Problem 1, CS21 Set 4, Matt Lim

We will show that the language

$$\text{2-COLORABLE} = \{G : G \text{ is 2-colorable}\}$$

is in P by reducing it to 2-SAT, which is in P.

Consider a graph G. We first show that there is a reduction function f that maps G to a set of two-tuples in polynomial time. Let graph G be represented as (V, E), where V is a set of vertices $\{V_1, V_2, \cdots\}$ represented by literals $\{x_1, x_2, \cdots\}$ (we can clearly do this mapping in polynomial time) and E is a set of edges, which are represented as two two-tuples of literals (x_i, x_{i+1}) and $(\neg x_i, \neg x_{i+1})$. We can see that there is a function $f((V, E)) \to S$, where S is a set of two-tuples, that is polynomial in time. This is true because with our graph, we can simply iterate through all the edges and put both two-tuples that each edge represents in S, a process which is clearly polynomial. So we have that our reduction function f is polynomial.

Now we must prove two statements. First we must show that if G is in 2-COLORABLE, then $f(G) = S$ is in 2-SAT. We will prove this by contradiction. Assume $G \in$ 2-COLORABLE. Then we have that $f(G) = S$, where S is a set of two-tuples. Now assume, to the contrary, that $S \notin$ 2-SAT. Let S' be the directed graph derived from S. Then we have that, since $S \notin$ 2-SAT, that there exists a path $x \implies {}^*\neg x \implies {}^*x$. This means there exists a path of the following form:

$$(x_1, \neg x_2)(\neg x_2, x_3) \cdots (x_n, \neg x_1) \cdots$$

We will assert that this means there is a path of edges that connects $(x_1, x_2, \cdots, x_n, x_1)$. To see this, consider the following: $(x_n, \neg x_1) \in S' \implies (\neg x_1, \neg x_n) \in S \implies (x_1, x_n) \in S \implies (x_1, x_n) \in E$. This argument can be applied for all the other pairs in the above path. This gives us a path from x_1 to x_2 to $\ldots x_n$ to x_1 in G. Observe that because of the way our path was written above, n is odd. Then we have that there is a cyclic path from x_1 back to itself with an odd number of edges and vertices. This means that, if we were to assign a color to each vertex, a color would have to repeat itself. That is, an edge would have both endpoints assigned to the same color. This means that G is not in 2-COLORABLE. This is a contradiction. So we have that if G is in 2-COLORABLE, then $f(G) = S$ is in 2-SAT.

Second, we must show that if G is not in 2-COLORABLE, then $f(G) = S$ is not in 2-SAT. So, assume G is not in 2-COLORABLE. Then we have that there is a path $x_1 \ldots x_n x_1$ in G, as shown above (else G would be in 2-COLORABLE). Using the reverse of the logic above, this means that we have a path in S', where S' is the directed graph derived from S, of the following form:

$$x_1 \to \neg x_2 \to x_3 \to \neg x_4 \to \cdots \to x_n \to \neg x_1 \to x_2 \to \neg x_3 \to \cdots \to \neg x_n \to x_1$$

That is, we have that there exists a path $x \implies {}^*\neg x \implies {}^*x$ in our derived S'. So by definition, we get that $f(G) = S$ is not in 2-SAT.

So we have reduced 2-COLORABLE to a problem known to be in P with a reduction function f that runs in polynomial time. Thus, 2-COLORABLE is in P.

Problem 2, CS21 Set 4, Matt Lim

We will prove that for every H, CONTAINS$_H$ is in P. Our algorithm is as follows. Let x be the number of vertices in H and y be the number of vertices in G. Then we will consider all the yCx ways that x vertices can be chosen in G. Note that if H contains more vertices then G, then yCx will be 0 and we will reject. If H does not contain more vertices than G, then we will continue on with the algorithm. So, we will consider all the yCx lists of x vertices in G. Generating the set of these lists is $O(y^x)$ time, which is polynomial. Then, for each list of vertices, we will see if the graph they generate, using the edges in G, is isomorphic to H. We can generate this graph in polynomial time in the following way. We can iterate over the list of x vertices once, and for each vertex, we can check each edge in G and see if it contains that vertex. If it does, we can mark that vertex in that edge. Then, once all the vertices have been iterated over, we can iterate over all the edges and pick the ones with two marked vertices to make up the edges in our subgraph. This is clearly in polynomial time, since the number of vertices x is independent of the length of the input and the number of edges in G are bounded by the number y^2 (it is $O(xy^2)$). We can then check if the graph we generated in polynomial time is isomorphic to H by comparing every mapping (brute force). If one of the graphs we generate is isomorphic to H we will accept, else we will reject. Since this brute force algorithm is dependent on the number of vertices in H, it runs in constant time in relation to the length of the input G. So it runs in polynomial time. Then we have an algorithm that decides CONTAINS$_H$ in polynomial time. And by the Extended Church-Turing Thesis, then this algorithm runs in polynomial time on a Turing Machine. So CONTAINS$_H$ is in P.

Problem 3, CS21 Set 4, Matt Lim

To prove that the problem UNARY SUBSET SUM is in P, we will do three things. First, we will provide an algorithm that solves this problem. Second, we will show that this algorithm runs in polynomial time. Third, we will prove this algorithm is correct. Let INPUT be the input symbols that excludes the first input value 1^B.

First, we will provide an algorithm that solves this problem. The algorithm will work as follows. First, we will iterate through the first input symbol to check that it is of the form 1^B. If it is, we will iterate through all the B ones in the first input symbol 1^B in order to get a value for B. If it is not, we will reject. We will then iterate through the rest of the input symbols and check that they are all positive integers. If they are, then we will keep going through the following algorithm. If they are not, then we will reject. This process is clearly in polynomial time. We will then have a table with $B+1$ boolean values. The values will be labeled $0, 1, 2, \ldots, B$. We will initialize this table so that the 0th boolean value is true and the rest are false. We will then iterate through each boolean value of the table in order, starting from 0. Every time a true value is encountered, we will add each number $x_i \in$ INPUT to the label of that true boolean. Let this sum be called S. If $S \leq B$, we will set the value of the Sth boolean value to be true. We will do this until the Bth boolean is set to be true, at which point we know that our input is in UNARY SUBSET SUM and we can accept, or until we have iterated through the whole table, in which case we know that our input is not in UNARY SUBSET SUM and we can reject. This part of the algorithm (the part that doesn't involve counting the B ones) also runs in polynomial time. We can see that this is true by considering the worst case scenario. The worst case scenario is that every value from 0 to B is in INPUT. In this case, we would have to iterate through all B booleans, which would all be true, and for each kth boolean, we would have to iterate through all n integers in INPUT and check if the sum of the label of the kth boolean and the input integer is less than or equal to B. So this would be $O(Bn)$, which is polynomial. Then we have that this algorithm runs in polynomial time. And by the Extended Church-Turing Thesis, then this algorithm runs in polynomial time on a Turing Machine.

So we have shown an algorithm that solves this problem in polynomial time on a Turing Machine. Now all that is left is to prove that this algorithm does indeed solve our problem. That is, we want to show that for each value B, given an arbitrary input, we will get the correct answer for whether or not the input is in UNARY SUBSET SUM. To do this, we will use induction on B. First consider the base case, when $B = 0$. In this case, it is clear to see that our algorithm properly solves the problem, because it will accept any input of the correct format. This is correct because with any input of the correct format, you can take I to be the empty set, in which case you will get that $0 = 0$. Note that in this case, we reject any inputs of the wrong format. Now, onto the inductive assumption. Assume that we get the correct value for $B = n$, where $0 \leq n \leq k$. Using this assumption, we must show that we get the correct value for $B = k + 1$. So, let's begin. We will consider an arbitrary input of the correct format called M. If the input is not of the correct format, we will just reject. Consider the case when $B = k + 1$. We know that for $B = n$, as n is described above, that we get the correct result for B. We have that the case where $B = k + 1$ is only solvable iff there exists an i such that $B = (k+1) - x_i$ is solvable, where $x_i \in$ INPUT. And we can get a correct value for $B = (k+1) - x_i$ by our inductive assumption. Also, if this value is negative for a given i, then for that i $B = (k+1) - x_i$ is not solvable. So we can get the correct value for $B = k+1$. Then we have that for any arbitrary input, our polynomial algorithm works for all values of B. So UNARY SUBSET SUM is in P.