Problem 1. First, we can transform any bounded linear program into standard form $\min cx | Ax = b, x \geq 0$ in polynomial time. Then, since the feasible point is not a basic feasible point, there are less than n linearly independent tight constraints. Similar to the method showed in class, let us choose any of the non-tight linearly independent constraints, and examine the corresponding line through x that is feasible, i.e. $x \pm \epsilon d$ where d is the direction of the line. For small enough ϵ, there is a direction in which $c(x \pm \epsilon d)$ is strictly non-increasing (otherwise x would be a vertex), so we can increment/decrement the value of x_i in this direction until another constraint becomes tight, at which point we stop. The stopping point will therefore have one more tight constraint that is also linearly independent. If we repeat this process until there are n linearly independent tight constraints, then we will have reached a basic feasible solution. Since we repeat this process a maximum of n times for n linearly independent constraints given a feasible point, the algorithm runs in polynomial time.

Problem 2. (a) We can define an auxiliary LP as follows: for every constraint in the original LP, we create a new constraint in the auxiliary LP $A_i x + y_i = b$, where A_i represents the i-th row of A and y_i represents a new slack variable, with an objective function $w = \min y$ such that $y \geq 0$. In other words, we create a new LP with an additional set of variables y_i representing the slack in each constraint, and seek to minimize the sum of the slack variables $y_i \geq 0$ for all y_i. Then the optima of this new LP are exactly the feasible points of the original LP, and additionally we have that the new LP is unbounded (since $\min y$ s.t. $y \geq 0 = 0$), and if the new LP has some assignment y such that $\exists y_i < 0$, then it follows that the original LP is infeasible. Thus, we can assume some feasible assignment y such that $y_i \geq 0$ for all y_i, and the obvious feasible point of this is $y = \mathbf{0}$, corresponding to a feasible point of the original LP.

(b) We can transform any LP into standard form and determine a feasible point using the argument from part (a), giving us a feasible starting point to input to the simplex solver. Thus, we can use the simplex solver to solve any arbitrary LP problem.

Problem 3. (a) Finding a common point within the two polyhedra P and Q corresponds to solving the following linear program:

$$\max \mathbf{0}x \qquad \text{s.t. } \begin{bmatrix} A \\ D \end{bmatrix} x \leq \begin{bmatrix} b \\ e \end{bmatrix}$$

If the intersection of P and Q is empty, then this linear program is infeasible. Then, this implies that the corresponding dual,

$$\min \begin{bmatrix} b \\ e \end{bmatrix}^\top y \qquad \text{s.t. } \begin{bmatrix} A^\top D^\top \end{bmatrix} y = \mathbf{0},\ y \geq 0$$

is unbounded. If we instead write y as the concatenation of two vectors $y = \begin{bmatrix} y \\ z \end{bmatrix}$, then the first constraint of the dual implies

$$A^\top y + D^\top z = y^\top A + z^\top D = \mathbf{0} \tag{1}$$

and the unbounded optimum implies

$$\begin{bmatrix} b \\ e \end{bmatrix}^\top y = y^\top b + z^\top e < 0 \tag{2}$$

Thus, there is some combination of vectors y, z such that (1) and (2) are satisfied.

(b) Let $c^\top = y^\top A$. From (1) above, we also have $c^\top = -z^\top D$. Then for all $x \in P$ and $w \in Q$, we have

$$\begin{aligned} c^\top x < c^\top w &\iff (y^\top A)x < (-z^\top D)w \\ &\iff y^\top (Ax) + z^\top (Dw) < 0 \\ &\iff y^\top b + z^\top e < 0 \end{aligned}$$

since $Ax \leq b$ and $Dw \leq e$ as defined by the polyhedra. Thus, as shown in part (a), if P and Q have an empty intersection, then $y^\top b + z^\top e < 0$, and it follows that $c^\top x < c^\top w$ for all $x \in P$ and $w \in Q$. Therefore, there is a separating hyperplane for P and Q.

(c) Whether or not the two polyhedra have a point in common is determined by whether there is a separating hyperplane for P and Q, which can be easily verified using the result from part (b), i.e. if there exists a pair of vectors $y, z \geq 0$ satisfying $y^\top A + z^\top D = \mathbf{0}$ and $y^\top b + z^\top e < 0$, then this corresponds to P and Q having an empty intersection.

Otherwise, if there is no such pair of vectors y, z, then the two polyhedra have at least one point in common, which can be verified by plugging in the coordinates to

$$P = \{x | Ax \leq b\} \text{ and } Q = \{x | Dx \leq e\}$$

Problem 4.

Problem 5. (a) Given the optimal diameter d, we can devise a greedy 2-approximation algorithm as follows:

- Pick any point p.
- Create a new cluster centered at p, and add all points within a d-radius to this cluster (which has diameter $2d$).
- Repeat this process for any unassigned points until there are no such points remaining.

Note that in the original problem, the optimal solution will divide the points into k clusters of diameter at most d, e.g. $c_1, c_2, \ldots, c_k$. Given any point $p_i \in c_j$, let us assume the worst case, which is that this point is on the boundary of the cluster (i.e. p_i is exactly $d/2$ units away from the center of c_j). Then our greedy algorithm will choose p_i to be the center of a new cluster c'_j with diameter $2d$, and any point in c_j will also be included in c'_j (since the furthest possible point $p_k \in c_j$ will be located d units away from p, which is within the limits of c'_j).

Thus, we are guaranteed that for each iteration of our greedy algorithm, a new cluster c'_j will be created that entirely encompasses a cluster c_j in the optimal solution. Since there are k clusters in the optimal solution, our greedy algorithm will produce k clusters of diameter at most $2d$, making it a valid 2-approximation algorithm.

(b) We know that in the optimal solution, any two points with a distance greater than d belong to two different clusters. Thus, it follows that after k iterations of the algorithm, in which each iteration chooses the point furthest away from all the previously chosen centers, we will have k clusters such that no point in any cluster is more than d units away from the corresponding center. Thus, the algorithm produces k clusters with diameter at most $2d$, making it a valid 2-approximation algorithm.