1.INTRODUCTION:

1.1 PROBLEM OVERVIEW:

The task is to simulate the motion of asteroids in a 2D plane, detect collisions based on their position, radius, and velocity, and output collision details. This problem is critical for space mission planning and planetary defense

1.2 OBJECTIVES:

-> Simulate asteroid motion over time.

-> Detect collisions and output them in a formatted result.

2.METHODOLOGY:

2.1 ALGORITHM OVERVIEW:

The core algorithm consists of three primary steps:

2.1.1 **Spatial Hashing:** Divide the 2D plane into a grid for faster collision detection.

2.1.2 **Position Update:** Update asteroid positions using their velocity over time.

2.1.3 **Collision Detection:** Check for overlapping distances between asteroids in neighboring grid cells

2.2 Pseudocode (High-Level):

For each time step:

For each asteroid in grid cells:

Update asteroid position

Check for collision with nearby asteroids using spatial hashing

Output collision results

3. Implementation Details:

Programming Language & Libraries Used:

- **Python 3.x**
- **Libraries:**
    - math: for distance calculation
    - concurrent.futures: for parallel collision detection (ThreadPoolExecutor)

Code Structure:

- main.py: Contains the core logic for simulation and collision detection.
- asteroids.txt: Input file with initial asteroid data.
- collisions.txt: Output file with the collision results.

- **How to Run the Code:**
- `bash`
- `python main.py`

4. Dataset Handling

4.1 Parsing asteroids.txt:

- The file contains asteroid data: ID, position (x, y), velocity (vx, vy), and radius.
- The program reads each line, processes it, and stores it in a structured format (list of tuples).

5. Output Format

5.1 Collisions.txt Format:

- The file contains collision events:
  - time_step asteroid1_id asteroid2_id

    Example:

    2.1 1 2

    2.2 1 2

6. Challenges and Solutions

6.1 Handling Small Time Steps:

- **Challenge:** Small time steps cause precision issues in distance calculation.

- **Solution:** Ensured calculations are performed with a high degree of floating-point precision and used spatial hashing to reduce unnecessary checks.

6.2 Efficient Parallelism:

- **Challenge:** Collision detection can be slow with a large number of asteroids.
- **Solution:** Used ThreadPoolExecutor to parallelize the collision detection process for each grid cell.

7. Test Case Results

7.1 Performance on Sample Test Case:

- **Test Case:** 100 asteroids, simulation for 10 seconds, grid cell size of 50 units.
- **Results:**
    - Example output:

2.1 1 2   2.2 1 2

8. Future Improvements

8.1 Optimizations:

- **Adaptive Grid Cell Size:** Dynamically adjust the spatial hash grid cell size based on average asteroid velocity and density at runtime to balance collision detection accuracy and computational load.

- **Time-Step Optimization:** Implement adaptive time steps – smaller steps during high-density collision moments, and larger steps when objects are far apart, to improve both precision and speed.

9. References

9.1 **Spactial Hashing:**
https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/

9.2 **Equation of motions :**
https://www.schoolphysics.co.uk/age14-16/Mechanics/Motion/text/Equations_of_motion/index.html

9.3 **Parallel processing :**
https://www.geeksforgeeks.org/what-is-parallel-processing/

9.4 Threadpool :
https://en.wikipedia.org/wiki/Thread_pool