P and NP classes

In this lesson, we discuss the two most important complexity classes P and NP.

P - Polynomial Time Problems

The *P* stands for *polynomial time*. **Problems which can be solved in n^k, where k is some constant from the set of natural numbers (1, 2, 3 ...), lie in the complexity class P. More simply, problems to which we can find solutions in polynomial time belong to the complexity class P**. Note that when we say *problems* we are talking about decision problems. All the problems in P are decision problems. Strictly speaking, algorithms such as sorting, binary search, and tree-travels are algorithms that can be used to solve decision problems. However, we can always *find* a problem that a given algorithm will solve. For instance, you may pose a decision problem if an array is sorted. You can either scan over it and verify elements appear in ascending order, or you may just run the fastest sorting algorithm on a copy of the array and compare the result with the given array.

- Multiplication can be solved in polynomial time. The decision version of the problem will be if we multiply x and y, is the result z? We will simply multiply x and y and compare the product with z.
- Finding the greatest common divisor is also a problem in the P class.
- Is a given string S a palindrome? This decision problem can be solved in polynomial time.
- Given two strings *p* and *q*, is p a substring of q?

Problems solved on whiteboards during interviews can be either algorithms or decision problems. Sorting, binary search, tree traversals are all polynomial time algorithms but not decision problems. Whenever talking about problems in complexity classes, we are referring to decision problems. So if you state that selection sort is in complexity class P, your statement isn't valid. However, the algorithm can be run in polynomial time to answer a

decision problem that insertion sort solves, and that decision problem would be considered to be in P.

NP - Non-deterministic Problems

NP stands for nondeterministic polynomial time. Go back to the factorization example at the start of the section and note it's hard to come up with prime factors for a given number. That being said, if the prime factors are already given to you, then its very easy to verify if the solution is correct. You simply multiply the two factors and verify if the product equals the given number. **The class of NP problems consists of those decision problems whose solution can be verified in polynomial time. The problems themselves may or may not be solvable in polynomial time.** We don't care about how we get the solution, but once we have the solution, we can prove it is correct in polynomial time. A few examples appear below:

- **Graph coloring problem**: Given an undirected graph, can we color nodes using only three colors in such a way that no two connected nodes have the same color? Note given a coloring, we can just run a graph traversal algorithm to verify that the solution is correct.
- **Subset sum problem**: Given a set of integers and a target integer, can we choose any combination of integers from the set in such a way that their sum equals the target? Note to answer this question, we'll need to exhaustively search the entire problem space, i.e. find all combination and see if the sum of any one of them equals the target. On the contrary, if given a combination, its very easy to *verify* in polynomial time if the given combination's sum equals target.
- **String Matching**: Don't forget that string matching is in P as well as in NP. We can verify if a string p is a substring of another string q in polynomial time by simply running any string matching algorithm and verifying the claim.

For the first two examples we don't know of a polynomial time solution that will solve either of the problems. However, that doesn't mean that no such algorithm exists. Theorists have not been able to prove otherwise either. Just because no one has been able to come up with efficient solutions to these problems isn't proof enough that no polynomial algorithm exists that will solve these problems. A formal proof that a solution does or doesn't exist has

solve these problems. A formal proof that a solution does or doesn't exist has been lacking thus far. It is our inability to provide such a proof that makes the study of complexity intriguing.