P, NP, NP-Complete, NP-Hard

Class P

- Decision problem: A question with a yes or no answer.
- P: A decision problem that can be solved in polynomial time. That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

Class P

- Example: Given a graph connected G, can its vertices be colored using two colors so that no edge is monochromatic. Algorithm: start with an arbitrary vertex, color it red and all of its neighbors blue and continue. Stop when you run our of vertices or you are forced to make an edge have both of its endpoints be the same color.

NP - Intro

- Suppose,
- you put a key into a lock and it doesn't really matter how "hard" the lock is, you can verify the solution. (lock being opened)
- On the other hand, it's very difficult to figure out what that key will be just by walking up to the door - you'd have to have a whole crapload of keys and try them one at a time. There's no deterministic way of figuring out which key (or solution) is the right one, you just have to try them.
- Verification is easier than finding the solution!

Class NP

- NP: A decision problem where instances of the problem for which the answer is yes have proofs that can be verified in polynomial time. This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

Class NP

- Example: Integer factorization is NP. This is the problem that given integers n and m, is there an integer f with $1 < f < m$ such that f divides n (f is a small factor of n)? This is a decision problem because the answers are yes or no. If someone hands us an instance of the problem (so they hand us integers n and m) and an integer f with $1 < f < m$ and claim that f is a factor of n (the certificate) we can check the answer in polynomial time by performing the division n / f.

- Take for example a Rubik cube. Solving a Rubik cube is actually an NP problem. If someone hands you a Rubik cube, if you tried to solve it by trying all possible combinations of turns, it would take forever.
- If on the other hand, if someone handed you a Rubik cube and a list of twists, you could find out quickly & tell whether the list was correct or not. Just apply the twists and see if at the end the cube is solved.

- Another example would be a hash of a file. You can validate the hash quickly, but it would take forever to come up with a file that matched a given hash (which is the whole point of a hash)

Class NP-Complete

- NP-complete: An NP problem X for which it is possible to reduce any other NP problem Y to X in polynomial time. Intuitively this means that we can solve Y quickly if we know how to solve X quickly. Precisely, Y is reducible to X if there is a polynomial time algorithm f to transform instances x of X to instances y = f(x) of Y in polynomial time with the property that the answer to x is yes if and only if the answer to f(x) is yes.

Class NP-Complete

- Example: 3-SAT. This is the problem wherein we are given a conjunction of 3-clause disjunctions (i.e., statements of the form

(x_v11 or x_v21 or x_v31) and (x_v12 or x_v22 or x_v32) and ... and (x_v1n or x_v2n or x_v3n) where each x_vij is a boolean variable or the negation of a variable from a finite predefined list (x_1, x_2, ... x_n)

NP C

- It can be shown that every NP problem can be reduced to 3-SAT. The proof of this is technical and requires use of the technical definition of NP (based on non-deterministic Turing machines and the like). This is known as Karp's theorem.

NP C

- What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time (one problem to rule them all).

NP Hard

- NP-hard: Intuitively these are the problems that are even harder than the NP-complete problems. Note that NP-hard problems do not have to be in NP (they do not have to be decision problems)
- an NP-hard problem is a problem for which we cannot prove that a polynomial time solution exists. NP-hardness of some "problem-P" is usually proven by converting an already proven NP-hard problem to the "problem-P" in polynomial time.

NP Hard

- a problem X is NP-hard if there is an NP-complete problem Y such that Y is reducible to X in polynomial time.
- But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time.
- Then if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

NP Hard

- The halting problem is the classic NP-hard problem. This is the problem that given a program P and input I, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one
- As an intuitive example, the optimization-version of traveling salesman where we need to find an actual schedule is harder than the decision-version of traveling salesman where we just need to determine whether a schedule with length <= k exists or not.

P Vs NP ?

- If you can understand something quickly, instantly recognise it, could you *do* it?

P Vs NP

- **What is the P versus NP Problem?**
- Suppose we have a large group of students that we need to pair up to work on projects. We know which students are compatible with each other and we want to put them in compatible groups of two. We could search all possible pairings but even for 40 students we would have more than 300 billion trillion possible pairings.
- The solution to above problem was found out in Matching Algorithm
- The class of problems with efficient solutions later became known as **P** for "Polynomial Time.

- But many related problems do not seem to have such an efficient algorithm.
- What if we wanted to make groups of three students with each pair of students in each group compatible (Partition into Triangles)?
- What if we wanted to find a large group of students all of whom are compatible with each other (Clique)?
- What if we wanted to sit students around a large round table with no incompatible students sitting next to each other (Hamiltonian Cycle)?
- What if we put the students into three groups so that each student is in the same group with only his or her compatibles (3-Coloring)?

- All these problems have a similar flavor: Given a potential solution, for example, a seating chart for the round table, we can validate that solution efficiently. The collection of problems that have efficiently verifiable solutions is known as **NP** (for "Nondeterministic Polynomial-Time)
- **P** = **NP** means that for every problem that has an efficiently verifiable solution, we can find that solution efficiently as well.

!

- the problems that cryptography is based on (for instance) are in NP.
- We rely on the fact that you can't just take a message and calculate the key to decrypt it; instead, you'd have to try many many keys. If we discover that P=NP, that will make it clear that there actually /is/ a way to solve these problems easily, making current methods of doing crypto useless - we'd have to go back and figure out how to keep messages secret from scratch.

P = NP ??

- P = NP: This the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences.
- It's clear that P is a subset of NP. The open question is whether or not NP problems have deterministic polynomial time solutions. It is largely believed that they do not