Problem 1, CS21 Set 5, Matt Lim

(a) False. To prove this we must find a language L that has the following properties: L $\in$ EXP and L $\notin$ TIME($2^{n^{100}}$). To find this L, we will use the Time Hierarchy Theorem. Consider the proper complexity function $f(n) = 2^{n^{100}} \geq n$. Then we have that $f(2n)^3 = (2^{2n^{100}})^3$. So we have that, by the Time Hierarchy Theorem, TIME($2^{n^{100}}$) $\subsetneq$ TIME($(2^{2n^{100}})^3$) $\subseteq$ EXP. This means that we can choose an L that is in TIME($(2^{2n^{100}})^3$) (and thus in EXP) that is not in TIME($2^{n^{100}}$). This proves (a) to be false.

(b) True. This is because we showed in lecture that P $\subsetneq$ EXP, which means that either P $\neq$ NP or NP $\neq$ EXP (or both).

(c) False. Since, due to the problem, we can assume that an NP-complete language has an $O(n^k)$-time algorithm for some fixed k, we have that NP = P. So then we can do the following. To prove this we must find a language L that is in P but not in TIME(n^k). To find this L, we will use the Time Hierarchy Theorem. Consider the proper complexity function $f(n) = n^k$. Then we have that $f(2n)^3 = (2n)^{3k}$. So we have that, by the Time Hierarchy Theorem, TIME(n^k) $\subsetneq$ TIME($(2n)^{3k}$) $\subseteq$ P. This means that we can choose an L that is in TIME($(2n)^{3k}$) (and thus in P) that is not in TIME(n^k). This proves (c) to be false.

(d) False. Consider the language L = Σ^*. This language is clearly in P (everything is accepted) and thus in NP (by the problem's assumption that P = NP). To be NP-complete, every language A in NP must reduce to L. But we cannot reduce any language A in NP to L because it is impossible to map no to no, since L does not have any instances of no to map to. Thus L is not NP-complete, yet is in NP. So (d) is false.

(e) True. This is because NP $\subseteq$ EXP. So since every EXP-complete language has the characteristic that every language A $\in$ EXP reduces to it, and since we have that NP $\subseteq$ EXP and thus that every language in NP is in EXP, we can conclude that every language in NP reduces to every EXP-complete language.

Problem 2, CS21 Set 5, Matt Lim

To show

$$\text{3-COLORABLE} = \{G : G \text{ is 3-colorable}\}$$

is NP-complete, we must show that it is in NP and that all NP-problems are polynomial time reducible to it. To prove the first part, we can simply give a certificate and a polynomial time verifier. The certificate is just a satisfying coloring of G; the verifier will iterate through all the edges in the graph and check to make sure that no edge has both endpoints assigned to the same color, and will also check that there are at most 3 colors used. This can clearly be done in polynomial time.

To prove the second part, we show that 3-SAT is polynomial time reducible to 3-COLORABLE. The following details the reduction.

Our reduction function f will map a 3-CNF formula φ to a graph G. The graph is detailed on the following pages:

Problem 3, CS21 Set 5, Matt Lim

To show that (3,3)-SAT is NP-complete, we must show that it is in NP and that all NP-problems are polynomial time reducible to it. To prove the first part, we can simply give a certificate and a polynomial time verifier. If φ_2 is a (3,3)-CNF formula, then the certificate is just a satisfying assignment A for φ_2, and the verifier will check that this assignment makes every clause in φ_2 true (clearly a polynomial procedure). To prove the second part, we show that 3-SAT is polynomial time reducible to (3,3)-SAT. The following details the reduction.

Our reduction function f will map a 3-CNF formula φ to a new 3-CNF formula φ_2. If φ itself has at most 3 occurrences of any variable, then we can just make φ_2 a copy of φ. For this reduction then clearly yes maps to yes and no maps to no. Else, if φ has more than 3 occurrences of any variable, we can do the following thing to each of these variables.

Let an arbitrary variable r repeat $n > 3$ times. Then we will introduce n new variables $y_1, y_2, \ldots, y_n$. We want all the y_is to be equivalent to each. That is, we want $y_1 \Leftrightarrow y_2 \Leftrightarrow \cdots \Leftrightarrow y_n$. To get this, it suffices to show that $y_1 \Rightarrow y_2 \Rightarrow \cdots \Rightarrow y_n \Rightarrow y_1$. To do this in CNF format, it will look like the following:

$$(y_2 \vee \overline{y_1}) \wedge (y_3 \vee \overline{y_2}) \wedge \ldots (y_1 \vee \overline{y_n})$$

Then, we will add the above clauses to φ (separated by $\wedge$s), and replace each ith instance of r with y_i.

Now we will show that this reduction function f maps yes to yes and no to no. First we will show it maps yes to yes. So, given that φ has a satisfying assignment, we want to show that φ_2 is satisfiable and that it is in (3,3)-SAT. To do this, we will consider our reduction function. Note that for any arbitrary variable r repeated $n > 3$ times, our reduction function adds new variables $y_1, y_2, \ldots, y_n$, each of which appear three times (twice in the added clauses (considering both y_i and $\neg y_i$), once in the replacements). Now we must consider what happens if $\neg r$ repeats $k > 3$ times as well. That is, both r and $\neg r$ repeat more than three times. If this is the case, then we can just add more y_is to our equivalence. So we will have $y_1, y_2, \ldots, y_n, y_{n+1}, y_{n+k}$. Then we will replace each $\neg r_i$ with $\neg y_{n+i}$. Note that this still gives us at most 3 occurrences of each variable. So, we have that our reduction function gives us a φ_2 that has at most 3 literals per clause and at most 3 occurrences of each variable. Now we must show that it is satisfiable to complete the proof of yes maps to yes. To prove this, we must only show that our added clauses are satisfiable, since our replacement method does not change the satisfiability of the original clauses (since all the replacements for a given repeated variable are equivalent). This is clearly true, since we can just either make all the y_i's to be true or all of them to be false. So we have that yes maps to yes.

Now we will show that this reduction function f maps no to no. To prove this, consider again the fact that our replacement method does not change the satisfiability of the original clauses (since all the replacements for a given repeated variable are equivalent). Thus, the non-added-in clauses in φ_2 are unsatisfiable. And adding in the extra clauses can do nothing to change this, since all the clauses are separated by $\wedge$s. So we have that no maps to no.

Now we must only prove that our reduction function f is polynomial time computable. A variable can repeat no more than $3n$ times, where n is the number of clauses. And there can be no more than $3n$ repeated variables. This means that for each repeated variable, we do no more than $3n$ replacements, and add no more than $3n + 1$ clauses. Clearly the function is polynomial. So we have that our reduction function is polynomial.

Finally, we can conclude that (3,3)-SAT is NP-complete.

Problem 4, CS21 Set 5, Matt Lim

To show that MAX2SAT is NP-complete, we must show that it is in NP and that all NP-problems are polynomial time reducible to it. To prove the first part, we can simply give a certificate and a polynomial time verifier. If ϕ is a 2-CNF formula, then the certificate is just an assignment A that simultaneously satisfies at least k clauses of ϕ. The verifier will check that this assignment does indeed satisfy at least k clauses of ϕ, a procedure that is clearly polynomial. To prove the second part, we show that 3-SAT is polynomial time reducible to MAX2SAT. The following details the reduction.

Our reduction function f will map a 3-CNF formula φ to a 2-CNF formula ϕ. It will do this in the following way. For each clause in φ, it will construct 10 new clauses with 2 literals. Let us consider a single clause $(x, y, z) \in \varphi$. We will map this clause to the 10 new 2-literal clauses shown below:

$$(x \vee x), (y \vee y), (z \vee z), (w \vee w),$$

$$(\neg x \vee \neg y), (\neg y \vee \neg z), (\neg z \vee \neg x),$$

$$(x \vee \neg w), (y \vee \neg w), (z \vee \neg w)$$

Consider the truth table for these 10 clauses, as shown below, letting 1 represent true, 0 represent false, and max be the max number of clauses that are simultaneously satisfiable (depending on what w is assigned):

x	y	z	max
1	1	1	7
1	0	0	7
0	1	0	7
0	0	1	7
1	1	0	7
0	1	1	7
1	0	1	7
0	0	0	6

Note that the maximum number of clauses that can be satisfied here is 7. Also note that this happens exactly when $(x \vee y \vee z)$ is true. So, this is our reduction function.

Now we will show that this reduction function f maps yes to yes and no to no. First we will show yes maps to yes. So, given that φ has a satisfying assignment, we want to show that ϕ is a 2-CNF formula for which it is possible to simultaneously satisfy at least k clauses. So, let $k = 7n$, where n is the number of clauses in φ. Since φ has a satisfying assignment, this means that at least one literal in each clause is true. That is, each arbitrary clause $(x \vee y \vee z) \in \varphi$ is true. This, given our reduction function f, means that for every 10 2-literal clauses generated from each 3-literal clause in φ, 7 of them are true. This then means that there are $7n$ simultaneously satisfiable clauses in ϕ. But this means that at least k clauses can be simultaneously satisfied in ϕ. So ϕ is in MAX2SAT, and we have that yes maps to yes.

Now we will show that this reduction function f maps no to no. To do this, it suffices to show that if ϕ is in MAX2SAT, then φ is satisfiable. So, consider our ϕ. More specifically, consider each set of 10 clauses in ϕ that were generated from one clause in φ. A max of 7 clauses can be made true from each set. This means that, for ϕ to be in MAX2SAT ($k = 7n$ true clauses in this case), 7 clauses in each set of 10 must be true. But this only happens when each clause that generates each set of 10 2-literal clauses is true; that is, when each arbitrary $(x \vee y \vee z) \in \varphi$ is true. In other words, we have that, for every clause in φ that generates each 10 clause set in ϕ, at least one literal is true. That is, we have that every clause in φ can be made true. And we will never run into the issue that x and $\neg x$ are both true for some arbitrary x, because this does not occur for an instance of MAX2SAT. So we have that we can construct a satisfying assignment for φ, and thus that no maps to no.

Now we must only prove that our reduction function f is polynomial time computable. Our function is clearly polynomial, because for each clause in φ, it simply generates 10 clauses to put in ϕ. So we have that our reduction function is polynomial.

Finally, we can conclude that MAX2SAT is NP-complete.