**Module 7 Notes – P vs NP/NP-Completeness**

* We have three classes of problems:
  + **P (Polynomial Time):** Problems solvable in polynomial time, more specifically problems
    - that can be solved in time *O(nk)* for some constant *k*, where *n* is the size of the input to the problem
    - Also known as “deterministic problems”
  + **NP (Nondeterministic Polynomial Time):** Problems that are “verifiable” (can de verified/checked) in polynomial time. If we’re given the solution to a problem, we can verify this given solution somehow, using polynomial time. (i.e. a sudoku puzzle. If we’re given the answer to one, we can easily and quickly verify its validity. But if we try to solve the puzzle on our own, it would take a long time)
    - **Any problem in P is also in NP, since when a problem is in P then we can solve it in poly.** **time without even being supplied the solution/certificate.**
    - **For now, we say P ⊆** **NP**
  + **NPC (NP-complete):** Problems in this class if it is “NP-hard” meaning it is a difficult problem located with the NP class.
    - The textbook states that if *any* NPC problem can be solved in poly time, then *every* problem in NP has a poly-time algo
* Any bipartite graph with an **odd** number of vertices is non-hamiltonian
* **Cook’s Theorem**
  + **SAT is in P, if and only if P = NP**
  + Must be known that this is a theorem and hasn’t been solved yet
* Some problems are **intractable:**
  + As they grow larger, we cannot solve them in a reasonable amount of time
  + What constitutes reasonable time? **Polynomial time**
  + Ex: O(n2), O(n3), O(1), O(n lg n)
  + **NOT POLY TIME:** O(2n), O(nn), O(n!)
* Problems not in P ca be either:
  + **Intractable**
  + **Unsolvable**
* **Examples of Intractable Decision Problems:**
  + **Hamiltonian Cycle (HAM-CYLCE) –** Given a directed graph *G = (V, E)* does there exist a simple cycle C that visits every vertex?
  + **CIRCUIT-SAT –** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?
  + **Travelling Salesman** **(TSP)** – Given a weighted graph *G = (V, E)*
* **NP problems have two stages:**
  + **Nondeterministic (“guessing”) stage –** generate randomly an arbitrary string that can be thought of as a candidate solution (“certificate”)
  + **Deterministic (“verification”) stage –** take the certificate and the instance to the problem and return **YES if the certificate represents a solution**
  + **NP algos need verification in poly-time**
  + Basically, if we’re given a certificate/solution to the problem, we can verify/certify it is correct in poly-time
  + **0-1 Knapsack, Hamiltonian Cycle (HAM-CYCLE), 3-SAT are examples of NP problems**
  + **All problems that can be solved in poly-time can be verified in poly-time**
    - Searching and sorting algos are within P since they can easily be solved and verified
    - Problems in NP may be in P, but we don’t have the proof/answer yet for these i.e. they haven’t been discovered yet. Most scientists say no but we still have no idea yet.
* Biggest question is **whether NP ⊆ P or P = NP**
  + i.e. if it is always easy to check a solution, should it also be easy to find a solution?
    - If YES: Efficient algos for KNAPSACK, TSP, FACTOR, SAT, etc.
    - If NO: No efficient algos possible for the above listed algos
    - Consensus opinion for **P = NP? Probably no.**
* **NPC problems are considered the “hardest” problems in NP (“NP-hard”)**
  + **If you can solve an NPC, then you can solve all problems in NP**
* Shortest simple path graph problem would be poly-solution, P
  + Modifying to *Longest* simple path would be NPC
* We will label **all problems in NPC as “problems that are B”**
* **What steps do we take to prove a problem is B is NPC?**
  + Pick a known NPC problem A. Reduce A to B
    - Describe a polynomial time transformation/reduction that maps instances of A to instances of B, s.t.“yes” for B = “yes” for A
    - Prove the transformation works
    - Prove it runs in poly-time
    - **By proving step 1, you have proved that problem B is NP- Hard**
  + Prove B ∈ NP
    - Show that a solution to B can verified in poly-time
  + Prove steps 1 and 2 and you’ve proven **B is NPC**
* **NP-Hard problems can’t even be verified in poly-time**
* Reduction is a way of saying that one problem is “easier” than another
* See slide 40 of Module Notes for NPC problems and their reductions