Theory of Computation
The Class NP

Search vs. Verification

Which tasks are easier?

- Writing a screenplay
- Doing a homework assignment
- Proving a new theorem
- Finding 1000 Facebook users who are all friends

- Reviewing a movie
- Grading a homework assignment
- Checking that a proof is valid
- Checking if 1000 Facebook users are all friends

3-CNF Formulas

Def: A **3-Conjunctive Normal Form (3-CNF)** formula is a CNF formula with at most 3 variables in each clause

Which of the following formulas are 3-CNF formulas?

A) $F = (x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6)$
B) $F = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_5)$
C) $F = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_4)$
D) $F = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2)$

3-CNF Formulas

Def: A **3-Conjunctive Normal Form (3-CNF)** formula is a CNF formula with at most 3 variables in each clause

Which of the following formulas are 3-CNF formulas?

A) $F = (x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6)$
B) $F = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_5)$ ✓
C) $F = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_4)$ ✓
D) $F = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2)$

The language 3-SAT

$$3\text{-SAT} = \{F | F \text{ is a satisfiable 3-CNF Formula}\}$$

Which of the following formulas are in 3-SAT?

A) $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5)$
B) $F = (x_1 \wedge x_2 \wedge x_3) \wedge (x_4 \wedge x_5 \wedge x_6)$
C) $F = (x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1)$
D) $F = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$
E) $F = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_4) \wedge (x_2 \vee x_6) \wedge (\neg x_1)$

The language 3-SAT

$$\text{3-SAT} = \{F | F \text{ is a satisfiable 3-CNF Formula}\}$$

Which of the following formulas are in 3-SAT?

A) $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5)$ ✓
B) $F = (x_1 \wedge x_2 \wedge x_3) \wedge (x_4 \wedge x_5 \wedge x_6)$
C) $F = (x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1)$
D) $F = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$
E) $F = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_4) \wedge (x_2 \vee x_6) \wedge (\neg x_1)$ ✓

3-SAT $\in$ EXP

Construct an exponential-time decider for 3-SAT

Input: Formula F with n variables and m clauses

1. For every possible truth assignment A do the following:
   1.1 Check if A satisfies the formula.
   1.2 If it does, accept F
2. If every truth assignment fails, reject F

3-SAT $\in$ EXP

Construct an exponential-time decider for 3-SAT

Input: Formula F with n variables and m clauses

1. For every possible truth assignment A do the following:
   1.1 Check if A satisfies the formula.
   1.2 If it does, accept F
2. If every truth assignment fails, reject F

- 2^n truth assignments (2 choices for each variable)

3-SAT $\in$ EXP

Construct an exponential-time decider for 3-SAT

Input: Formula F with n variables and m clauses

1. For every possible truth assignment A do the following:
   1.1 Check if A satisfies the formula.
   1.2 If it does, accept F
2. If every truth assignment fails, reject F

- 2^n truth assignments (2 choices for each variable)
- Can check whether an assignment works in polynomial time

3-SAT $\in$ EXP

Construct an exponential-time decider for 3-SAT

Input: Formula F with n variables and m clauses

1. For every possible truth assignment A do the following:
   1.1 Check if A satisfies the formula.
   1.2 If it does, accept F
2. If every truth assignment fails, reject F

- 2^n truth assignments (2 choices for each variable)
- Can check whether an assignment works in polynomial time
- $O(2^n) \cdot \text{poly-time} \in \text{EXP}$

$3\text{-SAT} \in \mathrm{P}$?

- Can 3-SAT be solved in polynomial time?

3-SAT $\in$ P?

3-SAT $\in$ P?

- Can 3-SAT be solved in polynomial time?
- Generally believed to be impossible
- But we also have reason to believe that 3-SAT is easier than some other problems in EXP
- 3-SAT can be **verified** in polynomial time

3-SAT search vs. verification

Is the following 3-CNF formula satisfiable?

$$F = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_1) \wedge (x_4)$$

Which of the following truth assignments satisfy F?

A) $x_1 = x_2 = \text{TRUE}, x_3 = x_4 = \text{FALSE}$
B) $x_1 = x_4 = \text{TRUE}, x_2 = x_3 = \text{FALSE}$
C) $x_1 = x_2 = x_3 = x_4 = \text{FALSE}$
D) $x_1 = x_2 = x_3 = x_4 = \text{TRUE}$
E) $x_2 = x_3 = x_4 = \text{TRUE}, x_1 = \text{FALSE}$

3-SAT search vs. verification

Is the following 3-CNF formula satisfiable?

$$F = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$
$$\wedge (x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_1) \wedge (x_4)$$

Which of the following truth assignments satisfy F?

A) $x_1 = x_2 = \mathrm{TRUE}, x_3 = x_4 = \mathrm{FALSE}$
B) $x_1 = x_4 = \mathrm{TRUE}, x_2 = x_3 = \mathrm{FALSE}$
C) $x_1 = x_2 = x_3 = x_4 = \mathrm{FALSE}$
D) $x_1 = x_2 = x_3 = x_4 = \mathrm{TRUE}$ ✓
E) $x_2 = x_3 = x_4 = \mathrm{TRUE}$, $x_1 = \mathrm{FALSE}$ ✓

Verifiers

Let L be a formal language. A **verifier** for L is a machine V with the following properties:

1. V takes two inputs: w and c
2. If $w \in L$, then V accepts $\langle w, c \rangle$ for some string c
3. If $w \notin L$, then V rejects $\langle w, c \rangle$ for all c

The string c is sometimes called a **certificate**, **witness**, or **proof** that $w \in L$

Poly-time verifiers

- We say V is a **poly(nomial)-time verifier** if it runs in polynomial time
- Note that this means that the certificate c must be polynomially bounded
  - $|c| \leq |w|^k$
- We say L is **poly(nomial)-time verifiable** if it has a poly-time verifier V
  - This means that every $w \in L$ has a polynomial-length certificate

3-SAT is poly-time verifiable

We'll construct a poly-time verifier V

1. V takes input $\langle F, A \rangle$, where F is a 3-CNF formula and A is a truth assignment
2. For each clause C_i do the following:
   2.1 For each variable x_i in the clause, check if x_i is assigned to TRUE (or FALSE if x_i is negated)
   2.2 If none of the variables are TRUE, the clause is not satisfied. Reject $\langle F, A \rangle$
3. If all clauses are satisfied, accept $\langle F, A \rangle$

3-SAT is poly-time verifiable

We'll construct a poly-time verifier V

1. V takes input $\langle F, A \rangle$, where F is a 3-CNF formula and A is a truth assignment
2. For each clause C_i do the following:
   2.1 For each variable x_i in the clause, check if x_i is assigned to TRUE (or FALSE if x_i is negated)
   2.2 If none of the variables are TRUE, the clause is not satisfied. Reject $\langle F, A \rangle$
3. If all clauses are satisfied, accept $\langle F, A \rangle$

- $|A| = O(n)$ (one truth value per variable)

3-SAT is poly-time verifiable

We'll construct a poly-time verifier V

1. V takes input $\langle F, A \rangle$, where F is a 3-CNF formula and A is a truth assignment
2. For each clause C_i do the following:
   2.1 For each variable x_i in the clause, check if x_i is assigned to TRUE (or FALSE if x_i is negated)
   2.2 If none of the variables are TRUE, the clause is not satisfied. Reject $\langle F, A \rangle$
3. If all clauses are satisfied, accept $\langle F, A \rangle$

- $|A| = O(n)$ (one truth value per variable)
- $O(m)$ loop iterations

3-SAT is poly-time verifiable

We'll construct a poly-time verifier V

1. V takes input $\langle F, A \rangle$, where F is a 3-CNF formula and A is a truth assignment
2. For each clause C_i do the following:
   2.1 For each variable x_i in the clause, check if x_i is assigned to TRUE (or FALSE if x_i is negated)
   2.2 If none of the variables are TRUE, the clause is not satisfied. Reject $\langle F, A \rangle$
3. If all clauses are satisfied, accept $\langle F, A \rangle$

- $|A| = O(n)$ (one truth value per variable)
- $O(m)$ loop iterations
- $O(n)$ to look up the truth value of a variable

3-SAT is poly-time verifiable

We'll construct a poly-time verifier V

1. V takes input $\langle F, A \rangle$, where F is a 3-CNF formula and A is a truth assignment
2. For each clause C_i do the following:
    2.1 For each variable x_i in the clause, check if x_i is assigned to TRUE (or FALSE if x_i is negated)
    2.2 If none of the variables are TRUE, the clause is not satisfied. Reject $\langle F, A \rangle$
3. If all clauses are satisfied, accept $\langle F, A \rangle$

- $|A| = O(n)$ (one truth value per variable)
- $O(m)$ loop iterations
- $O(n)$ to look up the truth value of a variable
- $O(m \cdot n)$ = poly-time verification

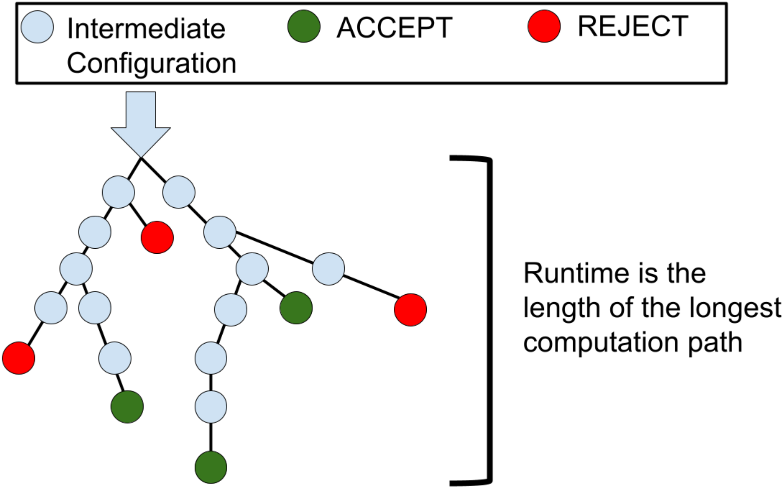
Nondeterministic Machines

- **Recall**: We have seen nondeterministic finite automata (NFAs) and nondeterministic Turing machines (NTMs)
- At each step, the machine "guesses" what the optimal computation path is
- The machine accepts w if there exists at least one accepting computation path
- Nondeterminism doesn't make our machines more *robust*
- **Does nondeterminism make our machines faster?**

Nondeterministic Runtimes

- Deterministic machines *always* behave the same way on the same input
- Nondeterministic machines may have different behavior on the same input!
- **Def:** a nondeterministic TM runs in time $T(n)$ if all computation paths take at most $O(T(n))$ steps
  - A nondeterministic TM runs in polynomial time if the length of longest computation path is always polynomially bounded
  - It only takes a polynomial amount of time to "guess" the solution

Nondeterministic Runtimes



The class NP

- **Def:** The class $\mathrm{NTIME}(T(n))$ is the set of all languages that can be decided by a *nondeterministic* TM in time $T(n)$
- **Def:** The class NP is the set of all languages that can be decided in nondeterministic polynomial time

$$\mathrm{NP} = \bigcup_c \mathrm{NTIME}(T(n^c))$$

3-SAT $\in$ NP

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

3-SAT $\in$ NP

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

Correctness

- If F is satisfiable, at least one computation path will guess a satisfying assignment
- If F is not satisfiable, every computation path will reject

$3\text{-SAT} \in \text{NP}$

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

Runtime:

3-SAT $\in$ NP

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

Runtime:

- $O(n)$ time to guess a truth assignment

$3\text{-SAT} \in \text{NP}$

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

Runtime:

- $O(n)$ time to guess a truth assignment
- Poly-time to check the truth assignment

3-SAT $\in$ NP

We will construct that a nondeterministic TM to decide 3-SAT in polynomial time

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

Runtime:

- $O(n)$ time to guess a truth assignment
- Poly-time to check the truth assignment
- $O(n) \cdot$ poly-time $\in$ NP

NP and verification

Let's re-examine the nondeterministic 3-SAT algorithm

Input: A formula F with n variables and m clauses

1. *Nondeterministically guess* truth assignment A
2. Check if A satisfies the formula F
3. Accept F if A satisfies F. Reject otherwise

NP and verification

Let's re-examine the nondeterministic 3-SAT algorithm

Input: ~~A formula F with n variables and m clauses~~
A string w

1. ~~*Nondeterministically guess* truth assignment A~~
   Nondeterministically guess a certificate c
2. ~~Check if A satisfies the formula F~~
   Check if c proves that $w \in L$
3. ~~Accept F if A satisfies F. Reject otherwise~~
   Accept if c proves that $w \in L$. Reject otherwise

NP and verification

- Why is the nondeterministic 3-SAT algorithm so efficient?
- While 3-SAT is hard to search, it is easy to verify!
  - The certificate that the machine needs to guess (a satisfying assignment) is short
  - After guessing the certificate, it is easy to verify that the certificate is valid
- **Nondeterministic machines are efficient when there is a short, easily verified certificate**

NP and verification

Theorem: A language $L \in \mathrm{NP}$ if and only if it has a polynomial-time verifier

NP and verification

Theorem: A language $L \in \mathrm{NP}$ if and only if it has a polynomial-time verifier

($\Rightarrow$)

- Suppose $L \in \mathrm{NP}$. Then L is recognized by an NTM M that runs in polynomial time
- Construct a verifier V that takes a string w and an *accepting computation history* H as input
- Because M runs in poly-time, the length of computation history is polynomially bounded
- We can verify that H is a computation history for which M accepts w in poly-time

NP and verification

Theorem: A language $L \in \mathrm{NP}$ if and only if it has a polynomial-time verifier

($\Leftarrow$)

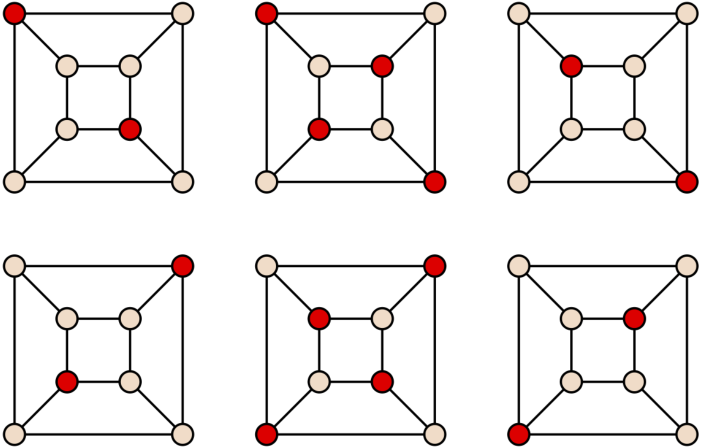
- Suppose L has a poly-time verifier V
- Construct an NTM that takes a string w as input, nondeterministically guesses a certificate c, and passes it to V to check if $w \in L$
- Since V is a poly-time verifier, the certificate has polynomial length and can be guessed in poly-time
- Since V runs in poly-time, it takes poly-time to check if c proves that $w \in L$

The class NP – Recap

- NP is the set of languages that can be decided in nondeterministic polynomial time
- Alternately, it is the set of languages that can be *verified* in (deterministic) polynomial time
- **To show that a language is in NP, it suffices to show that a potential solution to the problem can be checked for validity in polynomial time**

The language IND-SET

- **Def:** Let $G = (V, E)$ be a graph. A **independent set** is a collection of vertices $I \subseteq V$ such that no two vertices are connected



The language IND-SET

- **Def:** Let $G = (V, E)$ be a graph. A **independent set** is a collection of vertices $I \subseteq V$ such that no two vertices are connected
- **Search problem:** Given a graph G, find the largest independent set
- **Decision problem:** Given a graph G and an integer k, determine if G has an independent set set of size k

$$\text{IND-SET} = \{\langle G, k\rangle | G \text{ has a size k independent set}\}$$

IND-SET $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, I \rangle$ as input
2. Check that $|I| \geq k$
3. For every pair of vertices $u, v \in I$, check that u and v are not connected
4. If I is a valid independent set of size k, accept $\langle G, k, I \rangle$; otherwise reject

IND-SET $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, I \rangle$ as input
2. Check that $|I| \geq k$
3. For every pair of vertices $u, v \in I$, check that u and v are not connected
4. If I is a valid independent set of size k, accept $\langle G, k, I \rangle$; otherwise reject

- Certificate size $|I|$ is $O(n)$

IND-SET $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, I \rangle$ as input
2. Check that $|I| \geq k$
3. For every pair of vertices $u, v \in I$, check that u and v are not connected
4. If I is a valid independent set of size k, accept $\langle G, k, I \rangle$; otherwise reject

- Certificate size $|I|$ is $O(n)$
- $O(n^2)$ pairs of vertices to check

IND-SET $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, I \rangle$ as input
2. Check that $|I| \geq k$
3. For every pair of vertices $u, v \in I$, check that u and v are not connected
4. If I is a valid independent set of size k, accept $\langle G, k, I \rangle$; otherwise reject

- Certificate size $|I|$ is $O(n)$
- $O(n^2)$ pairs of vertices to check
- Verifier runs in polynomial time

IND-SET $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess an independent set $I \subseteq V$ of size k
2. Check that none of the vertices in I are connected
3. If I is an independent set of size k, accept G; otherwise reject

IND-SET $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess an independent set $I \subseteq V$ of size k
2. Check that none of the vertices in I are connected
3. If I is an independent set of size k, accept G; otherwise reject

- $O(n)$ to guess an independent set

IND-SET $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess an independent set $I \subseteq V$ of size k
2. Check that none of the vertices in I are connected
3. If I is an independent set of size k, accept G; otherwise reject

- $O(n)$ to guess an independent set
- $O(n^2)$ to check if we guessed the right independent set

IND-SET $\in$ NP

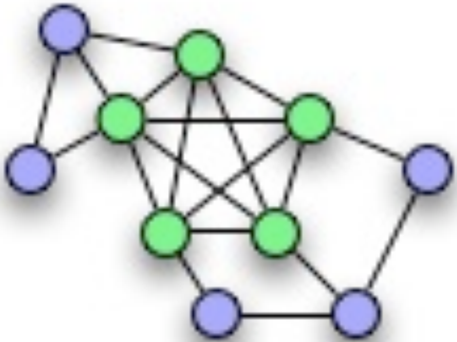
Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess an independent set $I \subseteq V$ of size k
2. Check that none of the vertices in I are connected
3. If I is an independent set of size k, accept G; otherwise reject

- $O(n)$ to guess an independent set
- $O(n^2)$ to check if we guessed the right independent set
- $O(n) + O(n^2) \in \text{NP}$

The language CLIQUE

- **Def:** Let $G = (V, E)$ be a graph. A **clique** is a collection of vertices $C \subseteq V$ every pair of vertices is connected



The language CLIQUE

- **Def:** Let $G = (V, E)$ be a graph. A **clique** is a collection of vertices $C \subseteq V$ every pair of vertices is connected
- **Search problem:** Given a graph G, find the largest clique
- **Decision problem:** Given a graph G and an integer k, determine if G has a clique of size k

$$\mathrm{CLIQUE} = \{\langle G, k\rangle | G \text{ has a size k clique}\}$$

CLIQUE $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, C \rangle$ as input
2. Check that $|C| \geq k$
3. For every pair of vertices $u, v \in C$, check that u and v are connected
4. If C is a valid clique of size k, accept $\langle G, k, C \rangle$; otherwise reject

CLIQUE $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, C \rangle$ as input
2. Check that $|C| \geq k$
3. For every pair of vertices $u, v \in C$, check that u and v are connected
4. If C is a valid clique of size k, accept $\langle G, k, C \rangle$; otherwise reject

- Certificate size $|C|$ is $O(n)$

CLIQUE $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, C \rangle$ as input
2. Check that $|C| \geq k$
3. For every pair of vertices $u, v \in C$, check that u and v are connected
4. If C is a valid clique of size k, accept $\langle G, k, C \rangle$; otherwise reject

- Certificate size $|C|$ is $O(n)$
- $O(n^2)$ pairs of vertices to check

CLIQUE $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes $\langle G, k, C \rangle$ as input
2. Check that $|C| \geq k$
3. For every pair of vertices $u, v \in C$, check that u and v are connected
4. If C is a valid clique of size k, accept $\langle G, k, C \rangle$; otherwise reject

- Certificate size $|C|$ is $O(n)$
- $O(n^2)$ pairs of vertices to check
- Verifier runs in polynomial time

CLIQUE $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a clique $C \subseteq V$ of size k
2. Check that all of the vertices in C are connected
3. If C is a clique of size k, accept G; otherwise reject

CLIQUE $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a clique $C \subseteq V$ of size k
2. Check that all of the vertices in C are connected
3. If C is a clique of size k, accept G; otherwise reject

- $O(n)$ to guess a clique

CLIQUE $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a clique $C \subseteq V$ of size k
2. Check that all of the vertices in C are connected
3. If C is a clique of size k, accept G; otherwise reject

- $O(n)$ to guess a clique
- $O(n^2)$ to check if we guessed the right clique

CLIQUE $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a clique $C \subseteq V$ of size k
2. Check that all of the vertices in C are connected
3. If C is a clique of size k, accept G; otherwise reject

- $O(n)$ to guess a clique
- $O(n^2)$ to check if we guessed the right clique
- $O(n) + O(n^2) \in \mathrm{NP}$

The language SUBSET-SUM

$$\text{SUBSET-SUM} = \left\{ \langle B, x_1, x_2, \ldots x_n \rangle \middle| \begin{array}{c} \text{B is binary} \\ \text{there is a combination of } x_i \text{ (no repeats)} \\ \text{that add up to B} \end{array} \right\}$$

Example: $\langle 31, 7, 4, 9, 5, 20 \rangle$
Solution: $7 + 4 + 20 = 31$✓

Example: $\langle 101, 6, 8, 10 \rangle$
Solution: It is impossible; $6 + 8 + 10 = 24 < 101$

SUBSET-SUM $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes as input $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$
2. For each y_i, check that $y_i \in (x_1, x_2, \ldots x_k)$
3. For each x_i, check that x_i is not used more than once
4. Check that $y_1 + y_2 + \ldots y_k = B$
5. If $y_1 + \ldots y_n = B$ (and it forms a valid subset), accept $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$; otherwise reject.

SUBSET-SUM $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes as input $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$
2. For each y_i, check that $y_i \in (x_1, x_2, \ldots x_k)$
3. For each x_i, check that x_i is not used more than once
4. Check that $y_1 + y_2 + \ldots y_k = B$
5. If $y_1 + \ldots y_n = B$ (and it forms a valid subset), accept $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$; otherwise reject.

- $O(n \cdot k)$ comparisons = poly-time

SUBSET-SUM $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes as input $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$
2. For each y_i, check that $y_i \in (x_1, x_2, \ldots x_k)$
3. For each x_i, check that x_i is not used more than once
4. Check that $y_1 + y_2 + \ldots y_k = B$
5. If $y_1 + \ldots y_n = B$ (and it forms a valid subset), accept $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$; otherwise reject.

- $O(n \cdot k)$ comparisons = poly-time
- $O(n \cdot k)$ comparisons = poly-time

SUBSET-SUM $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes as input $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$
2. For each y_i, check that $y_i \in (x_1, x_2, \ldots x_k)$
3. For each x_i, check that x_i is not used more than once
4. Check that $y_1 + y_2 + \ldots y_k = B$
5. If $y_1 + \ldots y_n = B$ (and it forms a valid subset), accept $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$; otherwise reject.

- $O(n \cdot k)$ comparisons = poly-time
- $O(n \cdot k)$ comparisons = poly-time
- Poly-time to add and compare numbers

SUBSET-SUM $\in$ NP

Approach 1: Construct a poly-time verifier V

1. V takes as input $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$
2. For each y_i, check that $y_i \in (x_1, x_2, \ldots x_k)$
3. For each x_i, check that x_i is not used more than once
4. Check that $y_1 + y_2 + \ldots y_k = B$
5. If $y_1 + \ldots y_n = B$ (and it forms a valid subset), accept $\langle B, x_1, \ldots x_n, y_1 \ldots y_k \rangle$; otherwise reject.

- $O(n \cdot k)$ comparisons = poly-time
- $O(n \cdot k)$ comparisons = poly-time
- Poly-time to add and compare numbers
- Poly-time + poly-time + poly-time $\in$ NP

SUBSET-SUM $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a subset $(y_1, y_2, \ldots y_k) \subseteq (x_1, x_2, \ldots x_n)$
2. Check if $y_1 + \ldots y_k = B$.

SUBSET-SUM $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a subset $(y_1, y_2, \ldots y_k) \subseteq (x_1, x_2, \ldots x_n)$
2. Check if $y_1 + \ldots y_k = B$.

- $O(n)$ to guess a subset

SUBSET-SUM $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a subset $(y_1, y_2, \ldots y_k) \subseteq (x_1, x_2, \ldots x_n)$
2. Check if $y_1 + \ldots y_k = B$.

- $O(n)$ to guess a subset
- poly-time to check if subset sum matches the desired total

SUBSET-SUM $\in$ NP

Approach 2: Construct a machine M that runs in nondeterministic poly-time

1. Nondeterministically guess a subset $(y_1, y_2, \dots y_k) \subseteq (x_1, x_2, \dots x_n)$
2. Check if $y_1 + \dots y_k = B$.

- $O(n)$ to guess a subset
- poly-time to check if subset sum matches the desired total
- $O(n) \cdot$ poly-time $\in \mathrm{NP}$

P vs. NP

Does $\mathrm{P} = \mathrm{NP}$?

- **Can every nondeterministic polynomial time algorithm be converted to a deterministic polynomial time algorithm?**
- Are nondeterministic machines fundamentally faster than deterministic machines?
- Can every efficient verification algorithm be converted to an efficient search algorithm?
- Is searching fundamentally harder than verifying?

Activity: Search vs. Verification