NP Proof:

* A language L is efficiently verifiable if there exists an algorithm V, called a verifier, that takes two inputs x, c, s.t. (The class of P and NP are only concerned w/ decision problems (languages) not search problems)
  1. The computation V(x, c) takes polynomial time with respect to |x| — Efficiency
  2. If $x \in L$, then there exists some certificate c s.t. V(x, c) accepts — Correctness
  3. If $x \notin L$, then V(x, c) rejects for all certificates c

Eg. MAIZE = {(G, s, t) : G is a graph that has a path between s and t} A certificate is a sequence of vertices V on inputs (G = (V, E), s, t), c = (v_1, ..., v_m) =

If $m > |V|$ or $v_1 \neq s$ or $v_e \neq t$, reject — validity checks

for i = 1 to m − 1 :

if $(v_i, v_{i+1}) \notin E$, reject

Accept

Remember to check duplicate if a Hampath-like certificate is given

* The Cook-Levin's Theorem gives us a process of converting an instance of an arbitrary NP language L to an instance of SAT, s.t. if $x \in L$, then $\phi_{V,x} \in$ SAT | if $x \notin L$, then $\phi \notin$ SAT

* The reduction $A \leq_p B$ tells us:

* Proving NP-Hard *	Known fact	conclusion
1. polytime f(...) (make sure input format matches)	A ∈ P	?
	A NP-Hard	B NP-Hard
2. correctness $x \in L \Rightarrow f(x) \in L'$, $x \notin L \Rightarrow f(x) \notin L'$	B ∈ P	A ∈ P
	B NP-Hard	?
3. efficiency		

RSA

* Alice picks prime numbers p and q and publishes the modulus $n = p \cdot q$ $e \in Z_n^+$ s.t. $\gcd(e, \phi(n)) = 1$, $d \equiv e^{-1} \pmod{\phi(n)}$. The pair of keys (e, d) is a matching pair if it satisfies: $\underbrace{e}_{\text{public key}} \cdot \underbrace{d}_{\text{private key}} \equiv 1 \pmod{\phi(n)}$ equiv $e \cdot d = 1 + k\phi(n)$

Alice publishes n and her public key e

Bob sends the cypher text $c = m^e \pmod n$ to Alice

Alice computes $m' \equiv c^d \pmod n$ using her private key d. $m' = c^d = (m^e \bmod n)^d = m^{ed} \bmod n$

$= m^{1+k\phi(n)} \bmod n = m \bmod n$

* Fermat's Little Theorem: Let p be a prime number. Let a be any element of Z_p^+ (or as long as not a multiple of p, where $Z_p^+ = \{1, 2, \cdots, p-1\}$. Then $a^{p-1} \equiv 1 \pmod p$ ⇒ Equiv. Let p be a prime number. Then $a^p \equiv a \pmod p$ for any element a ($\in Z_p$), where $Z_p = \{0, 1, \cdots, p-1\}$. Fermat's little theorem can be extended to the product of two distinct primes, i.e. $p \neq q$. Let a be an element of Z_n. Then, $a^{1+k\phi(n)} \equiv a \pmod n$ where $k \in \mathbb{N}$ and $\phi(n)$ is Euler's totient function, whose value is the number of elements in Z_n^+ that are coprime to n. $\phi(pq) = (p-1)(q-1)$ ($a^{k\phi(n)} \equiv 1 \bmod n$)

RSA Signature

* Alice computes $s = m^d \pmod n$ and sends m and s to Bob

Bob computes $m' = s^e \pmod n$ and verifies that the result is m

$m' \equiv s^e \pmod n \equiv m^{ed} \pmod n \equiv m^{1+k\phi(n)} \pmod n \equiv m \pmod n$

* Generator: An element $g \in Z_p$, where p is prime, is a generator of Z_p if for ev nonzero element $x \in Z_p$ (i.e. every element in Z_p^+), there exists a number $i \in \mathbb{N}$ s.t. $g^i \equiv x \pmod p$

If p is prime, then Z_p has at least one generator

Recall the following languages (unless otherwise specified, all graphs are undirected):

- $L_{\text{HALT}} = \{\langle M, x\rangle : M \text{ is a Turing machine that halts on input } x\}$
- $\overline{L} = \{x \in \Sigma^* : x \notin L\}$
- SAT = {φ : φ is a satisfiable Boolean formula} — a Boolean formula in conjunctive normal form where each clause has exactly three literals
- 3SAT = {φ : φ is a satisfiable 3CNF formula}
- CLIQUE = {(G, k) : G is a graph with a clique of size (at least) k} — a subset of the vertices C ⊆ V s.t. there's an edge between every pair of vertices in clique
- VERTEX-COVER = {(G, k) : G is a graph with a vertex cover of size (at most) k} — a subset of the vertices C ⊆ V s.t. every edge in E is covered by a vertex in C
- Subset-sum = {(S, k) : S is a set of integers that has a subset S* ⊆ S s.t. $\sum S = k$}
- INDEPENDENT-SET = {(G, k) : G is a graph with an independent set of size (at least) k} — no two vertices in the independent set share an edge
- SET-COVER = {(S, S_1, S_2, ..., S_n, k) : $S_i \subseteq S$ for each i and there is a collection of k sets S_i that covers S}
- HAM-CYCLE = {G : G is a graph with a Hamiltonian cycle}
- HAM-PATH = {(G, s, t) : G is an undirected graph w/ a Hamiltonian path between s, t}
- GRAPH-ISOMORPHISM = {(G_1, G_2) : G_1 and G_2 are isomorphic graphs}
- GRAPH-NONISOMORPHISM = {(G_1, G_2) : G_1 and G_2 are non-isomorphic graphs}
- TSP = {(G, k) : G is a weighted graph with a tour of weight at most k}
- MAX-CUT = {(G, k) : G is a graph with a cut of size at least k}

A cut of the graph is the partition of the vertices V into two disjoint sets S and T

$C(S,T) = \{e = (u, v) : e \in E \text{ and } u \in S \text{ and } v \in T\}$ ⇒ size of the cut S, T is $|C(S,T)|$

CNF: $(a \vee b \vee \cdots) \wedge (\neg \vee \neg \vee) \wedge \cdots$

* Diffie-Hellman Protocol: Alice and Bob want to establish a shared secret key k. Alice and Bob first establish public parameters p, a very large prime number, and g, a generator of Z_p. Alice picks a random $a \in Z_p^+$ as private key, computes $A \equiv g^a \pmod p$ (insecure) as her public key, sends A to Bob. Bob picks a random $b \in Z_p^+$ as his private key, computes $B \equiv g^b \pmod p$ as his public key, sends B to Alice. Alice computes $k \equiv B^a \equiv (g^b)^a \equiv g^{ab} \pmod p$ as their shared secret key. Bob computes $k \equiv A^b \equiv (g^a)^b \equiv g^{ab} \pmod p$

* Discrete Logarithm: Let q be a modulus and let g be a generator of Z_p. The discrete logarithm of $x \in Z_q^+$ with respect to the base g is an integer i s.t. $g^i \equiv x \pmod q$ = compute i, given q, g and x (NP, but not known to be NP-Hard)

* Given an input x, a solution y is an α-approximization if: (1) for a maximization problem, $\alpha \cdot OPT(x) \leq value(x, y) \leq OPT(x)$ where $\alpha < 1$ (2) For a min problem: $OPT(x) \leq value(x, y) \leq \alpha \cdot OPT(x)$ where $\alpha > 1$ — α-approximation ratio: the closer α is to one, the ~~better~~ better the approximation

* **Markov's Inequality:** Let X be a nonnegative random variable (i.e. X never takes on a negative value), and let $v > 0$.

$$\Pr[X \geq v] \leq \frac{E[X]}{v}$$

$v > E[X]$ for Markov's to produce meaningful res

* **Reverse Markov's Inequality:** Let X be a random variable that is never larger than some value b, and let $v < b$

$$\Pr[X \geq v] \geq \frac{E[X] - v}{b - v} \qquad \Pr[X \geq v] \geq \frac{E[X] - v}{b - v}$$

Theorem 150 (Hoeffding's Inequality – Upper Tail) *Let $X = X_1 + \cdots + X_n$, where the X_i are independent indicator random variables with $\mathbb{E}[X_i] = p_i$. Let*

assuming the random variable is an average of indicator variables

$$p = \mathbb{E}\left[\frac{1}{n}X\right] = \frac{1}{n}\sum_i p_i$$

Let $\varepsilon > 0$ be a deviation from the expectation. Then

$$\Pr\left[\frac{1}{n}X \geq p + \varepsilon\right] \leq e^{-2\varepsilon^2 n}$$

Theorem 151 (Hoeffding's Inequality – Lower Tail) *Let $X = X_1 + \cdots + X_n$, where the X_i are independent indicator random variables with $\mathbb{E}[X_i] = p_i$. Let*

$$p = \mathbb{E}\left[\frac{1}{n}X\right] = \frac{1}{n}\sum_i p_i$$

Let $\varepsilon > 0$ be a deviation from the expectation. Then

$$\Pr\left[\frac{1}{n}X \leq p - \varepsilon\right] \leq e^{-2\varepsilon^2 n}$$

Theorem 152 (Hoeffding's Inequality – Combined) *Let $X = X_1 + \cdots + X_n$, where the X_i are independent indicator random variables with $\mathbb{E}[X_i] = p_i$. Let*

$$p = \mathbb{E}\left[\frac{1}{n}X\right] = \frac{1}{n}\sum_i p_i$$

The expectation of $\frac{X}{n}$ is independent of n. $E[X] = nE[X_i]$, whereas $E[\frac{X}{n}] = E[X_i]$

Let $\varepsilon > 0$ be a deviation from the expectation. Then

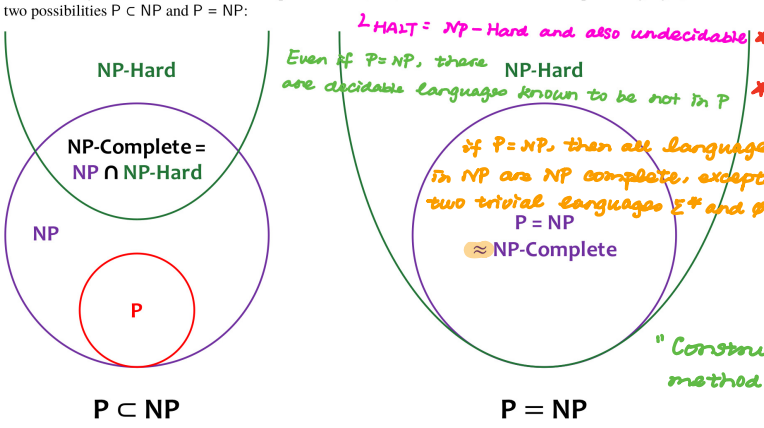
$$\Pr\left[\left|\frac{1}{n}X - p\right| \geq \varepsilon\right] \leq 2e^{-2\varepsilon^2 n}$$

Corollary 147 *Let $X = X_1 + \cdots + X_n$ be the sum of n independent, identically distributed random variables. Let $\varepsilon > 0$ be a deviation from the expectation $\mathbb{E}[X/n] = \mathbb{E}[X_i]$. Then*

$$\Pr\left[\left|\frac{1}{n}X - \mathbb{E}[X_i]\right| \geq \varepsilon\right] \leq \frac{\mathrm{Var}(X_i)}{\varepsilon^2 n}$$

optimization search { max: max clique; min: min vertex cover; exact: satisfying assignment or a Hamiltonian path

The following demonstrates the relationships between P, NP, and NP-Hard and NP-Complete languages, under th two possibilities P ⊂ NP and P = NP:



Even if P = NP, there are decidable languages known to be not in P

if P = NP, then all languages in NP are NP complete, except two trivial languages Σ^* and $\emptyset$

* If a language L is in P and L_1 is a nontrivial language (that is, $L_1 \neq \emptyset$ and $L_1 \neq \Sigma^*$), then $L \leq_p L_1$. For any input $x \in L$, an efficient map can run L's decider on x to determine in polynomial time if x is in L or not. Then if x is in L, output some fixed value in L_1. If not in L, then output some fixed value not in L (that's why L_1 can't be the trivial languages)

* A language L in P is NP-Hard if and only if P = NP

Theorem 199 (Extended Fermat's Little Theorem) *Let $n \in \mathbb{N}$ be a natural number such that $n \geq 2$. Le a* witness *in the range $1 \leq a \leq n - 1$. Then:*

- *If n is prime, then $a^{n-1} \equiv 1 \pmod{n}$ for any witness a.*
- *If n is composite and n is not a* Carmichael number, *then $a^{n-1} \equiv 1 \pmod{n}$ for at most half the witn $1 \leq a \leq n - 1$.*

$\mathrm{coNP} = \{\bar{L} \mid L \in \mathrm{NP}\}$

* It is not known whether there exists a coNP language that is NP-Hard. If that is the case, we would prove NP = coNP (unknown)

we already know

* If P = NP, then because P is closed under set complement, so is NP, therefore NP = coNP

* If NP ≠ coNP, then P ≠ NP

* **Variance:** Suppose X is a random variable with expectation $E[X]$. Then the variance of X is defined as $\mathrm{Var}(X) = E[(X - E[X]^2)]$ or equivalently $\mathrm{Var}(X) = E[X^2] - E[X]^2$

* Standard deviation – $\sigma(X) = \sqrt{\mathrm{Var}[X]}$

* expectation: $E[X] = \sum_i a_i \cdot \Pr[X = a_i]$

* Suppose X is a random variable. Then for any constant c, $\mathrm{Var}(cX) = c^2 \mathrm{Var}(X)$

* Let X and Y be independent R.V., then $E[XY] = E[X] \cdot E[Y]$ and $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

Absolute value is not required just not the tightest bound

* Chebyshev's Inequality: Let X be a R.V. and let $a > 0$. Then, $\Pr[|X - E[X]| \geq a] \leq \frac{\mathrm{Var}(X)}{a^2}$

* L_{HALT} = NP-Hard and also undecidable

* The decision problem is no harder than the search problem

* Given the decider, we want to find a max/min object:

S_1 on input $G = (V, E)$:
For $k = |V|$ to 0:
  If $D(G, k)$ accepts, **return** k

find the size of the max clique

S_2 on inputs $G = (V, E), k$:
For each $v \in V$:
  $G' := (V \setminus \{v\}, E \setminus \{(u, v) \in E\})$ [remove v and all adjacent edges]
  If $D(G', k)$ accepts: this implies v is
  $G := G'$ decider for Clique not necessary
Return $V(G)$ [return the remaining vertices in G]

"Destructive" method

S_2' on inputs $G = (V, E), k$:
  $C := \emptyset$
  For each $v \in V$:
    $G' := neighborhood(G, v)$
    If $D(G', k-1)$ accepts:
      $C := C \cup \{v\}$
      $G := G'$
      $V := V(G')$
      $k := k - 1$
Return C

could be dependent or independent

"Constructive" method:

* An NP-Hard search problem has an efficient algo iff corresponding decision problem is efficiently solvable

* **Union Bound:** Let $E_1, E_2, \ldots, E_m$ be a sequence of events over the same sample space. Then, $\Pr\left[\bigcup_{i=1}^{m} E_i\right] \leq \sum_{i=1}^{m} \Pr[E_i]$

A common type of random variable is an *indicator* random variable, also called a *0-1 random variable*. As the la name suggests, it is a random variable that only takes on the values 0 or 1. If X is an indicator and $\Pr[X = 1]$ = we have $\Pr[X = 0] = 1 - p$, and

$$\mathbb{E}[X] = 0 \cdot \Pr[X = 0] + 1 \cdot Pr[X = 1]$$
$$= \Pr[X = 1] = p$$

If we can define a random variable Y as the sum of n indicator random variables X_i, each with the same probab $\Pr[X_i = 1] = p$, then:

$$\mathbb{E}[Y] = \sum_i \mathbb{E}[X_i]$$
$$= \sum_i \Pr[X_i = 1]$$
$$= np$$

* We don't know an efficient algorithm to solve NP-Complete Problems on quantum computers.

Behavior	Finite Automata	Turing Machines
Tape alphabet a strict superset of input alphabet	No	Yes
Can have zero, or more than one, accept state(s)	Yes	No
Has a reject state	No	Yes
Terminates when accept/reject state is reached	No	Yes
Terminates when input is exhausted	Yes	No
Direction head can move	Right	Right or Left
Can write to the tape	No	Yes

Theorem 14 (Master Theorem) *Let $T(n)$ be a recurrence with one of the following forms, where $k \geq 1, b > 1, d \geq 0$ are constants:*

$$T(n) = kT\left(\frac{n}{b}\right) + O(n^d)$$
$$T(n) = kT\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + O(n^d)$$
$$T(n) = kT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

Then:

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \\ O(n^d \log n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$

The relationship between k, b, and d can alternatively be expressed in logarithmic form. For instance,

$$k/b^d < 1$$
$$k < b^d$$
$$\log_b k < d$$

taking the $\log_b$ of both sides in the last step. We end up with:

$$T(n) = \begin{cases} O(n^d) & \text{if } \log_b k < d \\ O(n^d \log n) & \text{if } \log_b k = d \\ O(n^{\log_b k}) & \text{if } \log_b k > d \end{cases}$$

Theorem 15 *Let $T(n)$ be a recurrence with one of the following forms, where $k \geq 1, b > 1, d \geq 0, w \geq 0$ are constants:*

$$T(n) = kT\left(\frac{n}{b}\right) + O(n^d \log^w n)$$
$$T(n) = kT\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + O(n^d \log^w n)$$
$$T(n) = kT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d \log^w n)$$

Then:

$$T(n) = \begin{cases} O(n^d \log^w n) & \text{if } k/b^d < 1 \\ O(n^d \log^{w+1} n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$

* P and NP are both countable $\Leftarrow$ ∵ decidable
* A power set of a finite set is finite, so a subset must be countable Eg. A subset of the power set of $\{0,1,3\}$
* A diagonalization argument can be used to show the set of languages over the given alphabet is uncountable Eg. The set of languages over the given alphabet $\{0,1,3\}$ (countably infinite finite-length strings)
* Σ^* is the set of all finite-length strings over an alphabet Σ (countably infinite!): $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$ Eg. $\{0,1\}$ $L \subseteq \Sigma^*$ A language L is just a set of finite-length strings over some alphabet Σ
* A subset of unrecognizable language can be both recognizable or not recognizable
* The set of all simple graphs over a finite vertex set is finite. Eg. The set of all simple graphs $G = (V, E)$ (where $|V|$ is finite) which have an MST
* The set of all decidable / recognizable languages is countably infinite
* The set of all undecidable / unrecognizable language is uncountably infinite
* Any language can be trivially Turing reduced to its complement
* A decidable language can be a subset of an undecidable language. An undecidable language can be a subset of a decidable language Eg. $\emptyset \subseteq L_{HALT} \subseteq \Sigma^*$
* If a language L can not be decided in efficiently. exponential time. L can not be verified

$\overline{L_{HALT}} \subset \overline{L_{ACC}}$ and $\overline{L_{HALT}}$ is unrecognizable

$L_{reject} = \{(\langle M \rangle, x) : M \text{ rejects } x\} \subset \overline{L_{ACC}}$ but is recognizable, similarly, $\emptyset \subset \overline{L_{ACC}}$ is recognizable

* Suppose $A \leq_T B$. then if B is decidable, A is also decidable
* Suppose $A \leq_T B$. then if A is undecidable, B is also undecidable
* If A is decidable, then $A \leq_T B$ for any language B
* The class of decidable / recognizable language is closed under intersection and union.
* The class of decidable language is closed under complement
* If a language A and its complement $\bar{A}$ are both recognizable, then $\bar{A}$ is decidable. If a language A is undecidable but recognizable, then $\bar{A}$ is unrecognizable
* A language L is co-recognizable if $\bar{L}$ is recognizable.
* A decidable language is both recognizable and co-recognizable.
* unrecognizable language = $\overline{L_{ACC}}$ $(\langle M \rangle, x)$ $\overline{L_{HALT}}$ $(\langle M \rangle, x)$
* $L \leq_T \Sigma^*$ if and only if L is decidable

Claim 127 (Averaging Argument) *Suppose X is a random variable over a sample space Ω. Then by the* averaging argument*:*

- *There is at least one outcome $\omega \in \Omega$ such that*

$$X(\omega) \geq \mathbb{E}[X]$$

- *There is at least one outcome $\omega \in \Omega$ such that*

$$X(\omega) \leq \mathbb{E}[X]$$

* We can conclude from this result that any exact 3CNF formula (the literals have distinct variables) has an assignment that satisfies at least $\frac{7}{8}$ of its clauses

expected # satisfied clauses

8. *Version 1:*
Suppose a COVID-19 test always comes back positive if a person has COVID-19 and comes back negative with probability ≥ 0.6 if the person does not. If a person tests positive, what is the minimum number of **additional** tests they need to take to know that they have COVID-19 with at least 98% confidence? Assume that the results of each test is independent of all the others.

A. 6
B. 4
C. 2
D. 3
E. 5

Solution: Let A = "a person doesn't have COVID-19 but the result gives positive back." $Pr(A) \leq 0.4$. In order to boost the probability of confidence, we use the strategy of repeat the test several times, more specifically: we repeat the test k times, and if any of the test returns negative, then return negative. Therefore, the probability that a person doesn't get COVID-19 but will still get a positive is $(Pr(A)^k)$. Therefore the probability of success is $1 - (Pr(A)^k)$, and we want is to be at least 98%, *i.e.* $1 - (Pr(A)^k) \geq 0.98 \implies 0.4^k \leq 0.02 \implies k \geq 5$. Hence we would need 4 additional test to guarantee to desired probability of success.

Solution: We can show that $\text{SAT} \leq_p L_{\text{HALT}}$. Let f be a function that maps an SAT instance to an instance of L_{HALT}. f outputs the pair (M_ϕ, ε), where M_ϕ is a Turing machine. M_ϕ behaves as follows on input x:

Proving that L_{HALT} is NP-Hard, but not NP-Complete

M_ϕ = "On input x:

1. Loop through all possible Boolean variable assignments a over the variables in ϕ.
2. If $\phi(a)$ is true, accept x.
3. If no assignment yields true, loop."

Notice that f outputs a Turing machine that is linear in size with respect to ϕ, so f is an efficient mapping.

Suppose $\phi \in$ SAT. Then there is some variable assignment a such that $\phi(a)$ is true, and M eventually accepts ε, so $(M, \varepsilon) \in L_{\text{HALT}}$.

Suppose $\phi \notin$ SAT. Then ϕ evaluated on *any* assignment a is false, and M eventually loops on ε, so $(M, \varepsilon) \notin L_{\text{HALT}}$.

Therefore $\text{SAT} \leq_p L_{\text{HALT}}$, and L_{HALT} is NP-Hard. Since L_{HALT} is undecidable, it is not in NP – NP $\subseteq$ EXP, so all NP languages are decidable in exponential time.

Algorithm 27 (Floyd-Warshall)

Algorithm *Floyd-Warshall*$(G = (V, E))$:
  Let $d^0(i, j) = weight(i, j)$ **for all** $i, j \in V$
  For $k = 1$ to $|V|$:
    For all $i, j \in V$:
      $d^k(i, j) := \min\{d^{k-1}(i, j),\ d^{k-1}(i, k) + d^{k-1}(k, j)\}$
  Return d^n

* Kruskal : $O(E \log E)$

DP

time : $O(V^3)$

space : $O(V^2)$

the length of the shortest path between vertices i and j, where the path is only allowed to go through intermediate vertices in $\{1, \dots k\}$

Claim 168 *Given $x > y \geq 0$, the extended Euclidean algorithm returns a triple (g, a, b) such that*

$$g = ax + by$$

and $g = \gcd(x, y)$.

$1 = ax + by$

The condition for the modular inverse to exist is $\gcd(a, b) = 1$.

$$ax \equiv 1 \pmod{y}$$
$$by \equiv 1 \pmod{x}$$

by definition of modular arithmetic. Thus, the extended Euclidean algorithm computes a as the inverse of x modulo y and b as the inverse of y modulo x, when these inverses exist.

Solution: Unknown. We do not know whether (the decision version of) integer factorization is NP-Complete, so even if P $\neq$ NP, is is possible for integer factorization to be in P.

Solution: True. We saw that the number of samples depends on the accuracy of estimation (confidence level and range of error) and the number of possible answers, but not on the population size.

Definition 172 (One-time Pad) A *one-time pad* is an encryption technique that relies on a key with the following properties:

- The key is a random string at least as long as the plaintext.
- The key is preshared between the communicating parties over some secure channel.
- The key is only used once (hence the name *one-time pad*).

Karatsuba Algorithm :

$N_1 = a \cdot 10^{n/2} + b$

$N_2 = c \cdot 10^{n/2} + d$

$$N_1 \times N_2 = a \times c \cdot 10^n + (a \times d + b \times c) \cdot 10^{n/2} + b \times d$$

Rather than computing $a \times c$, $a \times d$, $b \times c$, and $b \times d$ directly, we compute the following three terms:

$$m_1 = (a + b) \times (c + d)$$
$$m_2 = a \times c$$
$$m_3 = b \times d$$

Observe that $m_1 = a \times c + a \times d + b \times c + b \times d$, and we can subtract m_2 and m_3 to obtain:

$$m_1 - m_2 - m_3 = a \times c + a \times d + b \times c + b \times d - a \times c - b \times d$$
$$= a \times d + b \times c$$

This is exactly the middle term in our expansion for $N_1 \times N_2$. Thus:

$$N_1 \times N_2 = a \times c \cdot 10^n + (a \times d + b \times c) \cdot 10^{n/2} + b \times d$$
$$= m_2 \cdot 10^n + (m_1 - m_2 - m_3) \cdot 10^{n/2} + m_3$$

skip list :

To search for an item, we start at the sentinel node at the top left. We maintain the invariant that we are always located at a node whose value is less than the item we are looking for (or is a sentinel head node). Then in each step, we check the node to the right. If its value is greater than the item we are looking for, or if the right pointer is null, we move downward; however, if we are on the lowest level, we cannot move downward, so the item is not in the list. If the value of the node to the right is less than the item, we move to the right. If the value is equal to the item, we have found the item and terminate the search. The following illustrates a search for the item 0 in the list above:

For $i > 1$, define X_{ij} to be an indicator R.V. for whether or not the j^{th} element is replicated to level i.

$\Pr[X_{ij} = 1] = \frac{1}{2^{i-1}}$ (expected). The total number of elements on level i is just the sum of the indicators X_{ij}. $\frac{n}{2^{i-1}}$. The expected number of levels is $O(\log n)$

The expected number of nodes encountered on a level i is $O(1)$. The total across all levels is $O(\log n)$

Since insertion does only a constant number of operations in addition to the search, it too does $O(\log n)$ operations, and the same reasoning holds for deletion as well. We achieve this expected runtime with a much simpler implementation than a self-balancing search tree.

Lemma 2.1. *Let L be a language and let A be a randomized algorithm such that*

- $x \in L \implies \Pr[A \text{ accepts } x] \geq c$
- $x \notin L \implies A$ ***always*** *rejects x*

where c is a real number such that $0 < c \leq 1$. Then, for any constant $0 < \varepsilon < 1$, there exists a one-side error algorithm B such that

- $x \in L \implies \Pr[B \text{ accepts } x] \geq 1 - \varepsilon$
- $x \notin L \implies B$ ***always*** *rejects x*

Proof. Our new algorithm B is:
On input x:

1. Repeat k times [k will be determined later]:
   (a) Run A on x.
   (b) If A accepts, *accept*.
2. *Reject*.

no false positives, have false neg

B rejects x if and only if k independent runs of A all rej x

$\Pr(B \text{ rejs } x) = \prod_{i=1}^{k} \Pr(A \text{ rej } x)$

$= \prod_{i=1}^{k} (1 - \Pr(A \text{ accepts } x))$

$x \in L$ so $\Pr(A \text{ accepts } x) \geq c$

$\Pr(B \text{ rejs } x) \leq \prod_{i=1}^{k} (1-c) = (1-c)^k$

Algorithm 167 (Extended Euclidean Algorithm)

Algorithm *ExtendedEuclid*(x, y) :
  If $y = 0$ **return** $(x, 1, 0)$
  $(g, a', b') := ExtendedEuclid(y, x \bmod y)$
  Return $\left(g, b', a' - b' \cdot \left\lfloor \frac{x}{y} \right\rfloor\right)$

$13a \equiv 1 \pmod{21}$

$a \equiv 13^{-1} \pmod{21}$

Example 170 We compute the inverse of 13 modulo 21 using the extended Euclidean algorithm. Since 13 and 21 are coprime, such an inverse must exist.

We keep track of the values of x, y, g, a, and b at each recursive step of the algorithm. First, we trace the algorithm from the initial $x = 21, y = 13$ down to the base case:

Finding $13^{-1} \bmod 60$:

Step 1 :

$60 = 13 \times 4 + 8$

$13 = 8 \times 1 + 5$

$8 = 5 \times 1 + 3$

$5 = 3 \times 1 + 2$

$3 = 2 \times 1 + 1$

stop here when we get a remainder of 1

$13 \cdot a \equiv 1 \pmod{60}$

$13a = 60k + 1$

$13a + 60k = 1$

condition for the modular inverse to exist

solve for remainders

Step	x	y
0	21	13
1	13	8
2	8	5
3	5	3
4	3	2
5	2	1
6	1	0

Step 2 :

$8 = 60 - 13 \times 4$

$5 = 13 - 8 \times 1$

$3 = 8 - 5 \times 1$

$2 = 5 - 3 \times 1$

$1 = 3 - 2 \times 1$

Step 3 :

$5 = 13 - (60 - 13 \times 4)$

$= 13 - 60 + 13 \times 4$

$= 13 \times 5 - 60$

$3 = (60 - 13 \times 4) - (13 \times 5 - 60)$

$= 60 - 13 \times 4 - 13 \times 5 + 60$

$= 60 \times 2 - 13 \times 9$

$2 = (13 \times 5 - 60) - (60 \times 2 - 13 \times 9)$

$= 13 \times 5 - 60 - 60 \times 2 + 13 \times 9$

$= 13 \times 14 - 60 \times 3$

$1 = 60 \times 2 - 13 \times 9$

$-13 \times 14 + 60 \times 3$

$= 60 \times 5 - 23 \times 13$

$-23 \bmod 60 = 37 \bmod 60$

We can then trace the algorithm back up, computing the return values g, a, b:

Step	x	y	g	a	b
6	1	0	1	1	0
5	2	1	1	0	$1 - 0 \cdot \left\lfloor \frac{2}{1} \right\rfloor = 1$
4	3	2	1	1	$0 - 1 \cdot \left\lfloor \frac{3}{2} \right\rfloor = -1$
3	5	3	1	-1	$1 - (-1) \cdot \left\lfloor \frac{5}{3} \right\rfloor = 2$
2	8	5	1	2	$-1 - 2 \cdot \left\lfloor \frac{8}{5} \right\rfloor = -3$
1	13	8	1	-3	$2 - (-3) \cdot \left\lfloor \frac{13}{8} \right\rfloor = 5$
0	21	13	1	5	$-3 - 5 \cdot \left\lfloor \frac{21}{13} \right\rfloor = -8$

Thus, we have $by = -8 \cdot 13 \equiv 1 \pmod{21}$. Translating b to an element of $\mathbb{Z}_{21}$, we get $b = -8 \equiv 13 \pmod{21}$, which means that 13 is its own inverse modulo 21. We can verify this:

$$13 \cdot 13 = 169 = 8 \cdot 21 + 1 \equiv 1 \pmod{21}$$

Quicksort : The expected number of comparisons in randomized quicksort is $O(n \log n)$

In general, for two elements a and b, the elements get compared if and only if the first pivot chosen among the elements with value in the range $[a, b]$ is either a itself or b. As long as pivots are chosen outside of this range, a and b stay in the same partition (since the chosen pivot is either less than both a and b or greater than both of them), so they have a possibility of being compared. At the same time, since only pivots are compared to other elements, as long as neither a nor b are chosen as a pivot, they are not compared. When a pivot is selected in the range $[a, b]$, there are two possibilities:

- a or b is selected as the pivot, in which case they are compared, since the pivot is compared to every element in the same subarray. They are not subsequently compared again, since as we observed above, the pivot element is not placed into either partition.
- An element c where $a < c < b$ is chosen as the pivot, in which case a and b are partitioned into separate subarrays and therefore never compared.

Definition 173 (Caesar Cipher) Let $m = m_1 m_2 \dots m_n$ be a plaintext message. Let s be a single symbol to be used as the key. Then in a Caesar cipher, m is encrypted as

$$E_s(m) = c_1 c_2 \dots c_n, \text{ where } c_i \equiv m_i + s \pmod{26}$$

and a ciphertext c is decrypted as

$$D_s(c) = d_1 d_2 \dots d_n, \text{ where } d_i \equiv c_i - s \pmod{26}$$

The result is $D_s(E_s(m)) = m$.