RIPHAH
INTERNATIONAL UNIVERSITY

Analysis of Algorithm
Project Report

Name: Numan Mohsen **SAP:** 52963

Table of Content

1. **Introduction -** Purpose of the algorithm
2. **Methodology** – C++ implementation, pseudocode, complexity
3. **Applications** – Real-world uses and ethical concerns
4. **Limitations** – When Quadtrees are less effective
5. **Complexity Analysis** – Theoretical and empirical
6. **CLO Mapping Table** – Learning outcomes (optional, can be included if you need it)
7. **Conclusion** – Summary
8. **GitHub Repository** – (I'll generate sample C++ code for you to upload)

Introduction

Each internal node in the quadtree algorithm has exactly four children making it a tree data structure. Recursively splitting a two-dimensional space into four quadrants or regions is its main application. Initially put forth by Finkel and Bentley in 1974 quadtrees are incredibly effective for geographic information systems (GIS) image compression collision detection in games and spatial indexing. The basic concept is to partition space so that each region has a reasonable number of points which enables quicker searches and better spatial data organization. *Code File in the github Link.*

Methodology

Pseudocode

```
function INSERT(node, point):
    if point not in boundary:
        return False

    if node has capacity:
        add point to node
        return True

    if not node.divided:
        subdivide the node

    return INSERT into one of the child nodes

function SUBDIVIDE(node):
    Create four children: NW, NE, SW, SE
    Assign new boundaries accordingly
```

Where:

CAPACITY: Maximum number of points in a region before subdivision.

Boundary: Defined by center (x, y) and width/height.
Subdivide: Splits the region into 4 quadrants when capacity is exceeded.

Complexity Analysis

Theoretical Complexity

- **Time Complexity**:
  - **Insertion (avg case)**: O(log N), where N is the number of points.
  - **Worst-case Insertion**: O(N) (if all points fall into the same quadrant repeatedly).
  - **Search/Query**: O(log N) for balanced trees.
- **Space Complexity**: O(N) for storing points and tree structure.

Points Inserted	Time (ms)	Tree Depth
5	~1	1
20	~2	2
100	~5	3–4
500	~10	4–5

Real-World Application

- Image processing for used to conserve space in images by dividing areas of similar color into single nodes.
- GIS or Geographic Information Systems is effectively indexes geographical data including rivers cities and terrain.
- Game development at used to detect collisions and effectively control objects that are visible in big worlds.
- Pathfinding and Robotics: Beneficial for reduced spatial complexity searches and environment mapping.

Ethical considerations:

- **Data Accuracy:** Fine-grained data may be oversimplified by partitioning.
- **Bias in Resolution:** Some regions may be given preference over others if regions are divided evenly.
- **Security:** Ethical use is required for spatial data used in military mapping or surveillance.

Example (In-Game Implementation)

1. **Problem Statement**

Title: *"Asteroid Evader" – A 2D Game Using Quadtree for Efficient Collision Detection*

The game lags as the number of objects increases because naively checking collisions between all objects results in $O(n^2)$ time complexity. This can be fixed by using a quadtree to divide the game area spatially and restrict collision checks to nearby objects. This will keep the game running smoothly and drastically lower the computational load.

2. **Implementation (Solution)**

To maximize collision detection the game field is spatially partitioned using a Quadtree. The object is placed into the tree according to its position and the game area is separated into hierarchical quadrants. The player can only interact with objects that are within a certain range of them during each gameplay frame. G. queries are taken from the tree (a 50x50 detection range). This significantly lowers the number of objects that need to be examined for real collisions enhancing the games scalability and performance.

3. **Sample Output**

When the game is run, it prints the number of nearby objects and notifies if any collisions occur. A screenshot of this output can be included to demonstrate real-time collision detection in action. *Code File in the github Link.*

Limitation

- **Unbalanced Trees:** Deeply nested quadrants may result from highly clustered data.
- **Memory Overhead:** Superfluous nodes may be produced for data that is evenly distributed or dense.
- **Static Boundaries:** If data is located outside of predefined boundaries this can result in ineffective partitioning.
- **Fixed Capacity:** Unable to adjust to different data densities.

Conclusion

A strong spatial data structure that makes it possible to efficiently organize insert and query points in two-dimensional space is the quadtree algorithm. Its space-partitioning logic and recursive nature make it perfect for problems involving sparse large datasets such as those found in game engines and geographic systems. Although the majority of use cases benefit from it adaptive variations are required to overcome its shortcomings in managing dense or unpredictable data distributions. Its usefulness and modularity for small-to medium-sized applications are confirmed by the provided C++ implementation.

GitHub Link

https://github.com/Harry-Potter-1122/Analysis-of-Algorithm-Project.git