P, NP, NP-complete problems

(references:
- Kleinberg & Tardos, *Algorithm Design*,
- Levitin, *The Design and Analysis of Algorithms*,
- Cormen, et al, *Introduction to Algorithms*)

From Table 2.1 Kleinberg

<table>
<thead>
<tr>
<th>n</th>
<th>n log n</th>
<th>n²</th>
<th>n³</th>
<th>2ⁿ</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=10</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>4 s</td>
</tr>
<tr>
<td>n=50</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>10¹⁰⁻⁹</td>
<td>[v long]</td>
</tr>
<tr>
<td>n=10⁰</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>1 s</td>
<td>10⁴⁻¹</td>
<td>[v long]</td>
</tr>
<tr>
<td>n=10⁶</td>
<td>1 s</td>
<td>20 s</td>
<td>12 d</td>
<td>10⁴⁻⁵</td>
<td>[v long]</td>
</tr>
</tbody>
</table>

tractable, intractable

An algorithm solves a problem in polynomial time if its worst-case time efficiency belongs to O(p(n)) where p(n) is a polynomial of the problem’s input size n. Problems that can be solved in polynomial time are called **tractable**, problems that cannot be solved in polynomial time are called **intractable**.

**Informally**, the problems that can be solved in polynomial time are in class P.

**Formally**, **Class P** is a class of decision problems that can be solved in polynomial time by deterministic algorithms (what is deterministic algorithm?).

A decision problem is a problem that has a yes/no answer. There are two parts to a decision problem:

1. An instance description, i.e., a description of the input problem.
2. The question stated as a yes-or-no question.

The way the problem is stated is fundamental to classifying it into class P or not.

Example: Finding cliques. A clique is a subset of vertexes in a graph which is complete (i.e., there is an edge between every pair of vertexes.)

A. Instance, G = (V, E)
   Question: does G contain a clique of k vertices?

B. Instance: G = (V, E) and k
   Question: does G contain a clique of k vertices?

The algorithm is O(k² nᵏ).
In A, k is a variable, hence O(k² nᵏ) in not P
In B, k is fixed, hence O(k² nᵏ) is in P.

Note also the **optimization problem**, Given G=(V, E), what is the largest clique?
Why is P restricted to decision problems solved in polynomial time?
Many important problems naturally expressed can be reduced to a series of decision problems. For example:

*Optimization problem*: what is the minimum number of colors needed to color the vertices of a graph so that no two adjacent vertices are given the same color.

*Decision problem*: does there exist a coloring of the graph’s vertices with no more than m colors for m = 1, 2, … The first value of m for which the decision problem has a solution solves the optimization version of the graph coloring problem.

Can every decision problem be solved in polynomial time?
No. Indeed, some decision problems are undecidable, i.e., they cannot be solved by any algorithm.

**Most famous example of an undecidable decision problem: the halting problem.**
Given a computer program and an input to the problem, will the program halt on that input or will it continue working indefinitely on it?

There are several important decision problems that we don’t have a polynomial algorithm for AND we can’t prove that a polynomial solution exists or doesn’t exist. Informally, these are called class NP.

Problems that we don’t have polynomial solution for and can’t prove the existence or non-existence of a polynomial algorithm:

1. **Hamiltonian circuit**: determine whether a given graph has a Hamiltonian circuit (A Hamiltonian circuit is a path that starts and ends at the same vertex and passes through all remaining vertexes exactly once).

2. **Travelling salesman**: find the shortest tour through n cities with known positive integer weights on edges (i.e., find the shortest Hamiltonian circuit in a completed weighted graph).

3. **Partition problem**: given n positive integers, determine whether it is possible to partition them into two disjoint subsets with the same sum.

4. **Bin packing**: given n items whose sizes are positive rational numbers not larger than 1, put them into the smallest number of bins of size 1.

5. **Graph coloring**: for a given graph, find its chromatic number (the smallest number of colors that need to be assigned to the graph’s vertices so that no two adjacent vertices are assigned the same color).

6. **Integer linear programming**: find the optimum (max or min) value of a linear function of several integer-valued variables subject to a finite set of constraints in the form of linear equalities and/or inequalities.
Those problems above that are not already expressed as decision problems can be re-cast as decision problems.

A common characteristic of these problems is that a solution can be checked in polynomial time, while finding a solution is not polynomial.

Example: Given set {-2, -3, 5, 15, 14, 7, -10}.
   Check: does subset {-2, -3, 15, -10} sum to 0? O(n)
   Solve: Find a subset that sums to 0.

Informally, class NP solutions cannot be found in polynomial time, while a class NP solution can be verified in polynomial time.

A **nondeterministic algorithm** is a two-stage procedure that takes an instance I of a decision problem and does the following:
1. nondeterministic (“guessing”) stage: An arbitrary string S is generated (i.e., a “candidate solution”).
2. deterministic (“verification”) stage: take I and S as input and output ‘yes’ if S is a solution to I.

A nondeterministic algorithm must be capable of guessing a correct solution at least once and is capable of verifying its validity.

A nondeterministic algorithm is **nondeterministic polynomial** if its verification stage is polynomial.

Formally, **Class NP** is the class of decision problems that can be solved by nondeterministic polynomial algorithms.

**P versus NP**

Most decision problems are in NP. This includes all problems in P. So, \( P \subseteq NP \).

Why is P contained in NP? Use a nondeterministic algorithm for the problem L in P. First, (1) ignore the proposed candidate solution to L found in the first step of the nondeterministic algorithm, then (2) solve the problem L using the P algorithm -- that equates to the verification step in the nondeterministic algorithm.

**Most important open question in theoretical computer science**: does P = NP?

If P = NP then we’ll know that many extremely important NP problems can be solved in polynomial time, even though we don’t yet know the P algorithm. Actually, we know more than that, but to see what else we’ll know, we have to consider NP-complete problems.
Informally, **NP-complete** characterizes a subset of NP problems that can be “reduced” in polynomial time to any other element of the NP-complete set. So all NP-complete problems are equally difficulty. This does not say that an NP-complete problem is solvable in polynomial time. It says that if *any* NP-complete problem can be solved in polynomial time, then any other NP-complete problem can also be solved in polynomial time by a correctly chosen polynomial time reduction.

A decision problem D1 is **polynomially reducible** to a decision problem D2 if there exists a function t that transforms instances of D1 to instances of D2 such that:
1. t maps all “yes” instances of D1 to “yes” instances of D2; all “no” instances of D1 are mapped to “no” instances of D2;
2. t is computable by a polynomial time algorithm.

A decision problem D is **NP-complete** if
1. it belongs to class NP
2. every problem in NP-complete is polynomially reducible to D.

If P = NP, then all NP problems are NP-complete problems, and can be reduced in polynomial time to a problem that can be solved in polynomial time. If P ≠ NP then there must exist NP problems that are not in P and are not in NP-complete.
Recap and survey:

By definition $P \subseteq NP$

solution can be found in $P$  

solution can be checked in $P$

2002 poll: does $P = NP$?

61% no

9% yes

22% unsure

8% impossible to prove or disprove in current formulation
Example:

\[ G = (V, E) \]

A subset of nodes, \( S \subseteq V \) is \textit{independent} if no two nodes in \( S \) are joined by an edge.

- Find largest set of nodes to form independent set (optimization problem)
- Does \( G \) contain an independent set of size \( k \) or greater? (decision problem)

Note: (1) suppose we have an algorithm for the optimization question, then we automatically have an algorithm for the decision question.

(2) Suppose we have a solution to the decision question. Run it \( n \) times (\( n = |V| \)), to answer the optimization question. The largest \( k \) found is the answer to the optimization question.

Here is another hard problem
\[ G = (V, E) \]

A subset of nodes \( S \subseteq V \) is a vertex cover if every edge \( e \subseteq E \) has at least one end in \( S \).

- Given \( G \) and \( k \), does \( G \) contain a vertex cover of size at most \( k \)? (decision problem)
- What is the optimization problem?
We don’t know how to solve either independent set or vertex cover in polynomial time.

But what can be said about their relative difficulty?

We can “reduce” one problem to the other through the use of this theorem:

**Theorem:** $G = (V, E)$ is a graph. 
$S$ is an independent set iff $V - S$ is a vertex cover. 
The proof uses definition of independent set, vertex cover and edge.

$\Rightarrow$ independent set $\leq_p$ vertex cover because we can reduce between the two problems in either direction. 

If we have a black box to find vertex cover, then solve the independent set, size $k$, by asking black box to solve vertex covers of size at most $n-k$. 