Completeness

Within a class, the *complete* problems are the hardest – if you can solve a complete problem, you can solve every problem in the class.

- **P-complete** – set of problems in P such that every other problem in P is polynomial-time reducible to one in the set
  - these are problems believed to be "inherently sequential" i.e. a parallel computer would not significantly speed them up
- **NP-complete** – set of problems in NP such that every other problem in NP is polynomial-time reducible to one in the set

Hardness

Hard problems are at least as hard as the hardest problems in the class but may not be in the class itself.

NP-hard – set of problems that NP-complete problems can be reduced to

- all NP-complete problems are also NP-hard
- FNP versions of NP-complete problems are NP-hard
- there appear to be problems that are NP-hard but not NP-complete
  - e.g. determining whether or not a given chess board configuration is checkmate

Common Myths

NP problems require exponential time.

Probably **yes**.

But technically **unknown**, because P != NP hasn't been proven.

- though it is generally thought that there are problems in NP that are not in P (and thus some problems in NP do require exponential time)

Also **no**, because there may be specific instances or classes of instances which can be solved in polynomial (or at least subexponential) time.

- e.g. maximum independent set – NP-hard/complete in general, O(n) for trees
- e.g. longest path – NP-hard/complete in general, O(n+m) for DAGs

Common Myths

NP-complete problems are difficult because the search space is so large.

framed as a series of choices, the running time depends on branching factor, longest path

No, it's not the size of the search space as such, but that you have to look at so much of it.

- compare to greedy – and even many dynamic programming – algorithms

these are also framed as a series of choices, but either only one option is needed for each choice (a branching factor of 1) or there are many repeated subproblems so most branches can be pruned

Common Myths

P is good and NP is bad.

Not necessarily –

- a polynomial-time algorithm may have large constant factors or exponents
- the exponential-time worst-case behavior may be rare
  - e.g. simplex algorithm for linear programming
- you might not need to solve large input sizes

Harder than NP?

Are there any problems harder than NP?

Yes.

- e.g. Presburger arithmetic

Presburger Arithmetic

Definition –

- contains constants 0 and 1 and the operator +
- axioms

1. $\neg(0 = x + 1)$
2. $x + 1 = y + 1 \rightarrow x = y$
3. $x + 0 = x$
4. $x + (y + 1) = (x + y) + 1$
5. Let $P(x)$ be a first-order formula in the language of Presburger arithmetic with a free variable x (and possibly other free variables). Then the following formula is an axiom:
   $(P(0) \wedge \forall x(P(x) \rightarrow P(x + 1))) \rightarrow \forall y\, P(y)$.

http://en.wikipedia.org/wiki/Presburger_arithmetic

Why is Presburger arithmetic interesting?

- it is *decidable* – an algorithm exists to determine if any given statement is derivable from the axioms
- the decision problem has worst-case runtime $2^{2^{cn}}$ for c > 0
- there are theorems of length *n* whose proofs have doubly exponential length
  - implies that there are computational limits on what can be proven by computer programs

Unknown Problems

Are there problems whose hardness is unknown?

Yes.

Two examples –

- graph isomorphism
  - given two graphs G and H, determine if there is mapping *f* from vertices of G to vertices of H such that if (x,y) is an edge of G, (f(x),f(y)) is an edge of H
  - thought to be between P and NP-complete – can often be solved quickly in practice
- integer factorization
  - given integers n and m, does *n* have a factor at most *m*?
  - known to be in both NP and co-NP, thought to not be in P or NP-complete
  - primality testing is in P

co-NP = can verify "no" answer in polynomial time

Other Complexity Classes

PSPACE – decision problems which can be solved by a Turing machine needing only polynomial space

EXPSPACE – decision problems which can be solved by a Turing machine needing an exponential amount of space

- how does PSPACE compare to P and NP?
  - bigger (probably)
- does EXPSPACE contain PSPACE?
  - yes
- are there problems in EXPSPACE that aren't in PSPACE?
  - yes