Problem 1

(a) Here is our algorithm:

1. Find the cost of the min cut of the original flow network. This can be done by using Ford Fulkerson to find the max flow, then seeing how much flow goes out of the source node to get the size of the max flow, then applying the max flow min cut theorem. Let this cost be C_0.
2. Remove e from the flow network. If e does not disconnect the networ, finish this step. Else, proceed to the next step. So we have that e does not disconnect the network, since we are still in this step. Now find the cost of the min cut for the new flow network. Let this cost be C_1. If $|C_0 - C_1| = c(e)$, where $c(e)$ is the capacity of e, return true. Else return false.
3. Since we are here, we have that removing e from the flow network disconnects it. Let $e = (a, b)$. Let a be the vertex that is still connected to the source s (or is the source) and let b be the vertex that is still connected to the sink t (or is the sink). Then we will connect a to a new sink t' and connect b to a new source s'. Both edge that make these connections will have infinite capacity. We will then find the costs of the min cuts of these two new flow networks. Call the cost of the min cut of the flow network with s and t' C_2 and call the cost of the min cut of the flow network with s' and t C_3. Then, if both $C_2 \geq C_0$ and $C_3 \geq C_0$, return true. Else, return false.

This works for the following reasons. Step 2 works because, if removing e does not disconnect the network, then we have that $|C_0 - C_1| = c(e)$ if and only if e is in at least one minimum cut of the graph. This is true because if e is in at least one minimum cut, then the equality must hold, since the original min cut still disconnects the graph, but with $c(e)$ less capacity. And if $|C_0 - C_1| = c(e)$, then e is in at least one minimum cut because the new minimum cut (with cost C_1) must be equal to the original min cut without e. Step 3 works because, if removing e disconnects the network, we need to find if we could have disconnected either side of the disconnected network with less cost. And that is exactly what we do. Thus our algorithm works.

(b) Here is our algorithm:

1. Find a cut whose size is equivalent to the size of the maximum flow (a min cut). This can be done by using Ford Fulkerson to find the max flow, then seeing how much flow goes out of the source node to get the size of the max flow, then using a BFS to find a cut whose size is equivalent to the size of the max flow. Then we can iterate over the edges and find the edges that cross the cut. Call this set of edges C.
2. Let *count* $= k$. While *count* > 0 and $|C| > 0$, greedily remove the highest capacity edge $e_{c\_max} \in C$ from C, add it to S, and decrement *count* by 1. In this case, since the capacities are all 1, we can just keep removing any edge from C and adding it to S until we have either removed k edges or there are no more edges from C to remove.
3. Return S.

Now to briefly describe why our algorithm works. We have that every edge that crosses the min cut is saturated in the max flow, and that the edges that cross the min cut form a minimum cardinality set of edges that one can remove from the flow network to disconnect it (since all the capacities are 1). So clearly removing these edges is going to decrease the size of the max flow the most. This is because if removing one of these edges didn't decrease the max flow size by 1, the max flow min cut theorem would be violated for the new flow network (the flow network with the edge removed). Then we just remove as many of these edges as we can and return that set. In this case, it doesn't even matter which edges from the min cut we choose, since they all have the same capacity, and thus removing them affects the max flow the same.

Problem 2

We want an algorithm to check whether we can remove at most k edges from G so that e is in at least one MST of the resulting graph. We have that G has to remain connected after the removal. So, set $e = (u, v)$ and $d = w(e)$. e is in at least one MST of G if in every other path between u and v there exists an edge with weight greater than or equal to d. So, we wish to remove up to k edges, so that every other path between u and v contains an edge of weight greater than or equal to d. We can do this by removing from G every $e' \in E$ with $w(e') \geq d$. Then we get a problem that was solved in lecture 14: finding the minimum subset of edges such that its removal disconnects u and v. Call this minimum subset S. If $|S| \leq k$ and the graph G' obtained by removing S from the original graph G is still connected, then S is our desired set of edges. This is because by removing S, we have that the resulting graph is still connected (so an MST can be formed) and that on every path from u to v there exists an edge with weight greater than or equal to d, since we disconnected the paths that had edges all with weight less than d. If not, then we cannot remove at most k edges from G so that e is in at least one MST of the resulting graph, because we know that it is impossible to disconnect all paths from u to v that are made up of edges that all have weight less than d by removing at most k edges while still keeping the resulting graph connected.

Problem 3

Here is how we do this. We build a flow network (V, E, s, t, c). Here is how we structure it. For each producer we have a vertex v_{p_i} that represents the ith producer (so we have m of these vertices). The capacity of the edges from s to each v_{p_i} are of capacity 0. For each producer we also have a vertex v'_{p_i}. We connect each v_{p_i} to the respective v'_{p_i} (so connect v_{p_1} to v'_{p_1}, v_{p_2} to v'_{p_2}, etc.). The capacity of the edges that make these connections be ∞. Then for each actor we have a vertex v_{a_i} that represents the ith actor (so we have n of these vertices). Then we connect each vertex v_{p_i} to every v_{a_i} where the ith actor is one of the ith producer's favorite actors. That is, we basically connect the producers to their favorite actors. The capacity of the edges that make these connections is ∞. Then, for each vertex v'_{p_i}, we connect it to vertex t. The capacity of the edges that make these connections is $-X_i$. Then, for each vertex v_{a_i}, we connect it to vertex t. The capacity of the edges that make these connections is Y_i. Then our setup is done.

Now all we need to do is find the edges that cross the min cut of this graph. This can be done in the following way. Find a cut whose size is equivalent to the size of the maximum flow (a min cut). This can be done by using Ford Fulkerson to find the max flow, then seeing how much flow goes out of the source node to get the size of the max flow, then using a BFS to find a cut whose size is equivalent to the size of the max flow. Then we can iterate over the edges and find the edges that cross the cut. Call this set of edges C. Then we can just iterate over the edges in C, find all vertices of the form v'_{p_i} contained in these edges, and add those is to a list S. Then Victor, Henry, and Adam can just iterate through the indices contained in S and choose those producers. That is, if S contains the indices $x_1, x_2, \cdots, x_k$, Victor, Henry, and Adam should choose the x_1th, x_2th,..., x_kth producers.

Now we will briefly explain why this works. We basically have that representing this problem as our graph gives us the following. If the actors that a producer p_i wants are too expensive, taking into account the fact that other producers may want those actors as well, then we can just separate s and v_{p_i} in our cut with zero cost. Else, we will include the producer since we will cut along the edges with costs of X_is and Y_is (since the investment of the producer is more than the cost of the actors in this case, the cost of this part of the cut will be negative), which include v'_{p_i}. We also have that our algorithm takes into account actor overlap; that is, if multiple producers want the same actor, we only include that cost once. So clearly, our algorithm functions as intended.

Problem 4

(a) We will prove this is true by induction on the size of odd length cycles. Our base case will be cycles of length 5. To prove this, consider an arbitrary cycle $(x_1x_2x_3x_4x_5)$ of length 5. We have that this is just the same as the composition $(x_1x_2x_3)(x_3x_4x_5)$. So the base case is satisfied. Now assume that any odd cycle of length k can be expressed as a composition of cycles of length 3. Now we must show that odd cycles of length $k+2$ can be expressed as a composition of cycles of length 3. That is, we want to show that cycles of the form $(x_1x_2\cdots x_{k+2})$ can be expressed as a composition of cycles of length 3. We have that $(x_1x_2\cdots x_{k+2})$ can be expressed as $(x_1x_2\cdots x_k)(x_kx_{k+1}x_{k+2})$. And due to our inductive assumption $(x_1x_2\cdots x_k)$ can be expressed as a combination of cycles of length 3. So we can conclude that odd cycles of length $k+2$ can be expressed as a composition of cycles of length 3, and our inductive proof is complete.

(b) We have that every decomposition of an even permutation α into transpositions consists of an even number of elements. So all we need to do is to show that every pair of transpositions can be turned into a composition of cycles of length three. So, consider the arbitrary transposition pair $(x_1x_2)(x_3x_4)$. We have that

$$(x_1x_2)(x_3x_4) = (x_1x_4x_3)(x_1x_2x_3)$$

We can see that this is true because the composition on the left effectively swaps x_1 with x_2 and x_3 with x_4. So now we can just apply this equality to every pair of transpositions in the decomposition of α. So clearly, it is true that every even transposition can be expressed as a composition of cycles of length three.

Problem 5

We want to prove that if a permutation $\pi \in S_n$ (for $n \geq 3$) satisfies $\pi\tau = \tau\pi$ for every transposition $\tau \in S_n$, then $\pi = \text{id}$. To prove this, we will prove the contrapositive; that is, if $\pi \neq \text{id}$, then there exists a $\tau \in S_n$ for which $\tau\pi\tau^{-1} \neq \pi$ for $\pi \in S_n$, $n \geq 3$. So, consider an arbitrary $\pi \in S_n$ for which $n \geq 3$, $\pi = (x_1x_2\cdots x_k)$. Then we will choose $\tau = (x_1x_k) \implies \tau^{-1} \implies (x_kx_1)$. Then we have that applying τ^{-1}, then π, then τ to some arbitrary sequence of numbers $\cdots x_1 \cdots x_k \cdots$ results in the following.

$$\text{applying } \tau^{-1} \implies \cdots x_k \cdots x_1 \cdots$$

$$\text{applying } \pi \implies \cdots x_1 \cdots x_2 \cdots$$

$$\text{applying } \tau \implies \cdots x_k \cdots x_2 \cdots$$

But we have that just applying π to that same sequence of numbers results in the following.

$$\text{applying just } \pi \implies \cdots x_2 \cdots x_1 \cdots$$

So clearly, we have that $\tau\pi\tau^{-1} \neq \pi$. Thus we have proved the contrapositive and thus proved the original statement.