INTEGER INTERSECTION POINTS

1. Introduction

This work focuses on computing how many intersection points between two families of lines have integer coordinates. The idea originates from analytical tasks typically performed using graphing tools such as Desmos, where both visualization and computation are used to understand geometric relationships.

2. Problem Statement

Given two sets of linear equations of the form $y = x + p_i$ and $y = -x + q_i$, determine how many resulting intersection points have integer coordinates without using an inefficient $O(n^2)$ brute-force approach.

3. Functional Requirements

FR1: Data Input Module – Reads and validates lists of p_i and q_i values. FR2: Processing Module – Computes integer-coordinate intersections using parity logic (po*qo + pe*qe) in O(n) time. FR3: Visualization Module – Plots the lines and marks integer-coordinate intersections clearly.

4. Non-Functional Requirements

NFR1: Performance – Must support input sizes up to 10^5 efficiently. NFR2: Reliability – Ensures consistent and accurate intersection counting. NFR3: Usability – Graphs should be clean, readable, and accessible to the user. NFR4: Maintainability – Modular structure enabling easy modification and extension.

5. System Architecture

The system uses a modular architecture: Input Handling → Processing Logic → Visualization. This separation enhances clarity, maintainability, and scalability.

6. Design Diagrams

Use Case Diagram (Textual): - User → Inputs → System → Outputs intersection count + graph Workflow Diagram (Textual): 1) User inputs p_i and q_i 2) System classifies values into odd/even 3) Computes po*qo + pe*qe 4) Plots lines and integer intersections Sequence Diagram (Textual): User → InputModule → ProcessingModule → VisualizationModule → Output

7. Class / Component Diagram

Core Components: - InputModule: Handles data reading and validation - ProcessingModule: Computes intersection count using optimized logic - VisualizationModule: Produces graphs Relationships: InputModule → ProcessingModule → VisualizationModule

8. Design Decisions & Rationale

Parity-based computation was selected for its O(n) efficiency and mathematical correctness. Visualization adds conceptual clarity, helping users understand line interactions.

9. Implementation Details

Python is used for graphing (matplotlib), while C++ handles efficient computation for large datasets, reflecting competitive programming constraints.

10. Testing Approach

Testing included edge cases (all odd, all even, mixed, extremely large inputs) and validation of plotted visual accuracy.

11. Challenges Faced

Handling very large input sizes, avoiding integer overflow, and ensuring clear visualization were key technical challenges.

12. Learnings & Key Takeaways

Learned optimized computation using parity grouping, improved modular design techniques, and strengthened understanding of geometric visualization.

13. Future Enhancements

Future versions may include 3D visualization, interactive GUI, additional line types, and real-time intersection analysis.

14. References

Codeforces Problem A – Integer Points C++ STL Documentation Matplotlib Documentation