Practice Final

1. [TRUE/FALSE/UNKNOWN] If VERTEX-COVER has a polynomial time algorithm then 3-SAT has a polynomial time algorithm.

   TRUE. VERTEX-COVER is known to be NP-Complete, and we specifically reduced 3-SAT to this in class.

2. [TRUE/FALSE/UNKNOWN] There is a polynomial time algorithm to solve the INDEPENDENT SET PROBLEM.

   UNKNOWN. INDEPENDENT SET IS NP-Complete and there is no known polynomial time for any problem in this class

3. [TRUE/FALSE/UNKNOWN] Given a graph G with n vertices, deciding if there is a clique of size 195 is solvable in polynomial time.

   True. A naive algorithms runs in the $O(n^{196})$, which is a polynomial.

4. [TRUE/FALSE/UNKNOWN] All problems in the class NP can be reduced to INDEPENDENT-SET.

   True. INDEPENDENT SET IS NP-Complete, which means that all problems in NP can be reducted to it in polynomial time

5. Palindrome substring. Given a string, find the longest palindrome substring. Here is an example string. Let us identify all the palindromic substrings here.

   Input String : ACECCBA

   One letter palindrome substring A, B, C, E Two letter palindrome substring CC Three letter palindrome sequence CEC

   Give a Dynamic program to compute the length of the longest palindromic substring.

   An alternative discussion for this can be found here:

   `https://leetcode.com/articles/longest-palindromic-substring/`

   Solution: We want to find the Longest Palidromic Substring of a string S[1...n].

   Lets define $LPS(S, i, j)$ to be the true false value, answering the question, is the Substring starting at position i and ending at position j a palindrome?

   ```
   LPS(S,i,j)
   % BASE CASES

       if $i > j$, return 0 % if $i > j$ then our string has length 0
       if $i = j$, return 1 % any string of length 1 is a palindrome
       if $i + 1 = j$ AND $S[i] = S[j]$, return 2 % palindromes like aa or bb.

   %GENERAL CASE

       if $S[i] = S[j]$, return LPS(S,i+1,j-1)
       else return FALSE
   ```

   The general case says, if you are thinking about the string from character i to character j, if they are the same, then maybe S[i...j] is a palindrome, so check S[i+1...j-1].

   If S[i] isn't the same as S[j], then S[i..j] cannot be a palindrome.

   Turning this recursive definition into a DP, we create a table LPS(1..n, 1..n). This is initialized with TRUE along the diagonal, and so that LPS(i,i+1) is T if S[i] = S[i+1] corresponding to the case where that are two of the same letters in a row.

Then the table is filled out diagonal by diagonal, perhaps with a loop like:

```
maxLength = 0
for i = 1..n-2
    for k = 2 .. n-1 % Loop over how long the substring is
        j = i+k
        if $S[i] = S[j]$ AND LPS(S,i+1,j-1)
            LPS(i,j) = TRUE,
            maxLength = k
        else set LPS(i,j) = FALSE,
```

This code fills out the table that says which substrings are palindromes, and saves the length of the longest found substring as it goes.

6. Profs for Dinner: You want to invite some Profs from the CSE Department for dinner (wonder why?!). As is well known, Profs are jealous of each other and end up not even on speaking terms. A grad student in the local sociology department created a database which is a two dimensional $n \times n$ matrix H (where n is the total number of CS Profs) with $H[i, j] = 1$ if Prof i is on speaking terms with Prof j and 0 otherwise (there are many 0s but at least lets assume H[i, i] = 1 for all i!) To make a good party, you want to invite at least k Profs, but so that all are willing to talk to each other.

   First, show that the problem of deciding if there is such a group of Profs is in the class NP.

   To prove a problem is IN the class NP, we show that we can verify a solution. In this case, the solution is defined as a list of k professors. Given that list, we can check the every pairs of professor i, j in the group is on speaking terms, so $H[i, j] = 1$.

   Second show that this problem NP-complete by reducing INDEPENDENT SET to this problem.

   The INVITE PROFS problem is exactly the CLIQUE problem, so this is asking to reduce INDEPENDENT SET TO CLIQUE. A nice video of this is here: `https://www.youtube.com/watch?v=pIYR2xM7F7Q`

   To define this reduction, let $IS(G, k)$ be the question: Does graph G have an independent set of size k?

   To solve this problem, we construct the complement graph $\bar{G}$. To solve the question $IS(G, k)$ we simply return the answer $CLIQUE(\bar{G}, k)$.

   Proof: If there is a k-clique in $\bar{G}$ then there are k nodes that have every possible edge connecting them in $\bar{G}$. Those k-nodes will have zero edges in G (because G has no edges where $\bar{G}$ has edges. Therefore those k-nodes form and independent set.

7. Suppose that someone gives you an oracle (a magic algorithm whose running time you can consider to be constant) to answer the CLIQUE decision problem: Does graph G have an clique of size k?. Show how to use this to solve for the actual elements of the largest clique in Graph G. If there are multiple different cliques of size k, you can return any of them. Give the run-time of your algorithm (assuming the the oracle runs in constant time), and argue that your algorithm is correct.

   Let G be defined by a set of vertices V and a set of edges E, and let the number of vertices be n.

```
if CLIQUE(G,k) is FALSE, return 'you tricked me, there is no solution'
else
    for i = 1..n
         Make G' as G without v(i) (deleting v(i) from V.  delete edges that touch v(i) from E\\.

         if CLIQUE(G',k) is TRUE
              G = G'              % ditch v(i), we don't need it
         else
              output v(i)    % this will be part of our solution.
```

Run time: this is $O(n)$, it loops through all nodeso= once.

Correctness Proof:

Claim: This algorithm outputs k-nodes that are a clique

Proof: The algorithm loops through all nodes. IF deleting that node does not change the clique size to one smaller, then the node is deleted. IF deleting the node DOES change keep there from being a clique of size k then we keep it.

LET A be the final clique of size k. Removing any node not in A will maintain a clique of size k and that node will be removed. Nodes in A have the property that when they were removed, they made the graph no longer have a clique, so they were output.

8. For a graph G(V,E), an "ALMOST INDEPENDENT SET" (AVC) is a set of nodes V' where there is at most one edge connecting nodes in V'. Prove that "ALMOST INDEPENDENT SET" is NP-Complete.

   (a) Prove ALMOST INDEPENDENT SET is in the SET NP

   (b) Describe how to reduce a regular Vertex Cover problem to an ALMOST INDEPENDENT SET problem.

   (c) Argue that the solution to the regular Independent Set problem on the original input is always the same as the solution to the ALMOST INDEPENDENT problem you create.

   (d) Argue that your translation to create the ALMOST INDEPENDENT problem is polynomial time.