Task2(a)

One example of a state space where iterative deepening search (IDDFS) performs much worse than depth-first search (DFS) is in a highly unbalanced tree where the goal node is located deep within a long chain of nodes. Consider a scenario where you have a binary tree, but instead of a balanced tree, it is highly unbalanced, resembling a linked list. Each node has exactly one child, forming a chain-like structure. The goal node is located at the very end of this chain. In DFS, the algorithm would traverse down the leftmost path, expanding nodes one by one until it reaches the goal node. The time complexity of DFS in this scenario is *O*(*D*), as it explores each node along the longest path. However, in IDDFS, the algorithm would repeatedly perform depth-limited searches with increasing depth limits until it finds the goal node. Since each iteration of IDDFS would traverse the entire depth of the tree up to the current depth limit, the time complexity of IDDFS would be the sum of depths explored in each iteration, which is approximately*O*(*L*^2)This is because at each iteration, IDDFS would have to traverse the entire depth of the tree, up to the current depth limit.

Task2(b)

a.Formulation of the Task as a Search Problem:

Let's denote the track pieces as nodes in a graph, where each piece is represented by its unique configuration (including flipping over if applicable). The edges between nodes represent valid connections between the track pieces. The task is then to find a sequence of track piece configurations such that:

No piece overlaps with another.

There are no loose ends where a train could run off onto the floor.

We can represent the problem as a search problem where the initial state is the set of track pieces provided, and the goal state is a valid arrangement of these pieces that satisfies the above conditions.

b. Suitable Uninformed Search Algorithm:

A suitable uninformed search algorithm for this task would be Depth-First Search (DFS). Because

the problem space is finite (as there's a finite number of ways to arrange the given track pieces), DFS will eventually explore every possible arrangement if it exists.

c. Why Removing a "Fork" Piece Makes the Problem Unsolvable:

Removing any one of the "fork" pieces would make the problem unsolvable because these pieces provide branching points in the railway track. Without them, the track becomes a linear structure with no way to diverge into multiple paths. As a result, it would be impossible to create a closed loop or a circuit that doesn't have loose ends without the ability to switch tracks using these fork pieces. Therefore, removing any one of the fork pieces would restrict the possible configurations of the railway track, making it impossible to solve the problem of creating a complete, closed railway track without loose ends.