Aim:

To implement a Hierarchical Reinforcement Learning (HRL) architecture (specifically a Manager-Worker or Goal-Conditioned model) that enables a robot to perform a complex object assembly task by decomposing it into manageable subgoals.

Algorithm:

We utilize the Option-Critic or Feudal inspired approach, which consists of two distinct policy levels:

High-Level Policy (The Manager): * Observes the global state S.

- Selects a high-level subgoal g from the available task space (e.g., "Pick Tool," "Align Part").
- Operates at a lower temporal frequency (macro-steps).

Low-Level Policy (The Worker): * Takes the current state s and the subgoal g as input

- Outputs primitive actions a (e.g., joint velocities).
- Receives intrinsic rewards for reaching the subgoal g.

Dependency Handling: Subgoals are sequenced such that Subgoal $n+1$ is only activated once Subgoal n is achieved within a distance threshold ϵ.

Code Implementation (Python):

```
import numpy as np

import matplotlib.pyplot as plt



class HRLRobotSimulation:

    def __init__(self):

        # Environment constraints

        self.state = np.array([1.0, 1.0])  # Start position

        self.assembly_steps = {

            "1. Pick Up Tool": np.array([2.0, 8.0]),

            "2. Move to Part A": np.array([5.0, 5.0]),
```

```
            "3. Align Part A to B": np.array([8.0, 2.0]),
            "4. Final Fastening": np.array([9.0, 9.0])
        }
        self.trajectory = []
        self.subgoal_reached_log = []

    def low_level_worker(self, current_pos, subgoal):
        """
        Policy: Move toward subgoal using proportional control (simulating
        a learned motor policy).
        """
        error = subgoal - current_pos
        distance = np.linalg.norm(error)

        if distance > 0:
            step_size = 0.4 # Action magnitude
            action = (error / distance) * step_size
            return current_pos + action
        return current_pos

    def high_level_manager(self):
        """
        Decomposes the main task into a sequence of subtasks.
        """
        print("--- Initiating Hierarchical Task Execution ---")
```

```
    for task_name, goal_coords in self.assembly_steps.items():
        print(f"Manager: Activating Subgoal -> {task_name}")

        # Worker attempts to reach the manager's target
        steps_taken = 0
        while np.linalg.norm(self.state - goal_coords) > 0.2:
            self.state = self.low_level_worker(self.state, goal_coords)
            self.trajectory.append(self.state.copy())
            steps_taken += 1
            if steps_taken > 100: break # Safety timeout

        self.subgoal_reached_log.append(self.state.copy())
        print(f"Worker: Subgoal reached in {steps_taken} steps.")

def visualize_hierarchy(self):
    traj = np.array(self.trajectory)
    goals = np.array(list(self.assembly_steps.values()))
    labels = list(self.assembly_steps.keys())

    plt.figure(figsize=(10, 7))

    # Plot Trajectory
    plt.plot(traj[:, 0], traj[:, 1], 'b--', alpha=0.5, label='Robot Path (Low-Level)')
```

```
        # Plot Subgoals
        plt.scatter(goals[:, 0], goals[:, 1], c='red', s=150, edgecolors='black', label='Subgoals (Manager)')

        # Annotate tasks
        for i, label in enumerate(labels):
            plt.annotate(label, (goals[i, 0]+0.2, goals[i, 1]+0.2), fontweight='bold')

        plt.title("Hierarchical Task Decomposition & Execution")
        plt.xlabel("Workspace X")
        plt.ylabel("Workspace Y")
        plt.legend()
        plt.grid(True)
        plt.show()

# Execution
robot_system = HRLRobotSimulation()
robot_system.high_level_manager()
robot_system.visualize_hierarchy()
```

Result:

Task Decomposition: The high-level manager successfully segments the assembly process into four distinct spatial subgoals.

Temporal Abstraction: The worker focuses on the local geometry of the workspace to reach a specific coordinate without needing to "know" the final assembly steps.

Efficiency: By breaking a long-horizon task (Start $\rightarrow$ Final Fastening) into smaller segments, the state space exploration is significantly reduced.

Visualization: The resulting plot shows a smooth trajectory connecting the red markers, representing the successful handover of control from one subtask level to the next.