1 What is NP?

NP is a class of languages that have polynomial time verifiers

1.1 Verifiers

A verifier for a language A is an algorithm V where
$A = \{w|$ V accepts $\langle w, c\rangle$for some string c $\}$
The time it takes to run a verifier is measured in terms of the length of w. In this system, c is the certificate

2 NP-COMPLETENESS

NP-Complete problems are problems that are both in NP and NP-hard. An NP-Hard problem is a problem that is at least as hard as all other NP-hard problems. Putting these toegether, if one NP-complete problem has a polytime solver, they all do. But, if one NP-complete problem can be shown to have no polytime solver, none of them do

2.1 The grandfather of NP-Complete problems: SAT

SAT = $\{\langle\phi\rangle \,|\phi$ is a satisfiable boolean formula$\}$
A satisfiable boolean formula is a boolean formula such that some assignment of variables makes the formula output true. Truth assignments are often labeled τ, and assigning the variables in ϕ with the assignments in τ is $\tau(\phi)$
This proof looks too big to put here

3 Polytime reductions

Language A is polytime reducible to language B, $A \leq_p B$ if a TOTAL, POLYTIME, and COMPUTABLE function $f : \Sigma^\star \to \Sigma^\star$ exists with the property that $w \in A \iff f(w) \in B$.
So to reduce A to B, create a function which maps yes instances of A to yes instances of B, and no instances of A to no instances of B, this is the contrapositive.

3.1 How do polytime reductions relate to time complexity?

If $A \leq_p B$, or A polytime reduces to B, and B is in P, then, here's a polytime solver for A

```
solveA⟨$w$⟩ :
    $w_b = f(w)$
    return solveB($w_b$)
```

Since f is a polytime computable total function that maps yes instances of A to yes instances of B, and solveB is polytime, if solveB accepts, then w must be in A, and if solveB rejects, w must NOT be in A. Additionally, since B and f are polytime computable, solveA must be polytime computable. This is the core of polytime computability

3.2 Polytime reductions and it's incomprehensible notation

$\leq_p$ is one of the most confusing and poorly implemented operators I have ever seen. Typically, symbols notate a relation between two objects. The "justification" for $\leq_p$ is that if $A \leq_p B$, A is "Less than or equal to B" in terms of hardness. This is complete gibberish. This is an "operator" which communicates the thing one can prove using the existence of a relation between two objects. Imagine if we wrote all notation this way! There would be COMPLETE ANARCHY!!!! Not only that, but this convention encourages un-mathematical, un-creative, instrumentalist applications of mathematical reasoning. This is wholly unfit for a theoretical course. This is unlike all other notation, and the person who invented it should be candidate #2 for NASA's man-on-the-sun space mission right after Elon Musk.

3.2.1 How to remember this difficult notation

Rule 1: read right to left: if $A \leq_p B$, then a polytime reduction FROM A, TO B exists, I.e. inputs in A, go TO, B. The english phraseology gets this exactly correct. A polytime reduction goes from A to B, and the inputs of A, go TO B.

3.3 Polytime reduction definition of NP-complete

Using our new approach, we can define NP-completeness to be
"A language B is NP-complete if it satisfies two conditions:
1. B is in NP
2. if A∈NP, then $A \leq_p B$ for any A in NP

This works because if B has a polytime solver, then A must have a polytime solver. This means that B's solvability in polynomial time makes a claim about every single NP-complete problem's solvability in polytime, which is the point of the NP-complete class

4 The first reduction: SAT$\leq_p$3SAT

PROVE 3SAT is in NP!
3SAT = $\{\phi | \phi$ is a 3-cnf formula which is satisfiable$\}$
A 3-cnf formula, or "conjunctive normal form formula with 3 variables in each clause" is composed of clauses as follows: $(x \vee y \vee \neg x) \wedge (z \vee p \vee \neg x) \wedge \cdots$. Notably, these formulas have no constants. An easy way to remember is that cnf formulas are "sad" $({}^{\odot}\vee^{\odot})$
WTS: SAT$\leq_p$3SAT. Showing SAT Polytime reduces to 3SAT will show that if 3SAT is in P, then SAT is in P. This proves that 3SAT is NP-complete
The details of this reduction are not super important, since finding an algorithm to take any formula composed of literals, ands, or's and nots and transforming it into a 3-cnf formula is very boring and not required knowledge. But an appropriate algorithm would have the following passes:
1. Transform literals into tautologies, $\neg x \vee x = 1$, $\neg x \wedge x = 0$
2. Use de morgan's law to make all nots go directly in front of variables
3. probably many others

Intuitively, such an algorithm makes sense since we are used to transforming logical statements into other equivalent statements, since this was tested in a67 and many other courses.
By the principle of arm waiving, we now have a function $f(\phi) \rightarrow \phi'$st if ϕ is a boolean formula, ϕ' is a 3-cnf formula. and ϕ is equivalent to ϕ'.
Now, for the proof:
If $\phi \in$ SAT, then ϕ' is satisfiable, which implies since it is 3-cnf, it is in 3SAT
(using the contraposition)
If $\phi \notin$ SAT, then ϕ' is not satisfiable, which implies it is not in 3SAT
QED

5 The first REAL reduction: 3SAT$\leq_p$CLIQUE, Showing CLIQUE is NP-complete

CLIQUE = $\{\langle G, k\rangle \mid$ G is a graph which contains a clique of size k$\}$

5.1 Proof that CLIQUE∈NP

Here's a polytime verifier for CLIQUE:
Let the certificate C be a set of vertices that form a clique, and let G be a graph

```
VerifyCLIQUE⟨⟨G, k⟩ , C⟩ :
    if |C| < k:
        return false
    for each vertex in C:
        check that it connects to every other vertex in c
        if it doesn't return false
    return true
```

Worst case, G=C, so at worst, this algorithm runs in n^2 time. This is sufficient justification to show that CLIQUE∈NP

5.2 Proof that CLIQUE is NP-complete

A clique is a set of vertices where each vertex in the clique is connected to every other vertex in the clique with an edge.
The way we will go about reducing these is to come up with some equivalent structure to clauses, and conjunctions in graphs, often called widgets or gadgets. Since some have argued that graphs are a "universal" data structure capable of storing all other data structures, this should be easy!
The steps to make this construction go as follows, Given 3-cnf formula ϕ, output G and k as follows:
1. For each variable in ϕ, add a corresponding vertex in Gnamed after the variable
2. Connect every vertex to every other vertex in G with an edge

3. Remove all edges between "contradictory" vertexes like $\neg x$ and x
4. Remove all edges between vertexes that appear in the same clause (so if $(c_1 \vee c_2 \vee c_3)$ appears in ϕ, no edge should exist from $c_1$to c_2 to c_3)
5. set k to be the number of clauses in ϕ
Now, to argue that $\langle\phi\rangle \in$ 3SAT $\iff \langle G, k\rangle \in$ CLIQUE
If $\langle\phi\rangle \in$ 3SAT, then a satisfiable truth assignment τ exists such that every clause in ϕ returns true. This means at least one variable in each triplet evaluates to true. If one puts these three variables (henceforth vertices) into a set, these vertices will form a k-clique.
Now, for the reverse direction, suppose that G contains a k-clique:
We know the elements of this clique must all appear in distinct clauses since there are no edges to vertices of the same clause. Similarly, we know none of these vertices have edges to contradictory labels by the given construction above. Now, create a truth assignment τ with the labels from the vertices inside the k-clique. Therefore, this truth assignment must satisfy ϕ. Therefore, if $\langle G, k\rangle \in$ CLIQUE, then $\langle\phi\rangle \in$ 3SAT

5.3 What is this?

What are the gadgets?
The core idea is that a clique must contain k vertices who's variables can ALL BE TRUE at the same time, AND are of different clauses. This is a corollary of satisfiability. This is why the two restrictions on edges exist. In short, the reasoning is that "If k variables can be true at the same time (SAT), then an edge exists between all of them", and "if k vertices form a clique (CLIQUE), then they can all be true at the same time, and are of different clauses

5.4 Time Complexity Analysis

I argue that our mapping reduction is polytime because

1. It iterates over ϕ once to create G's adjacency list: $\mathcal{O}(n)$
2. It iterates over G's adjacency list once to connect all edges: $\mathcal{O}\left(n^2\right)$
3. It iterates twice more to ensure a proper construction: $\mathcal{O}\left(2n^2\right)$

Therefore, this is a polytime construction

5.5 The big result?

CLIQUE is NP-complete

6 Vertex Cover is NP Complete

VERTEX-COVER=$\{\langle G, k\rangle$ G is an undirected graph which has a k-node vertex cover$\}$
A vertex cover VC is a set of vertices such that all edges either start or end at a node in VC

6.1 Proof that VERTEX-COVER$\in$NP

Here's a polynomial time verifier for VERTEX-COVER, where C is a set of nodes which are in the vertex cover

```
VerifyVERTEX-COVER$\langle\langle G, k\rangle, C\rangle$ :
    if $|C| < k$:
        return false
    For each vertex in C:
        mark each of its adjacent edges as covered
    for each edge in G:
        if the edge is not marked as covered:
            return false
    return true
```

This verifier clearly runs in polynomial time since it goes through every edge of G once, which serves a maximum complexity of $\mathcal{O}\left(n^2\right)$, where n is the number of vertices, and n^2 is the number of edges in a fully connected graph.

6.2 Proof that VERTEX-COVER is NP-Complete

While it might seem reasonable to reduce from CLIQUE $\leq_p$ VERTEX-COVER, we are instead going to choose 3SAT
WTS: 3SAT $\leq_p$ VERTEX-COVER
So, our reduction takes inputs in 3SAT, and transforms them into inputs into VERTEX-COVER, namely a graph G, and a size k.
Our input is a 3-cnf formula ϕ
To make this reduction, we will
1. Take every variable in ϕ, and add a vertex in G.
2. If x_i and $\bar{x}_i$appear in G, add an edge between x_i and $\bar{x}_i$.
3. If 3 variables occur in a clause of ϕ, add a new copy of each variable into G, and add an edge between each new vertice to every other vertice (GRAPH NEEDED)
4. Connect all x_i's to other x_i's and all $\bar{x}_i$to every other $\bar{x}_i$
5. set k to be the number of variables in ϕ + 2 times the number of clauses in ϕ
Now, for the justification:
If $\langle\phi\rangle \in$ 3SAT, then, select a satisfiable assignment τ, and put all the variable widgets that are assigned true in τ into our vertex cover. Now, take 1 variable that was assigned true from each clause, and put the other two variables in its clause into the vertex cover. Since our vertex cover has 1 variable per variable widget in the VC, and 2 clause widget variables per clause in the VC, it must have k vertices in the vertex-cover. Therefore, a vertex cover of size k exists.
If $\langle G, k\rangle \in$ VERTEX-COVER, then our Vertex Cover VC, must select 1 and only 1 variable per variable widget, since if it selected 2, it would have to skimp out on including clause variable widgets, which wouldn't form a full vertex cover. In addition, the vertex cover must select 2 nodes from each clause, where the unselected node is set to true, since if it didn't it would miss the edge from a node which was set to false to the variable widget. This forms a VC of size k, and the selected variables in the vertex clause must form satisfiable truth assignment since one unselected-but-true edge must exist in each clause.

7 HAMPATH is NP-Complete

A hamiltonian path is a path that visits each vertex of a graph exactly once. A hamiltonian path from s to t consists of a hamiltonian path with its head at s and tail at t. (heads and tails since these are directed graphs)
Formally: HAMPATH = $\{\langle G, s, t\rangle \mid$ G contains a hamiltonian path from s to t$\}$

7.1 HAMPATH$\in$NP

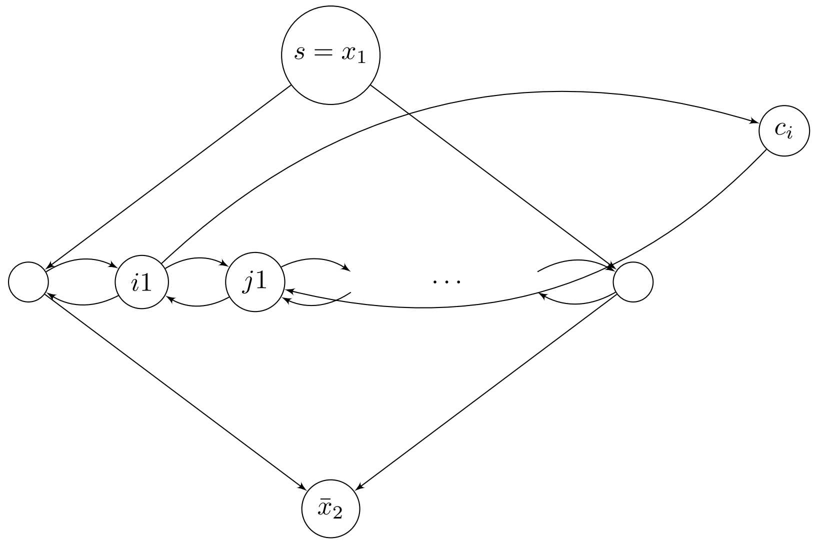
Here is a polytime verifier for HAMPATH. Let C, the certificate, be a hamiltonian path (a set of vertexes and edges) from s to t.

```
VerifyHAMPATH$\langle\langle G, s, t\rangle, C\rangle$ :
    Trace along the path C: and mark edges G which were visited. If an edge that was previously visited is visited
    again, reject
    check if every vertex in G, if all were visited, accept
    else reject (or loop?)
```

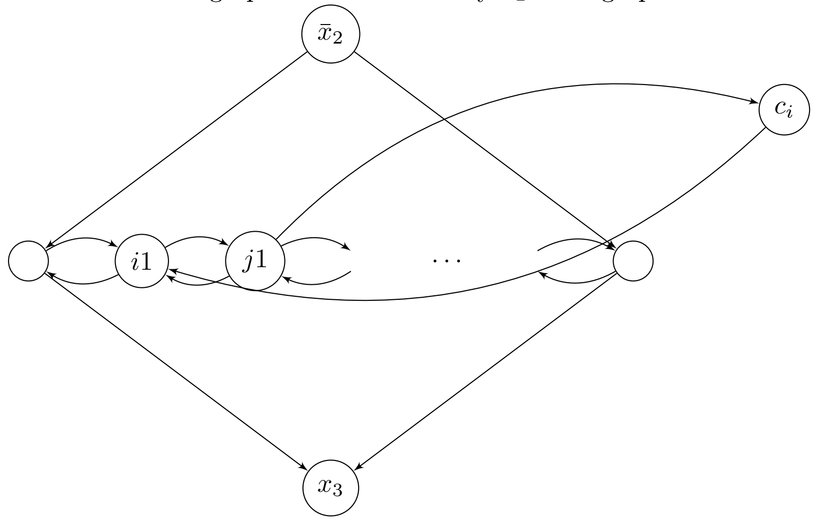
This verifier goes through the path C, which is $\mathcal{O}(n)$. Then, checking every vertex is visited takes $\mathcal{O}(n)$. Therefore, this verifier is polynomial time and takes $\mathcal{O}(n)$.

7.2 HAMPATH is NP-Complete

WTS: 3SAT $\leq_p$ HAMPATH
To do this, we need to take 3-cnf formulas ϕ and turn them into paths which contain a hamiltonian path if and only if ϕ is satisfiable. The way we do this is with two primary structures: The variable widget and the clause widget. The structure of the variable widget is such that a truth assignment is given to going left or right.



Note: These two graphs are connected by $\bar{x}_2$. The graph below shows what happens when a negated variable occurs



The number of these "vertices" in the middle is equal to 3 times the number of clauses. 3 times allows for 1 buffer node between pairs If a variable x_i is in clause c_j, then add an edge from the jth pair in the ith diamond to the jth clause node. But, if a variable $\bar{x}_i$ appears in clause c_j, then add two edges going in the reverse direction.
Finally, let G = the graph we have just constructed, and let s be the first variable widget, and t be the last variable widget. This is how we construct G,s, and t.
How this works: One can only assign 1 value (left/right or true/false) to each variable widget, and each clause widget must be gone through at least once by a single variable. This simulates a truth assignment, and that that truth assignment assigns true to each clause iff there is a satisfiable truth assignment for ϕ.

7.2.1 If $\langle \phi \rangle \in$ 3SAT

Then ϕ has a satisfiable truth assignment τ. Following the variable widget convention, we will go left/right according to what the truth assignment tells us to do. When we encounter the option to visit a clause, we will take it unless the clause has already been satisfied (i.e. visited). Since this formula is satisfiable, all clause nodes will be hit, as well as the top and bottom of each variable widget, and all nodes inbetween. This implies that $\langle G, s, t \rangle \in$ HAMPATH.

7.2.2 If $\langle G, s, t \rangle \in$ HAMPATH

Then G has a hamiltonian path from s to t, where s is the first variable, and t is the last variable in a truth assignment τ. This hamilton path must cross through each clause widget once, and must cross through each variable widget once. This implies that there must be a truth assignment such that every clause is satisfied, which implies that τ is a satisfiable truth assignment for ϕ. Therefore, $\langle \phi \rangle \in$ 3SAT

8 Assignment question: 2b

8.1 Prove that DISJOINT-CLIQUE is in NP

8.2 Show DISJOINT-CLIQUE is NP-Complete

We will reduce from CLIQUE $\leq_p$ DISJOINT-CLIQUE

To transform an input to CLIQUE $\langle G, k\rangle$ to an input $\langle G', k'\rangle$ to DISJOINT-CLIQUE, do as follows:
1. Put all vertices with k-1 or more edges into a group C
2. Let $G' = C + C'$where $C'$is all of the edges and vertices in C but with an added $'$ to the vertex names
3. Let $k' = k$
Pf that this reduciton works:

8.2.1 Assume $\langle G, k\rangle \in$ CLIQUE,

If G has a clique of size k, then all edges in the clique must have k-1 or more edges leading out of it. This implies that all these edges are cloned into G', which implies that G'has two cliques of size k (or more). This implies that $\langle G', k'\rangle \in$ DISJOINT-CLIQUE

8.2.2 Assume $\langle G', k'\rangle \in$ DISJOINT-CLIQUE

If G' has at least two cliques of size k, then suppose for purposes of contradiction that G has no clique of size k. If G has no clique of size k, then no subset of C' must form a clique of size k. This implies that none of the components of C form a clique of size k. This implies that G', which equals $C + C'$, has no cliques of size k, which is a contradiction to the assumption that G' has at least two cliques of size k. Therefore, if $\langle G', k'\rangle \in$ DISJOINT-CLIQUE $\implies$ $\langle G, k\rangle \in$ CLIQUE

8.2.3 This reduction is polytime

Finding all the vertices with $k-1$ or more edges takes $\mathcal{O}\left(n^2\right)$ time, so this reduction is polytime

8.3 Find a non-decision version of DISJOINT-CLIQUE

MAX-DISJOINT-CLIQUE
Input: A graph G
Output: A set of vertices V_1, V_2 such that V_1 and V_2 are disjoint cliques, and there are no two disjoint cliques of greater size in G
I did not want to prove this

9 DOUBLE-HAMPATH is NP-complete

9.1 Show that DOUBLE-HAMPATH$\in$NP

9.2 Show that DOUBLE-HAMPATH is NP-Complete

WTS: HAMPATH $\leq_p$DOUBLE-HAMPATH
To construct this, if $\langle G, s, t\rangle$ is an input to HAMPATH, we must map it to an input $\langle G', s', t'\rangle$ to DOUBLE-HAMPATH

9.3 Find a non-decision version