Section 1:

1. Problem formulation:
    a. glass boxes in a row, each with a lock. Each of the first five boxes holds a key unlocking the next book in line; the last box holds a fruit. You have the key to the first box, and you want the fruit
        - Initial State: Have the key to open the first box, all the boxes are locked
        - Successor/Action: Use the current key to unlock the box, retrieve the next key, move to the corresponding box that can be opened with the given key
        - Goal test: Get a banana from the last box
        - Path cost: Number of box opened during the process

    b. n x n grid of squares, each square initially being an unpainted floor. You start standing on an unpainted floor square and can either paint the square under you or move onto an adjacent unpainted floor square: You want the whole floor painted:
        - Initial State: Stand on the unpainted floor square, all the floor squares are unpainted
        - Successor/Action:
            - paint the unpainted floor square under you
            - move to the next unpainted floor square
        - Goal test: painted all the floor square
        - Path cost: Number of floor square got painted/Number of actions

    c. A container ship is in port, loaded high with containers. There are 13 rows of containers, each 13 containers wide and 5 containers tall. You control a crane that can move to any location above the ship, pick up the container under it, and move it onto the dock. You want the ship unloaded
        - Initial State: All containers are in port
        - Successor/Action: Pick a container from the ship with a crane, move it onto the doc
        - Goal Test: All the containers in ship are moved to the dock.(13*13*5)
        - Path Cost: Number of containers moved

2. The goal is to navigate a robot out of a maze. Start in the center of the maze facing north. You can turn the robot to face north, south, east, west. You can direct the robot to move forward a certain distance, although it will stop before hitting the wall. For sub-question a,b,c you have to answer with problem formulation consisting of Initial State, Goal test, Successor/Action function, Cost function which is updated to suit each sub-question.
    a. Formulate the problem. How large is the state space?
        - Initial State: Robot starts in the center of the maze and faces north.
            - In(Center), face(N)
        - Successor/Action: Direct the robot until it stops before hitting the wall, turn the robot in ¼ directions:
            - Move(forward), Turn(N/S/W/E)
        - Goal Test: Robot moves out of the maze

        - At(Exit)
    - Path cost: Number of actions/Distance moves
    - State Space: Each n position the robot can move to, there are 4 turning option => 4n states

b. In navigating a maze, the only place we need to turn is at the intersection of 2 or more corridors. Reformulate this problem using this observation. How large is the state space now:
    - Initial State: Robot still starts at the center of the maze and faces the north.
        - In(Center), face(N)
    - Successor/Action: Move forward until meeting the intersection of two or more corridors before turning: Move(forward), Turn(N/S/W/E)
    - Goal test: Robot moves out of the maze
    - Path cost: Number of actions/ Distance moves
    - State Space: We only need to turn at the intersection of corridors, and there are 4 ways to turn (n/s/e/w) => 4n
c. For each point in the maze, we can move in any of the 4 directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot's orientation?
    - Initial State: Robot still starts at the center of the maze.
        - In(center)
    - Successor/Action: From the current position, the robot will choose to move in ¼ directions. It will move until I reach a turning point (it can be a wall, intersection, or dead end).
    - Goal test: Robot moves out of the maze
        - At(Exit)
    - Path cost: Numbers of actions/Distance moves
    - In this case, I think we do not need to track the robot because the robot choose a new direction at each turning point

d. The simplification we already made:
    - Perfection in movement and position, this helps us to remove the steps to check if the robot is in the correct destination or not, or is there any problem occurs during the move
    - We simplify the need or batteries or fuels, so we do not need to actually calculate the energy need for each move
    - We also simplify the information about the maze, as the width of the path fit the robots or not, or the condition of each path

3. Search Strategies:
    a. BFS: This search will explore all the nodes in specific depth before going deeper
        - Data structure: Queue (FIFO)
        - Completeness: Yes (if b is finite)
        - Time: O(b^d)

- Space: O(b^d)
- Optimality: yes (if all step costs are equal)

b. DFS: This search will expand deepest unexpanded node
    - Data Structure: Stack (LIFO)
    - Completeness: No (it will fail in infinite-depth spaces, spaces with loops)
    - Time: O(b^m)
    - Space: O(bm)
    - Optimality: No

c. Iterative Deepening Search: This search applies DFS with depth limit
    - Data Structure: Stack
    - Completeness: Yes
    - Time: O(b^d)
    - Space O(bd)
    - Optimality: Yes (if step costs are equal)

d. A* Search: Use admissible heuristic to avoid expanding paths that are already expensive
    - Data Structure: Priority Queue
    - Completeness: yes (unless there are infinitely many node with f <= f(G))
    - Time: Exponential (depends on heuristic error)
    - Space: O(b^d)
    - Optimality: Yes (if both admissible and consistent)

We can tell that DFS is the only one that can't find optimal solution (optimality = no)
For time complexity, only A* can be exponential whereas the other stay at O(b^n)
For space complexity, DFS and IDS stands out with linear space (since IDS utilized the advantages of both BFS and DFS)
Lastly, DFS also the only one that can not guarantee to find a solution if one exist)

So depends on different scenarios:
- BFS: best for shallow solutions
- DFS: space-critical
- IDS: have both aspects of BFS and DFS
- A*: when good heuristic is available