PS11: Class Complexity

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. In addition to discussing with your cohortmates, you may discuss the problems with any other current CS3102 students you want, and use any resources you want except for any materials from previous offerings of this course or complete solutions that might be available on the web, which are not permitted. Sharing and subsequently submitting any text, code, images, figures, etc. constitutes plagiarism, so make sure all submitted materials are created exclusively by members of your cohort.

Problem 1: Asymptotic Operators

Provide a convincing argument for each statement below that uses the formal definitions of the various asymptotic operators from class. While your argument does not need to be formal, your strategy must correctly use the definitions.

(a) $1000n^2 \notin O(n^{1.5})$

(b) $\frac{n^2}{1+n} \in \Theta(n)$

(c) $n^2 \notin \Omega(2^n)$

Problem 2: Running Time Analysis

Consider the *INCREMENT* problem defined below:

> **Input:** A binary encoding of a natural number, x, with *most significant bit first.*
>
> **Output:** A binary encoding, most significant bit first, of $x + 1$.

For example $INCREMENT(1011) = 1100$ (the input was a binary representation of 11, the output is a binary representation of 12).

1. Characterize using asymptotic notation the *worst-case* running time cost for computing *INCREMENT*, where cost is the number of steps required by a standard output-on-tape Turing Machine. You should be able to get a tight bound using Θ notation. Express the running time as a function of the variable n, where n represents the size of the input (in bits).
2. Now re-express the worst-case running time that you gave in terms of the *value* of the natural number x. That is, take the function expressing the running time of the Turing machine you described in the previous part and re-express that function in terms of the value of the natural number given as input (i.e. x) instead of in terms of the number of the number of bits in the input (i.e. n).
3. Suppose I had a Turing machine whose input was a natural number x and whose running time was $\Theta(x)$. Suppose n represents the length of a representation of x in bits. First, provide the running time of this Turing machine re-expressed in terms of n. Next, indicate whether this a polynomial-time Turing machine.

Problem 3: Fact Checking

This *P vs NP on TV* Computerphile video (https://www.youtube.com/watch?v=dJUEkjxylBw) with Simon Singh (who normally does a great job presenting technically challenging topics in a very accessible, but essentially correct, way) contains a faulty discussion of the P vs NP question. Watch this video, and identify at least 4 incorrect or misleading statements (we were able to identify 2 between time stamps 0:25 and 0:30

alone). Propose corrections for all such statements you identify. It's fine if some of the statements relate to the same error/misconception.

Problem 4: Who Wants To Be A Millionaire?

For each of the following statements below, indicate whether it would prove or disprove $\mathrm{P} = \mathrm{NP}$. If it would resolve it, indicate the direction it would be resolved (equal or not equal). If it would not resolve it, explain why.

(a) A problem from class EXP is found to be in class P.

(b) A problem from class EXP is found to be in class *NP-Hard*.

(c) A problem from class *NP-Hard* is found to be in class P.

(d) A problem from class *NP-Hard* is found to be in class NP.

(e) A problem from class *NP-Hard* is found to be outside of the class P

(f) A problem from class *NP-Complete* is found to be outside of the class P

Problem 5: Flawed Reductions

Definition 1 (k-CNF) We say that a formula is in conjunctive normal form (CNF for short) if it is an AND of ORs of variables or their negations. E.g. $(x_7 \vee \overline{x_{22}} \vee x_{15}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7})$ is in CNF. We say that it is k-CNF if there are exactly k variables per clause (group of variables combined with OR).

Recall that 3-SAT is in *NP-Hard*, where 3-SAT requires determining whether there exists at least one way to assign Boolean values to each variable in a 3-CNF formula so that the formula evaluates to True.

The 4-SAT problem requires determining whether there exists at least one way to assign Boolean values to each variable in a 4-CNF formula so that the formula evaluates to True.

Below we've presented several different flawed approaches for demonstrating that 4-SAT is in *NP-Hard*. Identify the main flaw or flaws in each approach, and explain things the proposed approach is misunderstanding.

(a) Convert a given 3-CNF formula into a 4-CNF formula by putting `False` into each clause. For example, if the 3-CNF formula given to 3-SAT was $(x_7 \vee \overline{x_{22}} \vee x_{15}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7})$, we would add `False` to each clause to create the 4-CNF formula $(x_7 \vee \overline{x_{22}} \vee x_{15} \vee \texttt{False}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7} \vee \texttt{False})$. This new 4-CNF formula will be satisfiable if and only if the original 3-CNF formula is satisfiable.

(b) Convert a given 4-CNF formula into a 3-CNF formula by breaking each clause of 4 variables into two clauses of 3 variables in such a way that both of the new clauses are satisfiable if and only if the original clause was satisfiable. For example, if the 4-CNF formula that was given to 4-SAT was $(x_7 \vee \overline{x_{22}} \vee x_{15} \vee x_9) \wedge (x_{37} \vee x_{22} \vee \overline{x_7} \vee x_{12})$, we would introduce two new variables (one per original clause) and break up each clause to create the 3-CNF formula $(x_7 \vee \overline{x_{22}} \vee n_1) \wedge (\overline{n_1} \vee x_{15} \vee x_9) \wedge (x_{37} \vee x_{22} \vee n_2) \wedge (\overline{n_2} \vee \overline{x_7} \vee x_{12})$.

(c) In class, we showed that 1-SAT is easier than 2-SAT (the problem of satisfying a 2-CNF formula) which was easier than 3-SAT (the problem of satisfying a 3-CNF formula). This demonstrates that it is harder to satisfy CNF formulas with more variables per clause. Since 3-SAT is in *NP-Complete*, and 4-CNF has more clauses than 3-CNF, it must be harder, so 4-SAT is in *NP-Hard*.

Problem 6: TAUT

Definition 2 (DNF) We say that a formula is in disjunctive normal form (DNF for short) if it is an OR of ANDs of variables or their negations. E.g. $(x_7 \wedge \overline{x_{22}} \wedge x_{15}) \vee (x_{37} \wedge x_{22} \wedge \overline{x_7})$ is in DNF. We say that it is k-DNF if there are exactly k variables per clause (group of variables combined with AND).

We know that 3-SAT is in *NP-Complete*, where 3-SAT requires determining whether there exists at least one way to assign Boolean values to each variable in a 3-CNF formula so that the formula evaluates to True.

The 3-TAUT problem requires determining whether *every* assignment causes a 3-DNF formula to evaluate to True (i.e., no assignments will cause the formula to evaluate to False).

(a) Show that 3-TAUT is in *NP-Hard*.

(b) 3-TAUT is not known to belong to *NP*. Give an intuitive reason why it is difficult to show that 3-TAUT belongs to *NP*.

(For the historical significance of the 3-TAUT problem, see the optional video on the history of the Cook-Levin Theorem.)

