A4

Problem 1

Part A

Let p be the original problem of finding the minimum value of b and d be the decision problem given some value b'. We run d for increasing values of b', starting at 1 until the decision problem returns true. Then the minimum value of b is the smallest value of b' for which d returns true.

Part B

b = n is a trivial solution for the decision problem, as assigning a base to every planet possible will ensure there is at least one base on a node at each edge. Thus, the minimum value of b ≤ n. Therefore, assuming the decision problem, d, is correct as b must be an integer, testing every possible value of b = b' for b' = 1, 2, ..., n ensures that firstly, a solution will be found (as b = n is a trivial solution) and secondly the minimum will be found as every possible solution is checked incrementally from the minimum.

This process involves O(n) calls to d. At each value of b' we run d without any modifications to the graph and then either return true or check the next value. Thus, if d has time complexity O(p(n)) then the reduction has time complexity O(nd(n)).

Part C

Let CERT(G, B, b) be a certifier for the decision problem where G = (V, E) is a graph, B is the proposed set of vertices and b is the size limit of B.

CERT must check the following:

1. $\forall v \in B : v \in V$
2. $|B| \leq b$
3. $\forall v \in V \backslash B, \exists u \in B : (u, v) \in E$

The first can be done by a linear search through V for each vertex in B. There are at most n vertices in V and thus this check will run O(n^2) time.

The second can be done in O(n) time by counting the number of vertices in B and checking if it is less than b.

The last can be done by first finding the set S = V \ B. Then for each vertex u $\in$ B, check each of its edges e = (u, v). If V is in S then remove it and continue. If by the end S is empty, the third condition is met. Finding the set S can be done in O(n^2) time by a linear search through B for each vertex in V. Then, there are at most n^2 edges out of B (by the handshake lemma), at each we iterate through O(n) edges to check if the edge is in S. Thus it can be done in O(n^3) time.

Therefore, we can certify a solution in polynomial time and hence, the decision problem is in NP.

Part D

As the decision problem, d is in NP we now show that the set-cover problem, which is NP complete, reduces to it. That is, we prove set-cover ≤$_p$ d.

The set cover problem asks given:

- A set U of integers
- A set of sets S = {S$_1$, S$_2$, ..., S$_n$: S$_i$ $\subseteq$ U $\forall 1 \leq i \leq n$}

- An integer k
  Is there at most k sets in S whose union equals U?

To reduce to d, create a graph G with the following:

1. Create a node u$_i$ for each number in the set U (U layer of graph)
2. For each S$_i$ in S:
   a. create a node s$_i$ (S layer of graph)
   b. for each number c in S$_i$ create a forward edge from s$_i$ to u$_c$
3. Then connect every node in the s layer with an edge in either direction

Under this construction, the U layer of the graph represents the original set U and the S layer represents which numbers in U each subset contains. Further, the only forward edges available are those from a node in the S layer to one in the u layer or another node in the S layer. Thus, the only feasible additions to the set B in problem d are those in the S layer (adding a node in the U layer is equivalent to not taking any node as there are no edges out of that node). In doing so, the nodes added to the set B in d represent the collection of sets in S chosen in the set cover problem. Additionally, connecting all the nodes in the S layer ensures that any node that doesn't satisfy the condition must be in the U layer.

Now, for a random subset of nodes in the S layer, S', the union of the sets before the construction is represented exactly by the collection of nodes in the U layer that have an edge from S' as an edge is present if and only if the number represented by U was in one of the sets represented by a node in S'. Therefore, there exists a set S' of nodes where |S'| ≤ b if and only if there exists a set cover of size ≤ k in the original problem.

Part E

By part D: set-cover ≤$_p$ Decision problem d
By part A: d ≤$_p$ Original search problem

Thus, the original problem is NP hard as the set cover problem reduced to it. However, as only decision problems can be NP complete, the search problem is not NP complete.

Problem 2

Part A

Part B

Is there a choice of exactly one planet l_i from each quadrant Q_i such that $\Sigma_{1=i}^{k} c_{l_i} \leq D \: and \: \Sigma_{i=0}^{k} \delta_{l_i} = M$ for some integer M and D.

Part C

For set of quadrants Q, set of chosen planets P, max cost D and max distance M define Cert(Q, P, D, M) which must check the following:

1. For each quadrant in Q there is exactly 1 planet in P
2. The sum of costs of planets in P is less than D
3. The sum of distances of planets in P is equal to M

The first can be done in $O(k^2)$ time by first iterating through the quadrants in Q for each planet in P and checking each planet is assigned to a quadrant in Q. Then the same is for each quadrant in Q checking there is an assigned planet in P. If each planet in P has a quadrant in Q and each quadrant Q has a planet in P then there is exactly 1 planet chosen from each quadrant as |Q| = |P| = k.

The last two can be done in linear time by summing the cost and distances of each planet in P and checking if they are less than D and equal to M respectively.

Thus, there exists a polynomial time certifier, and thus the decision problem is in NP.

Part D

Consider the subset sum problem (SSS):
Given a set of positive integers S, and an integer t, is there a subset of numbers in S that sum to exactly t?

We reduce from SSS(S, t) to the quadrant decision problem from part B, QDP(Q, D, M). To begin, we create an instance of QDP from an instance of SSS. For each number s in S, create a quadrant in Q containing two planets. The first with δ = s and c = 0 and the second with δ = 0 and c = 0. Then we run the QDP using the constructed quadrants, D = 0 and M = t which returns true if and only if there is a valid subset sum.

Proof SSS = True $\implies$ QDP = True:
Assume there is a valid subset of S that sums to t. After the reduction, each integer s$_i$ in S is mapped to exactly one quadrant Q$_i$ in Q with Q$_i$ = {p$_1^i$ = (δ = s$_i$, c = 0), p$_0^i$ = (δ = 0, c = 0)}. Under this construction, selecting p$_1^i$ is equivalent to adding s$_i$ to the subset in the SSS while choosing p$_0^i$ is equivalent to not selecting it. As there is a valid SSS, there must be a valid subset P$_1$ = $\{p_1^i \mid 1 \leq i \leq |S|\}$ with sum M (as M = t). As these are in separate quadrants, choosing one will not restrict the ability to choose another. Set P$_0$ = $\{p_0^i \mid p_i^i \notin P_1\}$ with total distance = 0. Then for $P = P_1 \cup P_2$ the sum of all the distances will be (sum of the distances in P$_1$) + (sum of distances in P$_0$) = M. Further, as all planets have cost 0, the total cost of all planets in p will also be 0 satisfying total cost ≤ D = 0. As each quadrant has exactly one planet, the QDP will return true.

Proof QDP = True $\implies$ SSS = True:
If the quadrant decision problem returns True, then there exists a set P of exactly one planet from each quadrant whose costs sum to less than 0 and distances sum to exactly M. As each quadrant contains only planets with costs 0, the first restriction is met. Further, each quadrant contains a planet with zero distance and a planet with nonzero distance. Take all the planets in P with nonzero distance. As there is a direct mapping of numbers in the original set to quadrants in QDP, each planet with a nonzero distance corresponds to exactly one unique number in S. Thus there must exist a subset of numbers in S that sum to t (as t = M).

Part E

We have a decision problem QDP that is NP complete.
The goal is to minimize $\sum_{i=1}^{k} \delta_{l_i}$ while keeping the total cost $\sum_{i=1}^{k} c_{l_i}$ ≤ D.

First, we iterate through the quadrants and find the planet with the max distance in each and calculate the sum of these. That is, find $\Delta = \sum_{i=0}^{k} \delta_{l_i} \text{ where } \delta_{l_i} = \max_{l \in Q_i}\{\delta_l\}$. Now, any selection of one planet from each quadrant must have a sum of distances less than Δ. This can be done in one iteration through the n planets, using a temporary max variable for each quadrant and adding it to the total. Thus, this step can be done in polynomial time.

Then we modify the QDP to QDP' such that instead of returning a boolean indicating the existence of a satisfying subset, it will return the subset itself or none if no such subset exists.

Now, we can solve the original problem by calling QDP' with the maximum cost D and summed total distance M starting at 0 and increasing by 1 each time. Any call to QDP' that returns a subset of planets terminates the algorithm, and that subset minimizes the summed total distance. Otherwise, return the result of the call with M = Δ as either this will find a valid subset, or none exist. This requires Δ + 1 calls to a polynomial time algorithm where $\Delta + 1$ is some constant.

Thus, the algorithm can be solved using a polynomial number of steps and a polynomial number of calls to a polynomial time decision algorithm.

Hence, the original problem is NP hard.