# Introduction to Computational Complexity Theory

Last Updated September 15th, 2025

## 1 Introduction

**Definition 1.1.** The **Computational complexity** of a computational problem refers to the minimum amount of resources (e.g. execution steps or memory) needed to solve an instance of the problem in relation to its size.

1. In this chapter we focus almost entirely on decision problems.
2. One reason for this is that the vast majority of problems that are of interest to both computing practitioners and complexity theorists are either decision or optimization problems.
3. Most optimization problems can be readily converted to decision problems.

**Example 1.2.** An instance of optimization problem `Max Clique` is a simple graph $G$, and the problem is to find the the largest clique in $G$. On the other hand, an instance of decision problem `Clique` is a pair $(G, k)$ and the problem is to decide if $G$ has a clique of size $k$.

Notice that an algorithm for solving `Max Clique` immediately yields an algorithm for solving `Clique` (why?). Furthermore, if there is an algorithm for solving `Clique` in $O(t(n))$ steps, then it can be shown that there is also one for solving `Max Clique` in $O(\log(n)t(n))$ steps. □

Let G have 200 vertices
Determine max clique for G
given that we have an algorithm
that solves the decision problem

| Question | Answer |
|---|---|
| Is $(G, k=100)$ positive? | Yes |
| Does G have a $k=150$ clique? | NO |
| Does G have a $k=125$ clique? | Yes |

(Continue until max clique size has been found)

# 2 Problem Size and Size Parameters

**Definition 2.1.** Given a decision problem $A$ and an instance $x$ of $A$, $|x|$ denotes the **size** of $x$ and equals the number of bits needed to binary-encode $x$. The notation $|x|$ is often useful when speaking abstractly about a generic decision problem, and an instance $x$ of that problem.

**Definition 2.2.** Given a decision problem $A$, a **size parameter** for $A$ is a parameter that may be used to (approximately) represent the size of an instance of $A$. Given an algorithm $\mathcal{A}$ that decides $A$, its size parameters allow one to describe the number of steps (and/or amount of memory) required by $\mathcal{A}$ as a function of the size parameters.

**Example 2.3.** The following are some examples of problems, their size parameters, and examples of how those size parameters are used.

`Clique`
1. Instance: $(G = (V, E), k)$
2. Size parameters: $m = |E|$, $n = |V|$
3. Example: verifying that some $k$ vertices form a clique can be done in $O(n^2)$ steps.

`Subset Sum`
1. Instance: $(S, t)$
2. Size parameters: $n = |S|$, $b$ is the number of bits needed to write $t$ (we assume that $t \geq s$, for all $s \in S$.
3. Example: verifying that a subset of $S$ sums to $t$ can be done in $O(nb)$ steps.

`2SAT`
1. Instance: $\mathcal{C}$
2. Size parameters: $m = |\mathcal{C}|$, $n =$ number of variables of $\mathcal{C}$.
3. Example: verifying that an assignment $\alpha$ satisfies $\mathcal{C}$ can be done in $O(m)$ steps.

`Prime`
1. Instance: $n$
2. Size parameters: $b$ is the number of bits needed to write $n$ in binary. Note that $b = \lfloor \log_2 n \rfloor + 1$.
3. Example: there is an algorithm that can decide `Prime` using $O(b^6)$ steps.

$n = 2^{100}$ $n = 86 \leftarrow$ value

$b = 7$ bits

$(86)_2 = 1010110$

# 3 The Complexity Class P

**Definition 3.1.** A **complexity class** represents a set of decision problems, each of which can be decided by an algorithm that has one or more constraints placed on the resources (usually the number of steps or memory allowed) that it may use when deciding the problem.

**Definition 3.2.** Decision problem $A$ is a member of complexity class P if there is an algorithm that decides $A$ in a polynomial number of steps with respect to the size parameters of $A$.

$O(m+n) \Rightarrow$ linear number of steps.

**Notes:**

1. For example, if $n$ is the size parameter for $A$, then there must be an algorithm $\mathcal{A}$ that decides $A$ and, for an input $x$ of size $n$, $\mathcal{A}$ requires $O(n^k)$ steps before returning 0 or 1, where $k \geq 1$ is an integer.

2. Recall that $f(n) = O(n^k)$ iff there exists a constant $C > 0$ for which $f(n) \leq Cn^k$ when $n$ is sufficiently large. Thus, if we let $T(n)$ denote the number of steps taken by $\mathcal{A}$ to decide instance $x$, where $|x| = n$, then we require $T(n) = O(n^k)$.

3. Complexity class P is considered robust in the sense that its members tend to remain the same from one model of computation to the next (granted, some models of computation are inherently inefficient, and are not appropriate for use in complexity theory).

**Example 3.3.** Here is a description of some important decision problems that are members of `P`, some of which required an algorithmic breakthrough before acquiring membership.

**Distance between Graph Vertices** Given weighted graph $G = (V, E, w)$, vertices $a, b \in V$, and integer $k \geq 0$, is it true that the distance from $a$ to $b$ does not exceed $k$? Dijkstra's algorithm solves this problem in $O(m \log n)$ steps.

**Primality Test** Given natural number $n \geq 2$ is $n$ prime? (see "PRIMES is in P", Annals of Mathematics, Pages 781-793 from Volume 160 (2004), Issue 2 by Manindra Agrawal, Neeraj Kayal, Nitin Saxena). The algorithm requires $O(\log^6 n)$ steps.

**Linear Optimization** Given i) function $f(x) = cx$, for some $1 \times n$-dimensional constant matrix $c$ and $n \times 1$ real-valued matrix $x$, ii) constant $k \in \mathcal{R}$, iii) $m \times n$ constant matrix $A$, and iv) $m \times 1$ constant matrix $b$, is it true that there is an $x$ for which

$$f(x) = cx \leq k,$$

subject to

$$Ax \geq b?$$

Karmarkar's algorithm solves this problem in $O(n^{3.5}L^2 \cdot \log L \cdot \log(\log L))$ steps, where $n$ is the number of problem variables, and $L$ is the number of bits needed to encode an problem instance.

**Maximum Flow** Given directed network $G = (V, E, c, s, t)$ and integer $k \geq 0$, is there a flow from $s$ to $t$ of size at least $k$? The Ford Fulkerson algorithm solves this problem in $O(n^3)$ steps.

**Perfect Matching** Given bipartite graph $G = (U, V, E)$, where $|U| = |V| = n$, does $G$ have a **perfect matching**, i.e. a set of edges $M = \{e_1, \ldots, e_n\} \subseteq E$ such that any two edges $e_i, e_j \in M$ neither share a vertex in $U$, nor share a vertex in $V$? The Ford Fulkerson algorithm solves this problem in $O(n^2 + mn)$ steps.

**2SAT** Given a set of Boolean formulas $\mathcal{C}$, where each formula (called a **clause**) has the form $a \vee b$, where $a$ and $b$ are literals, is there a truth assignment for the variables so that each clause has at least one literal that is assigned `true`? This problem can be solved in $O(m + n)$ steps.

**Bitonic Traveling Salesperson** given $n$ cities $c_1, \ldots, c_n$, where $c_i$ has grid coordinates $(x_i, y_i)$, and a cost matrix $C$, where entry $C_{ij}$ denotes the cost of traveling from city $i$ to city $j$, determine a left-to-right followed by right-to-left Hamilton-cycle tour of all the cities that minimizes the total traveling cost. In other words, the tour starts at the leftmost city, proceeds from left to right visiting a subset of the cities (including the rightmost city), and then concludes from right to left visiting the remaining cities. The problem can be solved in $O(n \log^2 n)$ steps.

**Example 3.4.** An instance of 10-`Clique` is a simple graph $G = (V, E)$ and the problem is to decide if $G$ has a clique of size 10. Show that 10-`Clique` is in `P`. In general, for any integer constant $C > 0$, the `C-Clique` decision problem is in `P`.

**Solution.** Iteratively check every possible subset having 10 vertices.

$\binom{n}{10}$ "n choose 10"

$$\frac{n(n-1)(n-2)\cdots(n-9)}{10!} =$$

$O(n^{10}) \Rightarrow$ 10-Clique $\in P$



$\frac{3\,2\,1}{3!} = \frac{3!}{3!} = 1$ way of making a 3-subset from $\{1,2,3\}$

**Example 3.5.** An instance of the 10-`Path` is a simple graph $G = (V, E)$ and the problem is to decide if $G$ has a simple path of length equal to 10. Show that 10-`Path` is in `P`. In general, for any integer constant $C > 0$, the `C-Path` decision problem is in `P`.

$n = |V|$

**Solution.**

There are

$$\underbrace{n(n-1)(n-2)\cdots(n-10)}_{11 \text{ factors}}$$

possible (simple) paths of length 10.

$\therefore$ checking each path requires $O(n^{11})$ steps.

Figure 1: Solving an NP problem can be like finding a needle in a haystack.

# 4 The Complexity Class NP

**Definition 4.1.** Decision problem $A$ is a member of complexity class NP if there is

1. a set Cert, called the **certificate set**,
2. a decision algorithm V, called the **verifier**, which has the following properties:
   (a) the inputs to V are i) an instance $x$ of $A$ and ii) a certificate $c \in$ Cert
   (b) the output is 1 iff $c$ is a **valid certificate** for $x$, meaning that $c$ proves that $x$ is a positive instance of $A$
   (c) V requires a polynomial number of steps with respect to the size parameters of $A$.

Although, for any given instance $x$ of $A$ and any certificate $c \in$ Cert, the verifier only requires a polynomial number of steps, what makes some NP problems very difficult to solve is that there are usually an exponential number of certificates, and finding a valid one is like finding a "needle-in-a-haystack" because there is no apparent way to avoid having to examine an exponential (in the size parameters of $A$) number of certificates.

**Example 4.2.** We show that `Clique` $\in$ `NP`. Let $(G = (V, E), k)$ be a problem instance for `Clique`.

**Step 1: define a certificate**. Certificate $C$ is a subset of $V$ where $|C| = k$.

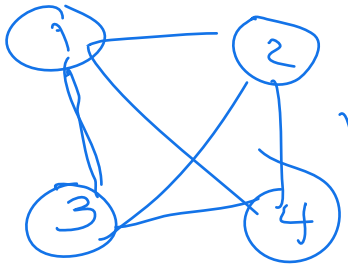
**Step 2: provide a semi-formal verifier algorithm**.

For each $u \in C$,

  For each $v \in C$ with $u \neq v$,

    If $(u, v) \notin E$, then return 0.

Return 1.




$n = 4$

$m = 6$

$\binom{4}{2} = \frac{4 \cdot 3}{2} = 6$

**Step 3: size parameters for `Clique`**. $m = |E|$ and $n = |V|$.

**Step 4: provide the verifier's running time with an explanation**.

The nested `for`-loops require at most $k^2 = \mathrm{O}(n^2)$ query to determine if a pair of vertices $(u, v)$. Each query can be answered using a hash table that stores the graph edges. Building such a table takes $\Theta(m)$ steps. Thus, algorithm's total number of steps is $\mathrm{O}(m + n^2) = \mathrm{O}(n^2)$ steps, which is quadratic in the size of $(G, k)$.

$m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$

**Example 4.3.** By repeating the steps of Example 4.2, prove that `Subset Sum` $\in$ `NP`. Let $(S, t)$ be an instance of `Subset Sum`.

**Step 1: define a certificate.**

$A \subseteq S$ is a subset of $S$

**Step 2: provide a semi-formal verifier algorithm.**

```
Sum = 0.
For each a ∈ A,
    sum += a.          → O(b) steps.
Return (sum == t). //A is valid iff Σ_{a∈A} a = t
```

**Step 3: provide size parameters for `Subset Sum`.**

$n = |S|$ $b$ = # of bits needed for $t$

**Step 4: provide the verifier's running time with an explanation.**

$n$ iterations of for-loop. each iteration requires $O(b)$ steps.

$\therefore$ Verifier requires $O(nb)$.

$\Rightarrow$ quadratic number of steps.

**Example 4.4.** Recall from Exercise 10 of the Computational Problems lecture that the `3-Dimensional Matching (3DM)` decision problem takes as input three sets $A$, $B$, and $C$, each having size $n$, along with a set $S$ of triples of the form $(a, b, c)$ where $a \in A$, $b \in B$, and $c \in C$. We assume that $|S| = m \geq n$. The problem is to decide if there exists a subset $T \subseteq S$ of $n$ triples for which each member from $A \cup B \cup C$ belongs to exactly one of the triples.

Show that $(A, B, C, S)$ is a positive instance of `3DM`, where $A = \{a, b, c\}$, $B = \{1, 2, 3\}$, $C = \{x, y, z\}$, and

$$S = \{(a, 1, x), (a, 2, z), (a, 3, z), (b, 1, x), (b, 2, x), (b, 3, z), (c, 1, x), (c, 2, z), (c, 3, y)\}.$$

**Solution.**

**Example 4.5.** By repeating the steps of Example 4.2, prove that 3DM $\in$ NP. Let $(A, B, C, S)$ be an instance of 3DM.

$|A| = |B| = |C| = n$

$|S| = m$

**Step 1: define a certificate**.

T is an $n$-subset of S.

**Step 2: provide a semi-formal verifier algorithm**.

Let H be an empty hash table

For each $t \in T$,

If ($t[0] \in H \vee t[1] \in H \vee t[2] \in H$),

return 0.

insert(H, t[0]), insert(H, t[1]), insert(H, t[2])

Return 1.

**Step 3: provide size parameters for** 3DM.

$m = |S|$, $n = |A| = |B| = |C|$

**Step 4: provide the verifier's running time with an explanation**.

n iterations of for-loop

Each iteration Requires $O(1)$ steps.

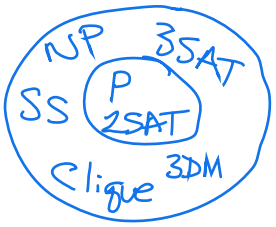
$\therefore$ $O(n)$ number of steps.

**Example 4.6.** By repeating the steps of Example 4.2, prove that 2SAT $\in$ NP. Let $\mathcal{C}$ be an instance of 2SAT.

$C = (x_1, \bar{x}_2)$ $C[0] = x_1$ $C[1] = \bar{x}_2$

**Step 1: define a certificate.** (Boolean)

$\propto$ is an assignment over the variables of $C$




**Step 2: provide a semi-formal verifier algorithm.**

$P \subseteq NP$

For each $c \in C$
If $!\propto[c[0]] \wedge !\propto[c[1]]$, then Return 0

Return 1.

**Step 3: provide size parameters for 2SAT.**

$m = |C|$

$n =$ # of variables

**Step 4: provide the verifier's running time with an explanation.**

There are $m$ iterations of the for-loop, and $O(1)$ array accesses per iteration. $\therefore O(m) \Rightarrow$ linear running time

$\therefore$ 2SAT $\in$ NP

**Theorem 4.7.** P $\subseteq$ NP.

**Proof.** Let $L \in$ P be a decision problem that can be decided in polynomial time. Then there is a polynomial-time computable predicate function $D(x)$ that decides $L$. Moreover, $D(x)$ serves as a verifier for problem $L$ and without need for a certificate. Note that we *could* define a certificate for an instance $x$ of $L$, but $D$ may ignore it because $D$ is able to decide $x$ in a polynomial number of steps using only knowledge of $x$ itself. □

### The Infamous P =? NP Problem

Do there exist decision problems that are in NP but not in P? This would imply that some problems have solutions that can be verified in polynomial time, but not solved in polynomial time. Moreover, NP problems such as Clique, Subset Sum, and 3DM are candidates since, at this writing, polynomial-time algorithms for these problems have yet to be established. But at the same time a proof that such algorithms do not exist has yet to be found.

The P=?NP problem is considered one of the most challenging and important in all of computer science and mathematics. The Clay Mathematics Institute is awarding a prize of $1 million dollars to anyone who can resolve this problem.

In the next lecture we provide strong evidence that suggests that NP is a larger class of problems than P because there are literally thousands of known problems in computer science that are in NP and have no known efficient algorithms. In the next lecture we show that, if one of these problems in in P, then *all* the problems must be in P. How can that be? I would like to tell you but right now I need to ....

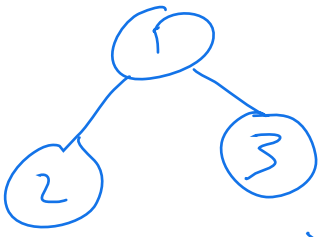
## The Complexity Class co-NP

$\overline{\text{Even}} = \text{Odd}$

Given a decision problem $L$, the **complement** of $L$, denoted $\overline{L}$, is that decision problem for which a positive (respectively, negative) instance $x$ of $\overline{L}$ is a negative (respectively, positive) instance of $L$.

**Example 4.8.** If $L$ is the problem of deciding if a positive integer is prime, then define its complement $\overline{L}$.

**Solution.** $\overline{\text{Prime}} = \text{Composite}$

G: 


$k=3$

$(G, 3)$ is a negative instance of Clique

Also, $(G, 3)$ is a negative instance of IS

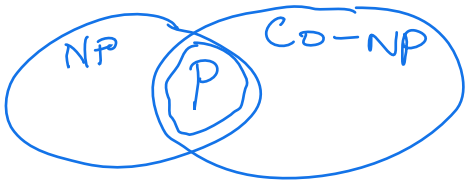
**Example 4.9.** Define the complement of the 2SAT decision problem.

**Solution.** $\overline{\text{2SAT}} = \text{UN2SAT}$

instance is a ~~set~~ of clauses $C$

Question: are $C$'s clauses unsatisfiable? Yes or No?

## A Logical definition of Co-NP




It is left as an exercise to show that if $L \in \texttt{P}$ then $\overline{L} \in \texttt{P}$. On the other hand, it is believed that NP and co-NP are different complexity classes because each has its own distinct predicate-logic definition.

For example, consider a problem $L \in \texttt{NP}$ which has certificate set $C$ and verifier function $v(x, c)$. Then we may logically write that $x$ is a positive instance of $L$ iff

$$\exists_{c \in C} v(x, c)$$

evaluates to 1.

Now consider its complement $\overline{L} \in \texttt{co-NP}$. Then we may logically write that $x$ is a positive instance of $\overline{L}$ iff it is a negative instance of $L$ iff

$$\neg \exists_{c \in C} v(x, c) \Leftrightarrow$$

$$\forall_{c \in C} (\neg v(x, c)) \Leftrightarrow$$

$$\forall_{c \in C} v'(x, c).$$

evaluates to 1, where $v'(x, c) = \neg v(x, c)$. In other words, co-NP problems are logically defined with a universal predicate-logic statement, while an NP problem is logically defined with an existential predicate-logic statement. Thus, logically speaking, these classes seem different in that their problems are complementary to one another.

**Example 4.10.** For each of the following problem definitions, provide the complexity class (`P`, `NP`, or `co-NP`) that best fits the problem.

$x_1 \vee \overline{x_1}$ is a tautology

`Tautology` Given a Boolean formula $F(x_1, \ldots, x_n)$ does $F$ evaluate to 1 on all possible $2^n$ binary input vectors? co-NP

`Reachability` Given a simple graph $G = (V, E)$ and two vertices $a, b \in V$, does there exist a path in $G$ starting at $a$ and ending at $b$? P

`Dominating Set` Given a simple graph $G = (V, E)$ and an integer $k \geq 0$, does there exist a set $D$ of $k$ vertices for which every vertex in $V - D$ is adjacent to some vertex in $D$? NP

`Bounded Cliques` Given a simple graph $G = (V, E)$ and an integer $k \geq 0$, it true that $G$ has no $k$-cliques? co-NP

Bounded Cliques = $\overline{\text{Clique}}$

**Theorem 4.11.** $\texttt{P} \subseteq \texttt{NP} \cap \texttt{co-NP}$.

**Proof.** Let $L \in \texttt{P}$ be given. Then by Theorem 4.7, $L \in \texttt{NP}$. But notice also that $\overline{L} \in \texttt{P}$ as well (why?) and hence $\overline{L} \in \texttt{NP}$. But then $\overline{\overline{L}} = L \in \texttt{co-NP}$. Therefore,

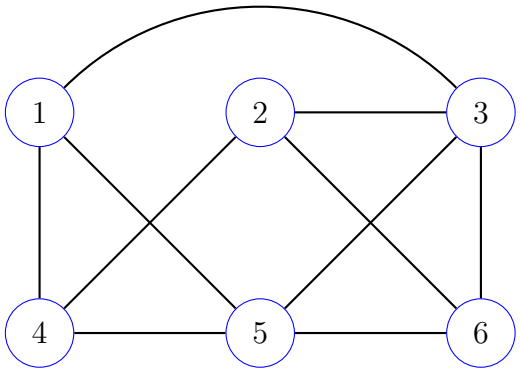
$$L \in \texttt{NP} \cap \texttt{co-NP}. \quad \square$$

# Complexity Core Exercises

1. Let `Triangle` be the problem of deciding if a simple graph $G = (V, E)$ has a 3-clique. Provide a semi-formal algorithm that establishes that `Triangle` is in `P`. Explain why your algorithm requires at most a polynomial number of steps.

2. An instance of `4-Subset Sum` is a pair $(S, t)$, where $S$ is a set of positive integers and $t > 0$ is a positive integer, and the problem is to decide if there are four distinct members of $S$, $x, y, z, w$, for which $x + y + z + w = t$. Provide a semi-formal algorithm that establishes that `4-Subset Sum` is in `P`. Explain why your algorithm requires at most a polynomial number of steps.

3. An instance of the `3-Coloring` decision problem is a simple graph $G = (V, E)$, and the problem is to decide if the vertices of $G$ can be colored using three colors (red, blue, and green) in such a way that no two adjacent vertices have the same color. In other words, does there exist a function color : $V \to \{\text{red}, \text{blue}, \text{green}\}$, for which, for any edge $(a, b) \in E$, color$(a) \neq$ color$(b)$? For example, verify that following graph




admits the 3-coloring

| Vertex | Color |
|---|---|
| 1 | red |
| 2 | green |
| 3 | blue |
| 4 | blue |
| 5 | green |
| 6 | red |

Prove that `3-Coloring` is in `NP` by completing the following steps.

   a. Define a certificate for `3-Coloring`.

   b. Provide a semi-formal algorithm for the `3-Coloring` verifier.

   c. Provide size parameters for `3-Coloring`.

   d. Provide the verifier's running time and defend your answer.

4. Recall that an instance of `Set Partition` is a set $S$ of nonnegative integers and the problem is to decide if there are are subsets $A, B \subseteq S$ for which i) $A \cap B = \emptyset$, ii) $A \cup B = S$, and iii)

$$\sum_{a \in A} a = \sum_{b \in B} b.$$

Prove that `Set Partition` is in `NP` by completing the following steps.

a. Define a certificate for `Set Partition`.

b. Provide a semi-formal algorithm for the `Set Partition` verifier.

c. Provide size parameters for `Set Partition`.

d. Provide the verifier's running time and defend your answer.

5. Recall the `DHP` decision problem, where an instance consists of a directed graph $G = (V, E)$ and vertices $a, b \in V$ and the problem is to decide if $G$ has a directed Hamilton path (DHP), Prove that `DHP` is in `NP` by completing the following steps.

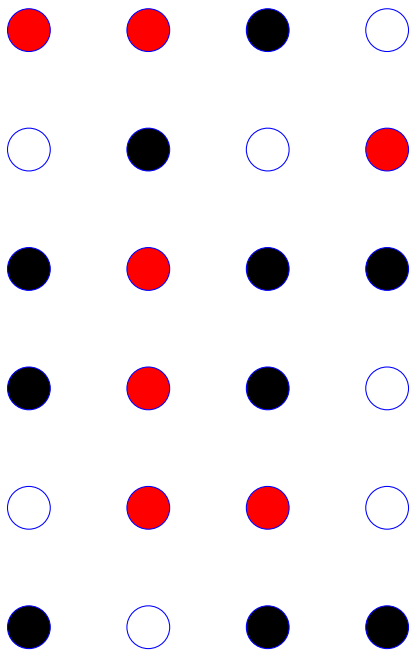
   a. Define a certificate for `DHP`.

   b. Provide a semi-formal algorithm for the `DHP` verifier.

   c. Provide size parameters for `DHP`.

   d. Provide the verifier's running time and defend your answer.

6. Consider the `Solitaire` decision problem, where an instance consists of an $m \times n$ grid, and each square in the grid is either empty, has a single red stone, or has a single black stone. The problem is to decide if, for each column, there is a subset of the stones that can be removed so that i) every column has zero or more stones of the same color, and ii) every row has at least one stone placed in it. Show that the following is a positive instance of `Solitaire`.



7. Prove that the `Solitaire` decision problem defined in the previous exercise is in `NP` by completing the following steps.

   a. Define a certificate for `Solitaire`.

   b. Provide a semi-formal algorithm for the `Solitaire` verifier.

c. Provide size parameters for `Solitaire`.

d. Provide the verifier's running time and defend your answer.

8. An instance of `Set Cover` is a triple $(\mathcal{S}, m, k)$, where $\mathcal{S} = \{S_1, \ldots, S_n\}$ is a collection of $n$ subsets, where $S_i \subseteq \{1, \ldots, m\}$, for each $i = 1, \ldots, n$, and a nonnegative integer $k$. The problem is to decide if there are $k$ subsets $S_{i_1}, \ldots, S_{i_k}$ for which

$$S_{i_1} \cup \cdots \cup S_{i_k} = \{1, \ldots, m\}.$$

Verify that $(\mathcal{S}, m, k)$ is a positive instance of `Set Cover`, where $m = 9$, $k = 4$, and

$$\mathcal{S} = \{\{1,3,5\}, \{3,7,9\}, \{2,4,5\}, \{2,6,7\}, \{6,7,9\}, \{2,7,9\}, \{1,3,7\}, \{4,5,8\}\}.$$

9. Each of the following graph decision problems described below takes as input a simple graph $G = (V, E)$ and a nonnegative integer $k \geq 0$. Classify each one as either being in `P`, `NP`, or co-`NP`.

a. Decide if it's true that $G$ has not vertex cover of size $k$.

b. Decide if $G$ has at least $k$ connected components. Note: two vertices are in the same component iff they are both reachable from each other.

c. Decide if the size of every independent set of $G$ is less than or equal to $k$.

d. Decide if $G$ has a vertex cover of size $k$.

10. Classify each of the following problems as being in `P`, `NP`, or co-`NP`.

a. An instance of the `Vertex Cover` decision problem is a pair $(G, k)$, where $G = (V, E)$ is a simple graph, $k \geq 0$ is an integer, and the problem is to decide if $G$ has a **vertex cover** of size $k$, i.e. a set $C \subseteq V$ for which every edge $e \in E$ is incident with at least one vertex in $C$.

b. An instance of `Substring` is a pair $(s_1, s_2)$ of binary strings and the problem is to decide if $s_1$ occurs as a substring of $s_2$. For example $(10101, 001010111)$ is a positive instance of `Substring` since 10101 is a substring of 001010111.

c. An instance of `Bounded Clique` is a graph $G = (V, E)$ and an integer $k \geq 0$ and the problem is to decide if the maximum cliqeu in $G$ is of a size that does not exceed $k$.

d. Given positive integers $a, b, c > 0$, determine if there are positive integers $x$ and $y$ for which $ax^2 + by = c$.

# Solutions to Core Exercises

1. We have the following algorithm.

   **Name:** `has_triangle`

   **Input:** simple graph $G = (V, E)$.

   **Output:** `true` iff $G$ has a triangle.

   Add each edge $e \in E$ to a lookup table.

   For each $u \in V$,

   For each $v \in V$ with $v \neq u$,

   For each $w \in V$ with $w \neq u$ and $w \neq v$,

   If $(u, v) \in E$, and $(u, w) \in E$, and $(v, w) \in E$, Return 1.

   Return 0.

   Each of the three nested loops makes at most $n = |V|$ iterations, for a total of $O(n^3)$ iterations. Therefore, `Triangle` is in `P`.

2. On inputs $S$ and $t$, Consider an algorithm that iterates through each 4-subset $\{x, y, z, w\} \in S$ and checks if $x + y + z + w = t$. Since there are

$$\binom{n}{4} = \frac{n(n-1)(n-2)(n-3)}{4!} = \Theta(n^4),$$

   4-subsets and each $x + y + z + w$ sum requires $O(3b) = O(b)$ steps (where $b$ is a bound on the number of bits used by each member of $S$ and $t$), we see that the algorithm requires $O(bn^4)$ steps which is a (fifth-degree) polynomial in the size parameters of `4-Subset Sum`.

3. The following establishes `3-Coloring` is in `NP`.

   a. Certificate $C$ is a vector of length $n$, where the $i$ th vector component, $i \geq 1$, is one of `red`, `blue`, `green`.

   b. The following is the verifier algorithm.

      **Inputs:** i) simple graph $G = (V, E)$ ii) certificate $C$ which is an $n$-dimensional color vector, and determines the color of the $i$ th vertex.

      **Output:** `true` iff $C$ does not color two adjacent vertices with the same color.

      For each $e = (u_i, v_j) \in E$,

      If $C_i = C_j$, Return 0.

      Return 1.

   c. Size parameters: $m = |E|$, $n = |V|$.

   d. Algorithm analysis: the algorithm requires $O(m)$ steps since it iterates once through the set of edges, and accessing a vertex's color from vector $C$ can be done in constant time. Therefore, `3-Coloring` is in `NP`.

4. The following establishes `SP` is in `NP`. Assume $S$ is an instance of `SP`.

a. Certificate $A$ is a subset of $S$.

b. The following is the verifier algorithm.

**Inputs:** i) set $S$ of nonnegative integers, ii) certificate $A \subseteq S$.

**Output:** `true` iff the members in $A$ sum to the members not in $A$ (i.e. in $B = S - A$).

Initialize: $B = S - A$

Return $(\sum_{a \in A} a = \sum_{b \in B} b)$.

c. Size parameters: $n = |S|$, $m$ is a bound on the maximum number of bits required by any number in $S$.

d. Algorithm analysis: the algorithm requires $O(mn)$ steps since it requires making at most $n$ additions with numbers that are at most $m$-bits each. Therefore, `SP` is in `NP`.

5. The following establishes `DHP` is in `NP`. Assume $(G = (V, E), a, b)$ is an instance of `HP`, where $a \in V$ is the start vertex and $b \in V$ is the end vertex.

a. For simplicity, assume $V = \{1, \ldots, n\}$, $a = 1$, and $b = n$. Certificate $P$ is a permutation of the numbers $1, \ldots, n$.

b. The following is the verifier algorithm.

**Inputs:** i) simple graph $G = (V, E)$, ii) certificate $P$, a permutation of the numbers $1, \ldots, n = |V|$.

**Output:** `true` iff $P$ forms a valid Hamilton path in $G$.

If $P(1) \neq 1$ or $P(n) \neq n$, Return 0.

Store the edges of $G$ in a lookup table.

For $i = 1, \ldots, n - 1$

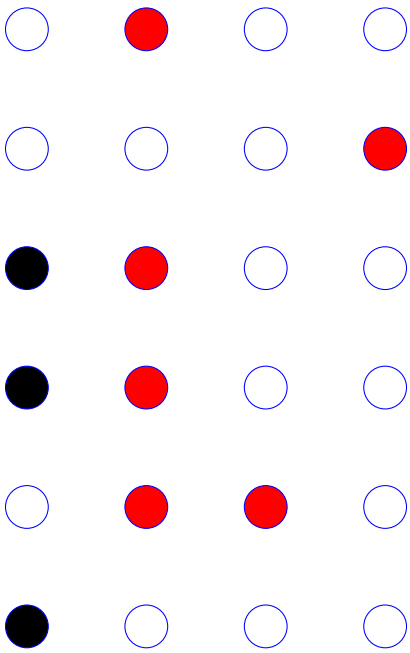
    If $(P(i), P(i + 1) \notin E$, Return 0.

Return 1.

c. Size parameters: $n = |V|$, $m = |E|$.

d. The algorithm requires $O(m + n)$ steps since it requires $O(m)$ steps to build the lookup table and then $O(n)$ to make sure each pair $(P(i), P(i + 1))$ is an edge of $G$. Therefore, `DHP` is in `NP`.

6. Remove all the red stones from column 1, and all the black stones from columns 2-4. Notice that every row has a stone and every column has stones of the same color.



7. The following establishes `Solitaire` is in `NP`. Assume $m \times n$ matrix $M$ is an instance of `Solitaire`, where the entries for $M$ are either 0 (empty), 1 (black), or -1 (red).

   a. Certificate $R$ is an $n$-dimensional vector $(R_1, \ldots, R_n)$, where $R_j \subseteq \{1, \ldots, m\}$ indicates those row values where a stone in column $j$ is to be removed.

   b. The following is the verifier algorithm.

      **Inputs:** i) $\{-1, 0, 1\}$-matrix $M$, ii) certificate vector $R = (R_1, \ldots, R_n)$ of subsets of $\{1, \ldots, m\}$.

      **Output:** `true` iff $R$ is a legal and winning prescription for which stones are to be removed in each column.

      //Check for columns that have stones of different colors.

      For each $j = 1, \ldots, n$,

      For each $i_1 = 1, \ldots, m$ for which $i_1 \notin R_j$,

      For each $i_2 = 1, \ldots, m$ for which $i_2 \notin R_j$,

      If $M[i_1, j]M[i_2, j] = -1$, Return 0. //A red and black stone each remain in column $j$.

      //Check for rows having no stones.

      For each $i = 1, \ldots, m$,

      If $\forall j(M[i, j] = 0 \vee i \in R_j)$, Return 0. //Row $i$ has no stones.

      Return 1.

   c. Size parameters: $m$: number of rows of $M$. $n$: number of columns of $M$.

   d. Algorithm analysis: We assume that all membership queries to each $R_j$ set, $j = 1, \ldots, m$, requires O(1) steps. This can be accomplished if we represent each $R_j$ as an array of size $m$. Checking if any columns have different colored stones after the removals have been made requires at most $O(m^2n)$ steps, while checking if any row is empty requires $O(mn)$ steps for a total of $O(m^2n)$ steps. Therefore, `Solitaire` is in `NP`.

8. The sets

$$\{\{1,3,5\},\{2,6,7\},\{2,7,9\},\{4,5,8\}\}$$

satisfy

$$\{1,3,5\} \cup \{2,6,7\} \cup \{2,7,9\} \cup \{4,5,8\} = \{1,2,\ldots,9\}.$$

9. a. `co-NP`
   b. `P`
   c. `co-NP`
   d. `NP`

10. a. `NP`
    b. `P`
    c. `co-NP`
    d. `NP`