pr[M(\sigma) \text{ accepts}] \geq \frac{1}{2} \}
\]

\[
\subseteq \{ \sigma : \#u \in \{0,1\}^x \text{ s.t. } M(\sigma, u) \text{ accepts} \geq \frac{1}{2} \cdot 2^{P(101)} \}
\]

**Proof:** \( \#P \leq \#P \) is pretty obvious.

1. Suppose \( L \subseteq \#P \)

   Let \( L' \) be \( \#P \)

   that makes cells to

   \[
f(\sigma) = \#u \in \{0,1\}^x \text{ s.t. } (M(\sigma, u) = 1)
\]

   Define a \( \#P \) problem \( \mathcal{L}' \):

   \[
   N(\sigma, w) = \begin{cases} 
   1 & \text{if } u \in \mathcal{L} \& M(\sigma, u) = 1 \\
   1 & \text{if } u \in \mathcal{L} \& u \in \mathcal{L}' \\
   0 & \text{else}
   \end{cases}
\]

   For \( w = \text{binary encoding of } j \),

   \[
   \langle \sigma, w \rangle \in L' \Leftrightarrow j + \left( \#u \in \{0,1\}^x \text{ s.t. } M(u, \sigma) = 1 \right) \geq 2^{P(101)}
\]

   i.e.,

   \[
   \left( \sum \right) \geq 2^{P(101)} - j.
\]

   This can be run search over \( c \) cells to determine the exact \# of \( u \in \{0,1\}^x \text{ s.t. } M(u, \sigma) = 1 \).
Then Sipser–Gacs (Sipser '84).

\[ \text{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p \]

**Proof:** Let \( L \subseteq \text{BPP}[\frac{1}{2^n}] \). Suffices to show \( L \subseteq \Sigma_2^p \).

By def. of \( \text{BPP}[\frac{1}{2^n}] \), \( \exists \text{PTM} M, \forall u, v \in \{0,1\}^n, M \text{ runs in poly time.} \)

\[ \forall u, v \in L \implies \Pr_{v \in \{0,1\}^n} M(u, v) = 1 > 1 - 2^{-n}, \quad n = 1, 2, \ldots \]

\[ \forall u, v \not\in L \implies \Pr_{v \in \{0,1\}^n} M(u, v) = 1 < 2^{-n}. \]

Let \( m = n^c, \quad c > 1 \)

Let \( S_u = \{ v \in \{0,1\}^m : M(u, v) = 1 \} \)

\[ \exists |S_u| > 2^{m-2^{-m-n}} \quad \text{(big)} \]

or \( |S_u| < 2^{-m-n} \quad \text{(small)} \)

**Claim 1:** If \( S_u \) small, \( |S_u| < 2^{m-n} \), and \( u_i, u_k \in \{0,1\}^m, k = \frac{m}{n} \),

then \( \bigcup_{i=1}^k (S + u_i) \neq \{0,1\}^m \).

Here \( S + u_i = \{ w + u_i : w \in S \} \) when "+" means addition Mod 2 \( (\text{XOR}) \).

\[ |S + u_i| = |S|, \quad \text{so} \quad |\bigcup_{i=1}^k (S + u_i)| \leq k |S| = (\frac{m}{n} + 1) 2^{m-n} < 2^m \]

**Claim 2:** If \( S \) is large, then \( \exists u_i, u_k \) s.t. \( \bigcup_{i=1}^k (S + u_i) = \{0,1\}^m \).

**Proof:** We'll show this works for a randomly chosen \( u_i, u_k \in \{0,1\}^m \).

Fix \( w \in \{0,1\}^m \). \[ \Pr_{u_i \in \{0,1\}^m} \left[ (S + u_i) \text{ contains } w \right] \]

\[ \leq \Pr_{u_i \in \{0,1\}^m} \left[ (S + u_i) \text{ intersects } S \right] \]

\[ = \Pr_{u_i \in \{0,1\}^m} \left[ (u_i + u_k) \in S \right] \]

\[ \geq \frac{15}{2^m} > 1 - 2^{-n}. \]

Since \( u_i \)'s are independent, \[ \Pr_{u_i, u_k} \left[ \bigvee_{i=1}^k (S + u_i) \text{ contains } w \right] > 1 - (2^{-n})^k \]

\[ \geq 1 - (2^{-n})^{m+1} = 1 - 2^{-m-n}. \]

I.e., \[ \Pr_{u_i, u_k} \left[ \bigwedge_{i=1}^k w \notin (S + u_i) \right] < 2^{-m} 2^{-n}. \]

Hence \[ \Pr_{u_i, u_k} \left[ \bigwedge_{i=1}^k w \notin (S + u_i) \right] < 2^{-m} 2^{-n} 2^m = 2^{-n} < 1. \]
Therefore

\[ x \in L \Rightarrow \Pr_{u_i, u_k} \left[ \forall \omega \in \{0,1\}^m \left( \left( u_i, u_k \right) \not\in (S + \omega) \right) \right] < 2^{-n} \]

\[ x \notin L \Rightarrow \quad \therefore \quad = 1 \]

\[ \sqrt{\text{Thus}} \quad x \in L \Rightarrow \Pr_{u_i, u_k} \left[ \forall \omega \in \{0,1\}^m \left( \left( u_i, u_k \right) \not\in (S + \omega) \right) \right] > 1 - 2^{-n} \]

\[ \forall x \in L \Rightarrow \quad \therefore \quad = 0 \]

So

\[ x \in L \Rightarrow \exists u_i, u_k \in \{0,1\}^m \forall \omega \in \{0,1\}^m \text{ we U} \left( S, u_i, u_k \right) \]

\[ \Rightarrow \exists u_i, u_k \in \{0,1\}^m \forall \omega \in \{0,1\}^m \text{ we U} \left( S, u_i, u_k \right) \left( \text{for some } \omega \leq \frac{m}{n} + 1, \text{ M}(\omega, u_i, u_k) = 1 \right) \]

poly time

\[ \text{(m+1) \text{ bits}} \]

\[ \leq \left( \frac{m}{n} + 1 \right) ^c \text{ bits} \]

So this is \( \Sigma^p_2 \)-complete.

Another way to think about it is we have

\[ \text{BPP} \subseteq R(\text{NP}) \]

Questions:

Is BPP \( \subseteq \) R(NP)?

Is BPP \( \subseteq \) ZP(NP)?
Suppose $X \in \{0, 1\}$ is a random variable.

1. \[ P[X = 1] = p \quad P[X = 0] = q = 1 - p. \]

Then \[ \mu = E[X] = p \quad \sigma^2 = \text{Var}[X] = E[(X - \mu)^2] = p(1-p) + (1-p)p^2 = p(1-p) = pq \]

\[ \text{Std Dev}(X) = \sigma = \sqrt{pq}. \]

Let $X_i \sim \mathcal{B}(1, p)$ be random variables, $i = 1 \ldots n$.

Let $X = \sum X_i$ be a R.V.

\[ E(X) = \sum_i E(X_i) = np \]

\[ \text{Var}(X) = \sum_i \text{Var}(X_i) = npq \]

\[ \text{Std Dev}(X) = \sigma = \sqrt{npq}. \]

Let $Y = \frac{1}{n} X$. (X as before).

Then \[ E[Y] = \frac{1}{n} \mu = p \]

\[ \sigma_Y^2 = \text{Var}(Y) = E[(Y - \mu)^2] = E\left(\frac{1}{n^2}(X - p)^2\right) = \frac{1}{n^2} npq = \frac{pq}{n} \]

\[ \sigma_Y = \text{Std Dev}(Y) = \sqrt{\frac{pq}{n}}. \]

Application: Suppose $p = \frac{3}{4}$, $q = \frac{1}{4}$.

\[ \sigma_Y^2 = \text{Std Dev}(Y) = \frac{1}{n^2} \sqrt{\frac{pq}{4}} = \frac{1}{n} \cdot \frac{\sqrt{pq}}{2}. \]

\[ P[Y \leq \frac{1}{2}] \leq P\left[Y - \frac{1}{2} \leq \frac{1}{6}\right] \leq \left(\frac{\sqrt{pq}}{6}\right)^2 = \frac{(2.5)^2}{6} = \frac{32}{6} = \frac{8}{n} \]

by Chebyshev's Inequality.

So roughly $n$ times, reduce probability of error to $\frac{1}{n}$.

But Chebyshev's Inequality does not require any independence.
Application 2. Suppose $\rho = \frac{1}{2} + \varepsilon$ and $\beta = \frac{1}{2} - \varepsilon$.

$\mu^* = \frac{1}{2} + \varepsilon$

$\sigma^* = \text{StdDev}(Y) = \sqrt{\frac{1}{n} \text{Var}(X; \varepsilon)} = \frac{1}{\sqrt{n}} \sqrt{\frac{1}{4} - \varepsilon^2} = \frac{1}{\sqrt{n}} \left( \frac{1}{4} - \frac{\varepsilon^2}{2} + O(\varepsilon^3) \right)$

$\Pr\left[ \frac{Y - \mu^*}{\sigma^*} \leq \frac{1}{2} \right] = \Pr\left[ \frac{Y - \frac{1}{2} + \varepsilon}{\sigma^*} \geq \varepsilon \right] \leq \frac{(\sigma^*)^2}{\varepsilon^2} = \frac{1}{\sqrt{n}} \left( \frac{1}{4} - \varepsilon^2 \right) / \varepsilon^2$

$= \frac{1}{\sqrt{n}} \left( \frac{1}{4\varepsilon^2} - 1 \right)$.

So choosing $n \geq \frac{1}{\varepsilon^2}$ means $\Pr\left[ \frac{Y - \mu^*}{\sigma^*} \leq \frac{1}{2} \right] < \frac{1}{4}$.

Choosing $n \geq \frac{1}{4\varepsilon^2}$ means $\Pr\left[ |X - \mu| \geq \frac{1}{2} \right] < \frac{1}{16}$.

Again, we have not needed any independence of $\varepsilon$.

But needed pairwise independence for the calculation of the variance, $\mathcal{V}$, of $X$.

However, we can do better with full mutual independence.

Then $BPP \left[ \frac{1}{n^2} \right] \leq BPP$ by iterating $4^{\frac{1}{\sqrt{N}}} = n^c$ times.
Let $X_1, \ldots, X_n$ be independent, $i.i.d.$ random variables taking values in $\{0, 1\}$ with $X_i = E[X_i] = \mu = \mu_i$. Let $s > 0$.

Then

\[
\Pr \left[ X \geq (1 + s)\mu \right] \leq \left( \frac{e^s}{(1 + s)^{1 + s}} \right)^\mu \leq e^{-\min \left( \frac{s^2}{4}, \frac{s}{2} \right) \cdot \mu}.
\]

\[
\Pr \left[ X \leq (1 - s)\mu \right] \leq \left( \frac{e^s}{(1 - s)^{1 - s}} \right)^\mu.
\]

**Corollary** Let $\nu = \frac{1}{n}$. Suppose each $X_i \sim \text{Bernoulli}(p)$ (same values), $p = \frac{1}{2} + \epsilon$.

\[
\Pr \left[ Y \leq \frac{1}{2} \right] = \Pr \left[ Y \leq \frac{1}{2} \right] \leq \left( \frac{1}{1 - \frac{2\epsilon}{1 + 2\epsilon}} \right)^{\frac{1}{2} + \epsilon}
\]

\[
= \left( \frac{e^s}{(1 - s)^{1 - s}} \right)^n \leq e^{-\min \left( \frac{s^2}{4}, \frac{s}{2} \right) \cdot (\frac{1}{2} + \epsilon) n}.
\]

For $s = 0$, $s \approx 0$, $e^{-\min \left( \frac{s^2}{4}, \frac{s}{2} \right) \cdot (\frac{1}{2} + \epsilon) n} \approx e^{-\frac{s^2}{4} n} \approx 1 - \frac{s^2}{2} n$

So

\[
\Pr \left[ Y \leq \frac{1}{2} \right] \leq \left( 1 - \frac{s^2}{8} \right)^n \leq \left( 1 - \frac{s^2}{8} \right)^{\frac{\nu^2}{\epsilon^2}} \leq \left( \frac{1}{2} \right)^{\frac{\nu^2}{\epsilon^2}}.
\]

It suffices to show $s \leq 2\epsilon$, so

\[
\Pr \left[ Y \leq \frac{1}{2} \right] \leq \left( \frac{1}{2} \right)^{\frac{\nu^2}{\epsilon^2}} \leq \left( \frac{1}{2} \right)^{\frac{\nu^2}{\epsilon^2}}
\]

i.e. $n = \frac{\nu^2}{\epsilon^2}$ puts probability of error at $< \frac{1}{2^n}$.

Thus $BPP_{\frac{1}{2n}} \subseteq BPP$ by using $2 \cdot \frac{2}{(\ln 2)^2} = 4n^2$ structures. But also $BPP \subseteq BPP[1 - \frac{1}{n}]$. Pf: Will be done, $\frac{24n^2}{\epsilon^2}$ times (left as exercise).
Proof of Chernoff Bound

We do the 1−δ case, see Appendix B for the 1+δ case.

New dummy variable t (will equal \ln(1−δ)). Let \( Z = \exp(tX) \)

\[
E[\exp(tx)] = E[\exp(\sum_i tx_i)] = E[\prod_i \exp(-tx_i)]
\]

\[
= \prod_i E[\exp(-tx_i)]
\]

by mutual independence

\[
E[\exp(-tx_i)] = e^{t(1−\rho_i)}(e^{-t})(1−\rho_i) ≤ \exp(\rho_i(e^{-t}))
\]

\[
E[\exp(-tx)] ≤ \prod_i \exp(\rho_i(e^{-t})) = \exp(\mu(e^{-t}))
\]

Now, \( 1−\delta = \frac{\mu(e^{-t})}{e^{t\delta}} ≤ \frac{1}{e^{t\delta}} \) ≤ 1

\[
Pr[X ≤ (1−\delta)\mu] = Pr[X−\mu ≤ S\mu] = Pr[\mu−X ≥ S\mu]
\]

= \[
Pr[\exp(-t\mu−X) ≥ \exp(-tS\mu)]
\]

= \[
Pr[\exp(-t(X−\mu)) ≥ \exp(-tS\mu)]
\]

= \[
Pr[\exp(t(X−\mu)) ≥ \exp(-tS\mu)]
\]

≤ \[
\frac{E[\exp(t(X−\mu))]}{e^{−tS\mu}} \]

by Markov's inequality

= \[
\frac{e^{−t\mu}E[\exp(tX)]}{e^{−t\delta\mu}} = \frac{E[\exp(tX)]}{e^{+t(1−\delta)\mu}}
\]

≤ \[
\frac{\exp(\mu(e^{−t}))}{\exp(+t(1−\delta)\mu)} = \left(\frac{e^{−\delta}}{1−\delta}\right)\mu
\]

\( q.e.d. \)
Lemma \( \frac{e^{-t}}{(1-t)^{1-t}} \leq e^{-\min\left(\frac{5^2}{4}, \frac{t}{2}\right)} \)

Proof: Suffices to show
\[-t - (1-t) \ln(1-t) \leq -\min\left(\frac{5^2}{4}, \frac{t}{2}\right)\]

Power series expansion of \( \ln(1-t) = -(t + \frac{1}{2} t^2 + \frac{1}{3} t^3 + \ldots) \)

\[t^3/3 \quad \text{for } t \leq 1\]

\[t^2/2 \quad \text{for } t > 0 \]

For the \( t > 0 \) case, need to consider \( -\min\left(\frac{5^2}{4}, \frac{t}{2}\right) \).

Corollary: For all \( s \), exists constant \( c_s < 1 \) such that
\[Pr\left[|\sum_{i} X_i - \mu| > s\mu\right] \leq (c_s)^{\mu} .\]

In fact, for \( s < \frac{1}{2} \), \( c_s = \frac{s}{\sqrt{2}} e^{-s^2/2} \) works.
1. Probability results

(a) Markov's inequality: for $X \geq 0$,
\[
\text{Pr}[X \geq k\mu] \leq \frac{1}{k^2}.
\]

(b) Chebyshev's inequality: Let $X$ have mean $\mu$, std dev $\sigma$;
\[
\text{Pr}[X - \mu \geq k\sigma] \leq \frac{1}{k^2}.
\]
\[
\sigma = \sqrt{\text{Var}(X)}
\]

\[
\text{Var}(X) = E[(X - E(X))^2] = E(X^2) - E(X)^2 = \sigma^2
\]

Pf:
\[
\text{Var}(X) = E((X - E(X))^2) = E(X^2) - E(X)^2 = \sigma^2
\]

By Markov,
\[
\text{Pr}[X - \mu \geq k\sigma] \leq \frac{1}{k^2}
\]

Def 4. $X_1$ and $X_2$ are independent if
\[
\text{Pr}[X_1 = a, X_2 = b] = \text{Pr}[X_1 = a] \cdot \text{Pr}[X_2 = b].
\]

\[
\text{Pr}[X_1 = a_1, \ldots, X_n = a_n] = \prod_{i=1}^{n} \text{Pr}[X_i = a_i]
\]

Lemma: If $X_i, X_j$ independent, then $E(X_i X_j) = E(X_i) E(X_j)$ independent.

Thus,
\[
\text{Var}(\sum_{i} X_i) = \sum_{i} \text{Var}(X_i)
\]

Pf:
\[
\text{Pr} \mu = E(X_i), \quad E(\sum X_i) = \sum E(X_i).
\]

Let $X = \sum X_i$. So $\mu = E(X) = \sum a_i$.
\[
\text{Var}(X) = E((X - \mu)^2) = E\left[\left(\sum_{i} X_i - \mu\right)^2\right] = E\left(\sum_{i} (X_i - \mu)^2\right) = \sum_{i} \text{Var}(X_i)
\]

\[= E\left(\sum_{i} (X_i - \mu)^2\right) + \sum_{i} \sigma_i^2\]
Let \( X_i \) be \((0,1)\)-valued RVs, that are pairwise independent.

Let \( p_i = \Pr(X_i = 1) \quad 1-p_i = \Pr(X_i = 0) \).

Let \( X = \sum X_i \). Assume \( p_i \)'s are all equal, and \( > \frac{1}{k} \).

Want to know \( \Pr(\text{false}) \) that \( X \leq \frac{k}{2} h \) \("Majority decides")

Each \( X_i \) has mean \( \mu_i = p_i \) \((>p)\)

an variance \( \text{Var}(X_i) = p_i(1-p_i)^2 \) \((1-p_i)^2 \).

So \( X \) has mean \( kp \), variance \( kp(1-p) \).

and std dev \( \sigma = \sqrt{kp(1-p)} \).

By Chebychev,
\[
\Pr\left[ X \leq \frac{k}{2} h \right] = \Pr\left[ X - kp \leq \frac{1}{2} p h \right] = \Pr\left[ \frac{(X-kp)}{\sigma} \geq \frac{1}{\sqrt{kp(1-p)}} \right] \\
\leq \frac{\sigma^2}{\alpha^2} = \frac{k(p)(1-p)}{(k(p-\frac{1}{2}))^2} \leq \frac{1}{4} \frac{1}{k(p-\frac{1}{2})^2}.
\]

**Corollary** \( \text{BPP}(\frac{1}{2} + \frac{1}{n^c}) = \text{BPP} \).

\( \text{PF} \): Have \( p = \frac{1}{2} + \frac{1}{n^c} \). \( (p-\frac{1}{2})^2 = \frac{1}{n^c} \).

Want \( \frac{1}{4} k \frac{2c}{n^c} \leq \frac{1}{3} \), i.e. \( k \geq \frac{2c}{3} \).

So running the \( \text{BPP}(\frac{1}{2} + \frac{1}{n^c}) \) algorithm \( \frac{n^c}{4} \) times will tell the majority vote works.

**Remark** Similar use of Chebychev gives \( \text{BPP} = \text{BPP}(1 - \frac{1}{n^c}), \forall c \).

But we can do better.

In particular, note the random choices are independent, but the above analysis used only pairwise independence.
Let $X_1, \ldots, X_n$ be independent, $\{0, 1\}$ valued random variables. Let $\delta > 0$. Let $X = \sum X_i$, $\mu = E[X]$, $\sigma^2 = Var(X)$.

$\Pr \left[ \sum X_i \geq (1+\delta)\mu \right] \leq \left( \frac{(1+\delta)^2}{(1+\delta)(1-\delta)} \right)^\mu \leq \left( e^{\frac{\sigma^2}{\mu^2}} \right)^\mu \leq \left( e^{\delta^2 \sigma^2} \right)^\mu$ for all $\delta > 0$.

(d.i) Application to BPP

Then BPP: $BPP(1 - \frac{1}{2^nc})$, for $c \rightarrow \infty$, $c > 1$.

Proof

Do majority vote of $k$ independent trials. Fix $k$.

Here $p = \frac{k}{n}$. $d = \left( \frac{1}{2^nc} \right)$ for $x \in [0, 1]$, $E(k) = \frac{2}{3}$, and $\mu = \frac{2}{3} k$.

$\Pr \left[ X \leq \frac{1}{2} k \right] = \Pr \left[ X \leq \left( 1 - \frac{1}{2} \right) \cdot \frac{2}{3} k \right] \leq \frac{1 - \frac{1}{2}}{1 - \frac{1}{2} \cdot \frac{2}{3} k}$

$\leq \left( \frac{e^{-\frac{1}{2} k}}{(1-\frac{1}{2})^{\frac{1}{2} k}} \right) \leq \left( e^{\frac{3}{4} \sigma^2} \right)^k = \frac{e^{-\frac{k^2}{4}}} \text{ for } k \in \mathbb{R},$ for all $d \in [0, 1]$.

Taking $k = \left( \frac{1}{\log(n)} \right)^c$, for $c > 1$.

$\Pr \left[ X \leq \frac{1}{2} k \right] \leq 2^{-nc}$.

8.6

(d.ii) Proof of Chernoff Bound:

See page "P. 4."
(a) \[ \text{Then: } BPP \leq D/\text{poly} \]

Lemma: on averaging

See p. 35

(f) \[ \text{Then } \text{Super-Gac} \]

\[ BPP \subseteq \text{P/poly} \subseteq \text{P} \]

See p. 37-38

(g) \[ P^{\#P} = P^{\#P} \]

See p. 36