**A) Structure of the Program**

1. **Main Method**:
   * Sets parameters such as the total number of circles, painting delay, and the list of worker counts.
   * Generates random circle coordinates.
   * Simulates the painting process for different numbers of workers and measures execution time.
2. **Generate Circles Method**:
   * Generates a list of random (x, y) coordinates representing the circle centers.
3. **Simulate Painting Method**:
   * Handles the painting process using multiple workers.
   * Ensures thread safety with a lock mechanism.
   * Assigns circles to workers dynamically and simulates the painting with a delay.

**B) Evaluation of the Tasks**

**1. Is this problem able to be parallelized?**

Yes, it’s highly parallelizable. Each circle is independent, so different workers can paint simultaneously as long as synchronization is managed properly.

**2. How would the problem be partitioned?**

The problem can be partitioned as follows:

* Each worker fetches an unpainted circle from the shared list and paints it.
* Dynamic task allocation is used, meaning workers take tasks as they complete previous ones, which ensures efficient load balancing.

**3. Are communications needed?**

Minimal communication is needed:

* Workers need to know which circles have already been painted. This is achieved using a shared data structure protected by synchronization mechanisms (e.g., locks).

**4. Are there any data dependencies?**

Yes, workers depend on the shared list of unpainted circles. Synchronization ensures no circle is painted more than once.

**5. Are there synchronization needs?**

Absolutely. Synchronization is needed to:

* Prevent multiple workers from painting the same circle simultaneously.
* Update the shared list of completed circles safely.

**6. Will load balancing be a concern?**

No, dynamic task allocation ensures good load balancing. Workers fetch tasks as they finish their current ones, avoiding idle time.

**B) Test Result**



