NP-COMPLETENESS: a brief informal introduction
NP-COMPLETENESS: a brief informal introduction

We've seen algorithms with several time complexities:

- $O(\log n)$
- $O(n)$
- $O(n\log n)$
- $O(n^2)$
- $O(n^c)$
- $O(2^n)$
- $O(n!)$

**Focus on Decision Problems**

- **P**
  - $O(n^c)$
  - Haven't found $O(n^c)$ yet, but we can verify solutions in $O(n^c)$

- **NP**
  - Not polynomial $w(n^c)$
NP: non-deterministic polynomial
(NOT "non-polynomial")
NP: non-deterministic polynomial (NOT "non-polynomial")

The $1,000,000$ question...

$P \text{ v. NP} : = \text{ or } \neq ?$
NP: non-deterministic polynomial (NOT "non-polynomial")

The $1,000,000$ question...

$P \ p v \ NP : = \ or \ \neq ?$

Is it ever "much" harder to solve a problem than it is to verify a solution, if the verification takes poly-time?
NP: non-deterministic polynomial (NOT "non-polynomial")

The $1,000,000$ question...

$P \text{ v. } NP : = \text{ or } \neq ?$

Is it ever "much" harder to solve a problem than it is to verify a solution, if the verification takes poly-time?

T(verify) = o(n^c \cdot T(solve))
T(solve) = ω(n^c \cdot T(verify))

not within a polynomial factor:
There are thousands of problems for which no known polynomial-time solution is known, yet we can verify proposed solutions in poly-time.
There are thousands of problems for which no known polynomial-time solution is known, yet we can verify proposed solutions in poly-time.

e.g. Hamiltonian cycle

given a graph, find a cycle that visits each vertex exactly once.

\[ \Rightarrow \text{decide if one exists} \]
There are thousands of problems for which no known polynomial-time solution is known, yet we can verify proposed solutions in poly-time.

e.g. Hamiltonian cycle
given a graph, find a cycle that visits each vertex exactly once.

\[ \Rightarrow \text{decide if one exists} \]
"is there a set of $k$ independent vertices?"

(independent: no neighbors)
DECISION PROBLEM
"is there a set of $k$ independent vertices?"

OPTIMIZATION PROBLEM
"find the largest independent set"

(independent: no neighbors)
DETECTION PROBLEM

"is there a set of \( k \) independent vertices?"

OPTIMIZATION PROBLEM

"find the largest independent set"

independent: no neighbors

binary search on \( k: 0 \ldots |V| \)

Often, optimization problems are not polynomially harder than decision.
NP-COMPLETE PROBLEMS

1) in NP, & not known to be in P

(decision problems with solutions that can be verified in poly-time, but for which no poly-time algo is known)
NP-COMPLETE PROBLEMS

1) in NP, & not known to be in P
   (decision problems with solutions
   that can be verified in poly-time, but
   for which no poly-time algo is known)

2) if you ever find a polynomial-time solution for any NPC problem,
   this implies the same for every problem in NP.  $\rightarrow P=NP$
NP-COMPLETE PROBLEMS

1) in NP, & not known to be in P
   (decision problems with solutions that can be verified in poly-time, but for which no poly-time algo is known)

2) if you ever find a polynomial-time solution for any NPC problem, this implies the same for every problem in NP. → P = NP

⇒ if you ever prove that an NPC problem has no poly-time algo, then no NPC problem does → P ≠ NP
Are there other problems in NP but not in P or NPC?
Are there other problems in \( \text{NP} \) but not in \( \text{P} \) or \( \text{NPC} \) ?
- if \( \text{P} = \text{NP} \) then N/A.
Are there other problems in NP but not in \( P \) or NPC?

- If \( P = NP \) then N/A.
- If \( P \neq NP \) then yes. [Theorem]

Few "natural" problems (almost everything in NP is P or NPC)
Are there other problems in NP but not in P or NPC?

- if $P=NP$ then N/A.
- if $P \neq NP$ then yes. [Theorem]

Few "natural" problems (almost everything in NP is P or NPC)

If we solved such a problem in poly-time, it would just move into P without dragging everything else along.
NP-hard problems

\[\Rightarrow \text{as hard as any NP problem.}\]
NP-hard problems

- as hard as any NP problem.
- NPC problems are NP-hard.
NP-hard problems

- as hard as any NP problem.
- NPC problems are NP-hard.
- NP-hard need not be NPC
  - might not be decision problems.

independent set

decision ($\leq k$?)

NPC

optimization (max $k$)

NP-hard
NP-hard problems

- as hard as any NP problem.
- NPC problems are NP-hard.
- NP-hard need not be NPC
  - might not be decision problems
  - or might not have poly-time verification.

independent set

NP-hard

NP

NPC

decision (≤k?)

NPC

optimization (max k)

NP-hard

not many of these
NP-hard problems

- As hard as any NP problem.
- NPC problems are NP-hard.
- NP-hard need not be NPC
  - Might not be decision problems
  - Or might not have poly-time verification.

- Like NPC, solving an NP-hard problem quickly → same for all NP

Independent set

NPC

NP-hard

NP

Optimization (max k)

Decision (≤ k?)
NP-hard problems
- as hard as any NP problem.
- NPC problems are NP-hard.
- NP-hard need not be NPC
  - might not be decision problems
  - or might not have poly-time verification.
- like NPC, solving an NP-hard problem quickly → same for all NP

NPC = NP-hard & in NP
AN IMPORTANT DETAIL just mentioned here

For NPC problems, we measure input in terms of a finite alphabet (e.g., binary 1 = 1 bit; $k = \Theta(\log k)$ bits)

— unlike our treatment of constants so far

\[ 1 = O(1) \quad ; \quad k = O(1) \]

All this really means is that if you suspect a problem A is NP-hard (or NPC), to prove this you should measure the input in bits, and prove that you can transform problem A to a known hard problem in time polynomial in the number of bits representing the input to A.

Typically, your suspicion about A will arise after not being able to find a polynomial-time algorithm using regular O-notation. Again, typically, if your hunch is correct, measuring in bits won’t change much.