Team Name: The Searchers

Team Members: Blake McGahee, Thiago Ramirez de Arellano, Gerson Diaz

Project Title: Efficient Data Retrieval: A Search Algorithm Performance Study

Problem: To efficiently find specific data points (integers or strings) within a very large dataset and compare the performance of different search algorithms for this task.

Motivation: In many applications, from databases to online directories, quickly locating specific information is fundamental. Understanding the performance characteristics of different search algorithms on large datasets is crucial for designing efficient systems. This project will highlight how algorithmic choices impact real-world application speed.

Features:

- **Data Generation:** Ability to generate a large dataset of unique integers or strings.
- **Data Structure Population:** Populate a chosen data structure (e.g., a sorted array or a custom tree) with the generated data.
- **Search Query Input:** Allow users to input a value to search for.
- **Search Execution:** Execute the search using two distinct, implemented-from-scratch algorithms.
- **Performance Metrics:** Report the time taken (e.g., in milliseconds or microseconds) for each search algorithm to find the element (or confirm its absence).
- **Result Verification:** Confirm if the element was found and its location if applicable.

Data:

- **Randomly Generated Data:** We will randomly generate a dataset of 1,000,000 unique integers. Each integer will be within a large range (e.g., 1 to 10,000,000) to ensure uniqueness and avoid trivial cases.
- **Schema:** The dataset will consist of a single column of `int` type.
- **Size:** 1,000,000 integers, easily exceeding the 100,000 data points requirement.

Tools:

- **Programming Language:** C++ (as recommended and familiar to the team).
- **Libraries:** Only standard C++ libraries (`iostream`, `vector`, `algorithm`, `chrono` for timing, `random` for data generation). We will strictly avoid any pre-built search functions (like `std::binary_search` or `std::find`) or complex data structure libraries.
- **Data Storage:** The generated data will be stored in a `std::vector<int>` for easy access and manipulation

Visuals: The interface will be a command-line menu-driven program.

```
|------------------------------------------------------------|
|     Search Algorithm Performance Study       |
|------------------------------------------------------------|
| 1. Generate Dataset                                |
| 2. Search (Jump Search)                            |
| 3. Search (Interpolation Search)                   |
| 4. Exit                                            |
|------------------------------------------------------------|
| Output:                                            |
| > Enter choice: 2                                  |
| > Enter value to search: [54321]                   |
| Value 54321 found at index 12345.                  |
| Jump Search Time: 0.005 ms                         |
|------------------------------------------------------------|
```

Strategy:

- **Data Representation:** The randomly generated integers will be stored in a `std::vector<int>`. For the chosen search algorithms, this vector will be **sorted once** after generation. Sorting will be done using a standard library sort function (`std::sort`) as the focus is on search algorithm comparison, not sorting algorithm implementation.
- **Algorithms (Comparable & From Scratch):**
    1. **Jump Search (Block Search):** This algorithm will be implemented from scratch. It's a search algorithm for sorted arrays that works by jumping ahead by fixed steps (or blocks) until the range containing the target value is found, then performing a linear search within that block. It provides a good comparison to Interpolation Search in terms of performance characteristics on sorted data.
    2. **Interpolation Search:** This algorithm will also be implemented from scratch. It's an improvement over traditional binary search for uniformly distributed data, as it estimates the position of the target value more intelligently than simply dividing the search space in half. It can achieve O(log log N) on average for uniform data, offering a good comparison to Jump Search.

Distribution of Responsibility and Roles:

- **Blake McGahee:** Data Generation & Sorting, Jump Search Implementation, Performance Analysis & Documentation.
- **Thiago Ramirez de Arellano:** Interpolation Search Implementation, Search Query Logic, Result Verification.
- **Gerson Diaz:** User Interface (CLI/Text) & Output Formatting, Overall Project Integration & Video Presentation.

References:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. (For detailed algorithms of Jump Search and Interpolation Search).
- C++ Standard Library Documentation (for `std::vector`, `std::sort`, `std::chrono`, `std::random`).