1: Problem 1: Identify Computational Problems

a. Since we want to divide up the forest into several niches characterized by distinct distributions of plant species, we can pose this as a Maximum k-cut problem.
Input: The input will be a graph $G = (V, E)$. Each vertex in V represents a plot in the forest. An edge between vertices v_i, v_j has weight which is the maximum difference amongst niches present in these plots.
Output: The output will be k number of disjointed set of vertices which will represent ecological niches made up of distinct distribution of plants.
Objective: The objective is to maximize the edge weights between vertices lying in different partitions which translates into making sure that our partition into disjoint sets ends up clustering plots on the basis of niches.

b. To identify the most likely sequences of neuron-neuron signaling to trigger each response in a distinct part of the network to a given stimulus, we can model it as the shortest path problem.
Input: The input will be a directed and weighted graph G = (V, E). The vertices $v_1, v_2, \cdots, v_n$will represent neurons $N_1, N_2, \cdots, N_n$ and the edges $e_1, e_2, \cdots$ will represent synapses between neurons. The edge weight between any pair of vertices v_i and v_j will represent the ($-1\times$log(frequency)) with which the synapse between neurons n_i and n_j is used. Modifying the edge weights into the -log(f) ensures that a path with a synapse of no probability of firing does not get selected. There will be a source neuron $s \in V$ where stimulus will be provided and a target neuron $t \in V$ where the response will be observed for the given stimulus.
Output: The output will be a path $s, v_1, v_2, \cdots, v_n, t$ such that $e_0 = (s, v_1)\ \epsilon E$, $e_i = (v_i, v_{i+1})\epsilon E$ for all i, and $e_k = (v_n, t)$.
Objective: The objective is to minimize the edge weights from the source node s to the target node t. min $\sum_{i=1}^{n-1} f(v_i', v_{i+1}')$ where f = -log, $v_1' = s \to v_2' \to \cdots \to v_n' = t$ is the desired path.
In this way, we can find a path with the most likely sequences of neuron-neuron signaling for a given stimulus.

c. To determine the most efficient way to pick up all of the necessary items from the supply stations while spending as little time as possible traveling around the facility, we can pose this problem as traveling salesperson problem (TSP).
Input: The input will be an undirected graph G = (V, E), where vertices $v_1, v_2, \cdots, v_n$ will represent the supply stations $s_1, s_2, \cdots, s_n$ and edge weights represent the time $t_{i,j}$ it takes to move from one supply s_i station to the other s_j. We will assume that the workbench node (called s) is the starting as well as the end vertex.
Output: The output will be a directed cycle starting at the vertex s that represents the bench which covers all the vertices $v_1, v_2, \cdots, v_n$ representing supply stations $s_1, s_2, \cdots, s_n$ in some permutation of nodes and ends back at the vertex s.
Objective: The objective is to find the shortest path that visits each vertex (supply station) at least once and returns back to the bench while also minimizing the time taken for the entire journey.
min $\sum_i t_{a_i, a_{i+1}}$ where $s = a_0 \to a_1 \to \cdots \to a_t = s$ is the desired path

d. We can use minimum set cover to identify the smallest set of mutations needed to explain the disease assuming that everyone with the disease has at least one of the mutations in our set.
Input: The input will be a set S consisting of all the mutations $s_1, s_2, ..., s_n$, a collection C of subsets of S such that $c_1, c_2, \cdots, c_m$ are set of mutations observed in different individuals with this disease.
Output: The output will be $C' \subset C$ such that for all $s \in S$ there exists a $c \in C'$ for which $s \in c$.
Objective: Objective is to identify the smallest set of mutations to explain the disease under the assumption that everyone with the disease has at least one of the mutations in our set where we phrase the minimum set cover question iteratively, starting from $B = 1$ and increasing B to m.

e. We can use the longest common substring problem to identify the longest phrases in common across the medical reports.

Input: The input is all the sequences present across the medical reports.
Output: The output is a suffix tree of all the phrases found in the medical reports.
Objective: The objective is to find the deepest node with more than one child to get longest common substring. In this manner we will be able to identify the longest phrases in common across medical reports.

2: Problem 2: Graph Coloring Model

a. To reduce this problem to a graph coloring model, we can have a formal decision variant as follows: Input: An undirected graph G = (V, E). Experiments submitted to the lab can taken as a set X = $x_1, x_2, ..., x_n$. Each vertex v_i in the graph represents an experiment x_i. For a pair of experiments x_i and x_j represented by vertices v_i and v_j, we will draw an edge if they share a common piece of equipment. Objective: The objective is to find the minimum number of batches given that no two experiments that share a common piece of equipment are assigned to the same batch.

b. This pseudocode describes a greedy approach to solving the graph coloring problem. It is not a good strategy for either small number of experiments with lot of common equipment or large number of experiments with lot of common equipment. It is not reliable for either because it does not take care of the order of coloring which is important to achieve an optimal solution. Therefore, this approach lacks in achieving accuracy. It is also not efficient since it is applying a while loop which might potentially require a large number of iterations to assign colors as the number of experiments increase.

c. This pseudocode describes a brute force approach to solving the graph coloring problem. This brute force approach explores all sequences of batch assignment of the experiments and therefore it is possible to find an optimal and valid solution. It is a good strategy for a small number of experiments where we want to be as efficient as possible. Even though this approach exhaustively searches for an optimal assignment of experiments, it will remain efficient for a small number of experiments. Since the worst case scenario is that every experiment is its own batch i.e., n number of batches, this is not a reasonable strategy for a large number of experiments. Although the exhaustive search increases the chances of finding a valid solution, it will not be an efficient strategy for a large number of experiments.

3: Problem 3: RNA sequence assembly problem

a. We can model this as a decision problem using a graph model.
The input will be a directed graph G = (V,E), with a source node $s \in V$ and a target node $t \in V$. The source node s represents the sequence read of 5'UTR and the target node t represents the sequence read of 3'UTR. The vertices $v_1, v_2, \cdots, v_{k-1}$ in this graph will represent the reads $r_1, r_2, \cdots, r_{k-1}$. An edge will only be drawn between two vertices v_i and v_j if the suffix of r_i matches a prefix of r_j.
Does there exist a path of at least k overlapping reads $r_1, r_2, \cdots, r_{k-1}$ (in some order) from the source node s to the target node t in this graph G?
If such a path of overlapping reads exist, then we can join the vertices $v_1, v_2, \cdots, v_{k-1}$ (in some permuted order) to get an RNA assembly of at least k reads.

b. This problem is in NP because if given a sequence of reads $r_1, r_2, \cdots, r_{k-1}$ as a possible solution, we can verify if there is a chain of overlapping reads connecting s to t in polynomial time. We can verify if the suffix of r_i overlaps the prefix of r_j and ensure that there is sufficient overlap to put those two reads in a consecutive manner. Efficiency of such a pairwise verification is linearly dependent on the number of reads in the possible solution. Therefore, we can say that this problem is NP because it can be assigned a polynomial time verification certificate.

c. A mapping of problem inputs to inputs between the two problems (RNA sequence assembly problem and Hamiltonian Path problem):

For the Hamiltonian Path problem, the input is an undirected and unweighted graph G = (V, E) and the question is if we can find a path in the graph G that will visit every vertex exactly once. To map the input of Hamiltonian path problem to our RNA sequence assembly problem, we can have a graph G = (V, E) where each of the vertices $v_1, v_2, \cdots, v_{k-1}$ will represent reads $r_1, r_2, \cdots, r_{k-1}$. Additionally, We will have the 5'UTR as a vertex $s \in V$ and the 3'UTR as a vertex $t \in V$. And, most importantly, instead of asking if we can find a minimum of k overlapping reads between vertices $s \rightarrow t$, the question is if we can find exactly k number of overlapping reads between vertices $s \rightarrow t$ to ensure that we visit every vertex exactly once. An edge e will only be drawn between a pair of vertices v_i and v_j if there is enough overlap between the suffix of v_i and prefix of v_j.

d. Using the above mapping of inputs between the Hamiltonian path problem and the RNA sequence assembly problem in part c, we can map the answer of the Hamiltonian path problem to the answer of RNA sequence assembly problem. If we find a path that starts at the vertex s and using some permutation of other vertices $v_1, v_2, \cdots, v_{k-1}$ ends at the vertex t such that every vertex is visited exactly once, then we have found a Hamiltonian path between the vertices $s \rightarrow t$ of length k on the graph G described in part c. This also implies that we have found a path of minimum of k overlapping reads $r_1, r_2, \cdots, r_{k-1}$ between $5'UTR \rightarrow 3'UTR$ for the RNA sequence assembly problem.

e. In part b, it was proven a proposed solution for the RNA sequence assembly problem can be verified in polynomial time which suggests that the RNA sequence assembly problem is in NP. In part c and d, it was demonstrated that there is a direct mapping between inputs and outputs of the Hamiltonian path problem and the RNA sequence assembly problem. Therefore, it was shown that an instance of Hamiltonian path problem can be reduced to the RNA sequence assembly problem and hence the RNA sequence assembly problem is NP-hard. Both these parts combined, it can be claimed that the RNA sequence assembly problem is NP-complete.

f. Brute force approach is not a reasonable strategy for this problem if we need to be able to solve this problem for large read sets because of inefficiency. We can use some heuristic (greedy) approach to solve this problem with tolerance for some errors. We can apply branch and bound method or the kitchen sink technique to solve this problem for a large read sets. Additionally, in order to obtain some sub-optimal solution to the Hamiltonian path problem, we can try to find a path that does not touch every vertex exactly once or touches a few vertices more than once since we are willing to tolerate some errors.

4: Problem 4: Programming Problem

a. This problem is equivalent to the Set cover problem.
Input: The set of sequence reads R is the set S. Collection C of subsets of S is given by each strain in the genome G. B is the number of strains in G.
Question: Does there exist a subset of strains $C' \subset C$ which covers the set R?

b. Pseudocode for greedy algorithm:

```
def heuristicStep(n,m,g, subset):
  iterate over rows not in subset
  return maximum number of unaccounted columns (wrt subset) accounted for in a single row
def runGreedy(n,m,g):
  set subset to be empty
  calculate maximum number of cols accounted for in a single row using heuristicStep
  iterate over rows to find the ones that have this maximum number of unaccounted columns (wrt subset)
    add all these rows to the subset
  re-evaluate the unaccounted columns (wrt to the new subset)
  iterate the process until there is no unaccounted column or the number of iterations exceeds m
  return subset if no columns unaccounted for in the end, else return empty list.
```

c. Pseudocode for Brute Force algorithm:

```
def createSubsets(m,g):
  returns a list of all possible subsets of rows in g in increasing order of size
def bruteForce(n,m,g):
  find all the subsets for g in increasing order of size using createSubsets
  iterate over this list and check for subsets covering all the columns
    first subset in this list to cover all the columns is the smallest size solution
```

d. In the jupyter notebook

e. In the jupyter notebook

f. According to the plot generated in part e, it appears that the average solution size for the greedy algorithm (heuristic, marked with blue) is almost twice the average solution size for the brute force algorithm (marked with orange). Therefore, the plot for part e approves the conjecture that the greedy algorithm is a 2-approximation algorithm for the exact problem. However, since the graph only represents the average solution size for $m = n = 1, m = n = 2, \cdots, m = n = 15$, we cannot be entirely sure if heuristic is 2-approximation algorithm of the optimal (brute force) algorithm as there could be a different trend as teh x-axis grow s

5: Problem 5: Integer Linear Programming

a. $\sum_{j=1}^{n} y_{ij} = 1$ for all i

b. $y_{ik} + y_{jk} + x_{ij} \leq 1$ for all i, j, k

c. $\sum_{k=1}^{n} y_{ki}(1 - z_i) = 0$ for all i, k

d. min $\sum_{i=1}^{n} z_i$

e. This problem is still NP-hard because we can reduce an instance of the original graph coloring problem to this new problem. If each experiment i has a run time of $t_i = 1$, we can see that the original graph coloring problem can be reduced to this modified problem.

f. For batch k, time needed to run a batch = $\max_{i=1}^{n}(t_i y_{ik})$.
So total time = $\sum_{k=1}^{n}[\max_{i=1}^{n}(t_i y_{ik})]$.
If $z_k = [\max_{i=1}^{n}(t_i y_{ik})]$, then the objective function now is min$\sum_{k=1}^{n} z_k$