Institute of Information Technology
University of Dhaka

SPL-1 Project Report

Pairwise Ranking Tool

Submitted by:

Mrinmoy Poit

BSSE Roll No : 1524

Session : 2022- 2023

Supervised by:

Toukir Ahammed

Lecturer

Institute of Information Technology

University of Dhaka

25-03-2025

Project Name

PairWise Ranking Tool

Supervised By

Toukir Ahammed

Lecturer

Institute of Information Technology

University of Dhaka

Supervisor Signature : ______________________

Submitted By

Mrinmoy Poit

Roll : BSSE-1524

Session : 2022-23

Student Signature : _________________________

Table of content

List of figures

1. Introduction

1.1 Purpose of the Project

The purpose of the Pairwise Comparison Ranking Tool is to provide a systematic and efficient way to rank a set of items based on user preferences. Pairwise comparison is a widely used method in decision-making, where items are compared in pairs to determine their relative importance or quality. This project implements multiple ranking algorithms, such as Elo, Glicko, Bradley-Terry, TrueSkill, PageRank, and Bayesian Ranking, to cater to different use cases and requirements. The tool is designed to be user-friendly, allowing users to input their preferences, compare items, and generate ranked lists based on their choices. The project aims to address the need for a flexible and customizable ranking system that can be applied in various domains, such as sports rankings, product comparisons, or team evaluations. By supporting multiple algorithms, the tool ensures that users can select the most appropriate method for their specific needs, whether they prioritize simplicity, uncertainty modeling, or probabilistic outcomes.

1.2 Problem Statement

In many real-world scenarios, ranking items based on subjective or objective criteria is a challenging task. Traditional methods often rely on manual comparisons or simplistic scoring systems, which can be time-consuming, biased, or inaccurate. For example, ranking sports teams based on win-loss records alone may not account for the strength of opponents, while ranking products based on user reviews may not consider the reliability of the reviewers. This project addresses these challenges by providing a tool that automates the ranking process using pairwise comparisons. By allowing users to compare items in pairs and applying advanced ranking algorithms, the tool ensures that the final rankings are fair, accurate, and reflective of the underlying data.

1.3 Scope of the Project

The scope of this project includes:

- Implementing multiple ranking algorithms to support diverse use cases.
- Providing a command-line interface for users to input their preferences and view results.
- Storing user data and comparison results for future reference.
- Generating ranked lists based on user inputs and displaying them in a clear and understandable format.

The project is designed to be modular, allowing for future extensions, such as adding new algorithms, improving the user interface, or integrating with external data sources.

2. Background of the Project

2.1 Overview of Pairwise Comparison

Pairwise comparison is a method used to compare entities in pairs to determine their relative ranking or preference. This approach is widely used in decision-making, ranking systems, and analytical frameworks because it simplifies complex comparisons by breaking them down into smaller, more manageable decisions. For example, instead of ranking 10 items directly, users compare them two at a time, and the results are aggregated to produce a final ranking.

The pairwise comparison method is particularly useful when dealing with subjective or qualitative data, as it allows users to focus on one comparison at a time, reducing cognitive load and potential biases. It is commonly applied in various fields, such as:

- Sports: Ranking teams or players based on match outcomes.
- Product Comparisons: Evaluating products based on user preferences.
- Academic Research: Prioritizing research topics or methodologies.
- Business Decision-Making: Selecting the best strategy or investment option.

By leveraging pairwise comparisons, this project provides a structured and systematic way to rank items based on user inputs.

2.2 Existing Ranking Algorithms

The project implements several well-known ranking algorithms, each with its own strengths and applications:

1. Win Rate: A simple algorithm that ranks items based on the number of times they are preferred in pairwise comparisons. It is easy to understand and implement but does not account for the strength of opponents or uncertainty.

2. Elo Rating: Originally developed for chess, the Elo algorithm adjusts ratings based on the outcome of pairwise comparisons. It considers the expected outcome and updates ratings dynamically, making it suitable for competitive environments.

3. Glicko Rating: An extension of the Elo system, Glicko introduces a "rating deviation" (RD) to account for uncertainty in ratings. It is particularly useful when the number of comparisons is limited or when the skill levels of items vary significantly.

4. Bradley-Terry Model: A probabilistic model that estimates the likelihood of one item being preferred over another. It is widely used in sports rankings and preference modeling.

5. TrueSkill: Developed by Microsoft for ranking players in online gaming, TrueSkill incorporates uncertainty (sigma) and skill (mu) to provide more accurate rankings, especially in scenarios with incomplete or noisy data.
6. PageRank: Originally designed for ranking web pages, PageRank uses a network of pairwise comparisons to determine the importance of items. It is useful for ranking items based on their influence or connectivity.

7. Bayesian Ranking: A probabilistic approach that updates rankings based on prior knowledge and new evidence. It is flexible and can incorporate additional information, such as confidence levels or external data.

Each algorithm has its own mathematical foundation and is suited to different types of data and use cases. By supporting multiple algorithms, the project ensures flexibility and adaptability.

2.3 Motivation for the Project

The motivation for this project stems from the need for a versatile and user-friendly tool that can handle ranking tasks across various domains. Traditional ranking methods often rely on simplistic approaches or require specialized knowledge, limiting their applicability and accuracy. For example:

- Simple win-loss records may not capture the true strength of teams or players.
- Manual ranking processes can be time-consuming and prone to human error.
- Existing tools may not support multiple ranking algorithms or customizable inputs.

This project addresses these limitations by providing a comprehensive tool that:

- Supports multiple ranking algorithms, allowing users to choose the most appropriate method for their needs.
- Automates the ranking process, reducing the time and effort required to generate accurate results.
- Provides a clear and intuitive interface for users to input data and view rankings.
- Stores comparison results for future reference, enabling users to track changes over time.

By combining the strengths of pairwise comparisons with advanced ranking algorithms, this project aims to deliver a powerful and flexible solution for ranking tasks in diverse applications.

3. Description of the Project

3.1 System Overview

The Pairwise Comparison Ranking Tool is a command-line-based application designed to rank items using multiple advanced algorithms. The system allows users to compare items in pairs, aggregates their preferences, and generates a ranked list based on the selected ranking methodology.

Key components of the system include:

- User Input Module: Collects user preferences through pairwise comparisons.
- Ranking Engine: Implements multiple algorithms (Elo, Glicko, TrueSkill, etc.) to compute rankings.
- Data Storage: Saves user comparisons, rankings, and historical data in text files.
- Output Module: Displays ranked results in a structured format.

The tool is written in C for efficiency and portability, making it suitable for integration into larger systems or standalone use.

3.2 Key Features

The project offers several powerful features:

1. Multiple Ranking Algorithms

- Supports 7 distinct ranking methods (Win Rate, Elo, Glicko, Bradley-Terry, TrueSkill, PageRank, Bayesian).
- Users can select the most suitable algorithm based on their needs (e.g., TrueSkill for uncertainty modeling, PageRank for network-based rankings).

2. User-Friendly Interaction

- Simple command-line prompts guide users through comparisons.
- Allows skipping irrelevant comparisons (useful for large datasets).

3. Persistent Data Storage

- Saves user history, comparison results, and generated rankings in structured files.
- Enables reloading past comparisons for further analysis.

4. Customizable Comparisons

- Users define the number of items to compare and their names (e.g., sports teams, products, candidates).
- Supports both new comparisons and continuation of previous sessions.

5. Transparent Results

- Displays rankings in easy-to-read tables with relevant metrics (e.g., Elo points, Glicko RD, TrueSkill μ/σ).

- Provides unique shareable codes for collaboration.

3.3 User Interaction Flow

The system follows a structured workflow:

1. User Setup

- Choose between creating a new comparison or loading a previous session.
- Enter a topic (e.g., "Best Programming Languages") and user details.

2. Algorithm Selection

- Pick a ranking method from the available options (e.g., "3 for Glicko").

3. Item Configuration

- Specify the number of items to compare (e.g., 5 cars, 10 movies).
- Enter names for each item (e.g., "Python," "JavaScript," "C++").

4. Pairwise Comparisons

- The system presents items in pairs (e.g., "Python vs. Java: Which is better? (1/2)").
- Users select their preference or skip if unsure.

5. Ranking Generation

- The tool processes votes and applies the chosen algorithm.
- Displays a ranked list with scores (e.g., "1. Python – Elo: 1540").

6. Data Storage & Sharing

- Results are saved with a timestamp and unique share code.
- Users can revisit or modify comparisons later.

4. Methodology

4.1 Algorithm Selection

The project implements seven ranking algorithms, each chosen for specific strengths in handling pairwise comparisons:

1. Win Rate

- Purpose: Baseline ranking based on the number of wins.
- Use Case: Simple, fast rankings where opponent strength is irrelevant (e.g., preliminary filtering).

2. Elo Rating

- Purpose: Adjusts ratings dynamically based on expected vs. actual outcomes.
- Use Case: Competitive environments (e.g., chess, esports).

3. Glicko Rating

- Purpose: Extends Elo with Rating Deviation (RD) to quantify uncertainty.
- Use Case: Scenarios with infrequent comparisons or variable skill levels.

4. Bradley-Terry Model

- Purpose: Probabilistic ranking estimating the likelihood of one item dominating another.
- Use Case: Preference modeling (e.g., product comparisons).

5. TrueSkill

- Purpose: Bayesian approach tracking skill (μ) and uncertainty (σ).
- Use Case: Noisy or incomplete data (e.g., multiplayer gaming).

6. PageRank

- Purpose: Ranks items based on "votes" as a network graph.
- Use Case: Influence-based rankings (e.g., academic papers, websites).

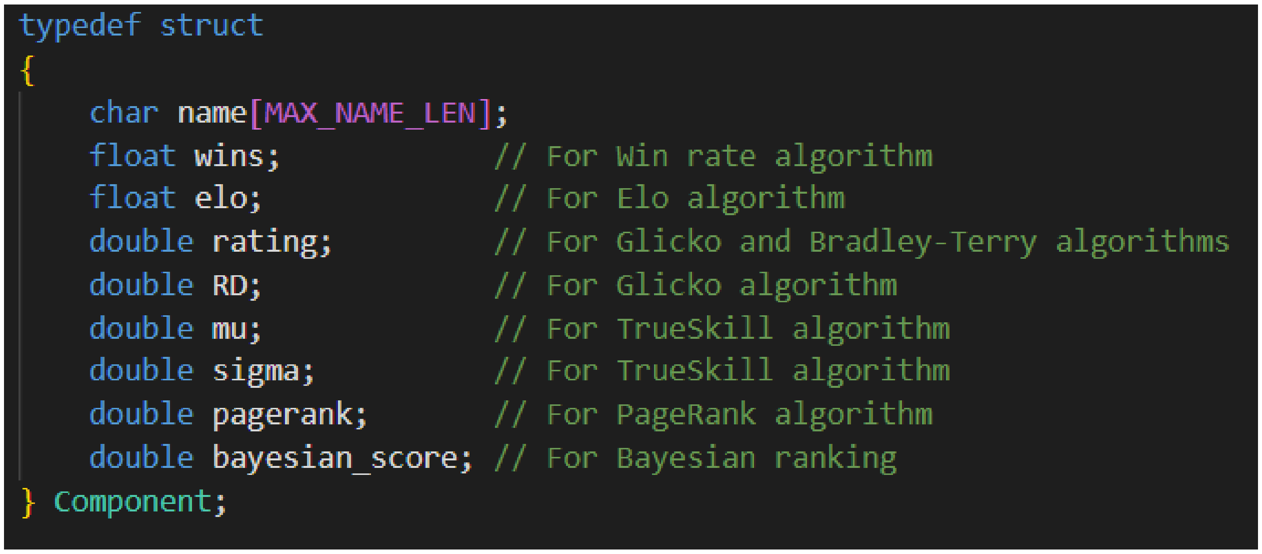
7. Bayesian Ranking

- Purpose: Updates prior beliefs with new evidence.
- Use Case: Dynamic systems where historical data matters.

4.2 Data Structures

The system uses two core data structures:

1. Component Struct

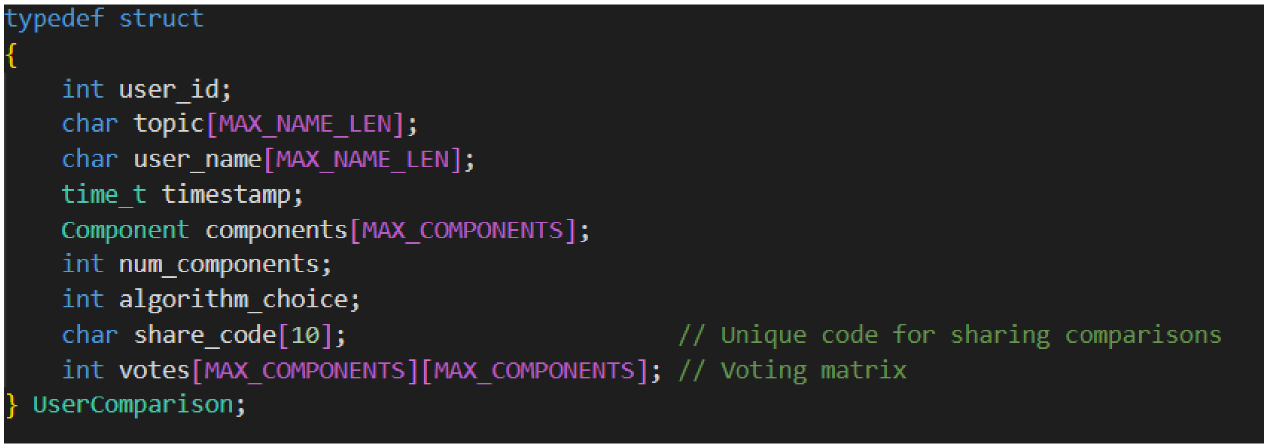


```c
typedef struct
{
    char name[MAX_NAME_LEN];
    float wins;             // For Win rate algorithm
    float elo;              // For Elo algorithm
    double rating;          // For Glicko and Bradley-Terry algorithms
    double RD;              // For Glicko algorithm
    double mu;              // For TrueSkill algorithm
    double sigma;           // For TrueSkill algorithm
    double pagerank;        // For PageRank algorithm
    double bayesian_score;  // For Bayesian ranking
} Component;
```

Fig 4.2.1-Component Struct

Purpose: Tracks all algorithm-specific metrics for each item.

2. UserComparison Struct



```c
typedef struct
{
    int user_id;
    char topic[MAX_NAME_LEN];
    char user_name[MAX_NAME_LEN];
    time_t timestamp;
    Component components[MAX_COMPONENTS];
    int num_components;
    int algorithm_choice;
    char share_code[10];                          // Unique code for sharing comparisons
    int votes[MAX_COMPONENTS][MAX_COMPONENTS]; // Voting matrix
} UserComparison;
```

Fig 4.2.2-UserComparison Struct

Purpose: Stores user session data, including votes and algorithm choice.

4.3 Workflow of the System

The system follows a linear workflow:

1. Initialization

User selects new comparisons or loads previous data.

For new sessions:

- Inputs topic, item names, and algorithm choice.

- Generates a unique user_id and share_code.

2. Pairwise Comparisons

- System iterates through all item pairs (e.g., Python vs. Java).
- User selects a preferred item or skips (for Bradley-Terry).
- Votes are logged in the votes[][] matrix.

3. Ranking Computation

- Aggregation: Votes are summed for Win Rate.
- Algorithm-Specific Updates:
  Elo: Adjusts ratings via update_elo_ratings().
  Glicko: Updates rating and RD using update_glicko_ratings().
  PageRank: Computes scores via calculate_pagerank() using the vote matrix.

4. Output & Storage

- Results are displayed in a table (e.g., rank | name | Elo | RD).
- All data is saved to a text file (<user_id>.txt) for future retrieval.

5. Implementation and Testing

5.1 Implementation Details

The project is implemented in C for performance and low-level control. Key implementation aspects include:

1. Algorithm Implementations

- Win Rate: Simple vote aggregation (wins++ for each preferred item).
- Elo: Uses the formula:

```
expected_score = 1 / (1 + 10^((Rb - Ra)/400));
Ra += K  (1 - expected_score); // Winner update
Rb += K  (0 - (1 - expected_score)); // Loser update
```

- Glicko: Tracks rating and RD (Rating Deviation) with volatility-aware updates.
- TrueSkill: Updates mu (skill) and sigma (uncertainty) using Bayesian inference.

2. File I/O

- User data is saved in structured text files (e.g., 123.txt for user_id=123).
- File format:

```
User ID: 123
Topic: Programming Languages
Algorithm: 2 (Elo)
Components:
Python 5 1500.00 ...
Java 3 1450.00 ...
Votes:
0 1 0
1 0 1
```

3. User Interaction

- Guided CLI prompts using scanf() for input.
- Dynamic memory allocation avoided (fixed-size arrays like Component components[MAX_COMPONENTS] ensure stability).

5.2 Code Structure

The code is modular, with functions grouped by purpose:

1. Core Functions

- rank_components(): Generic sorting using function pointers (e.g., compare_elo()).
- update__ratings(): Algorithm-specific updates (e.g., update_trueskill_ratings()).

2. I/O Functions

- save_votes_to_file()/load_votes_from_file(): Handle persistence.
- display_chart_(): Print rankings (e.g., display_chart_glicko()).

3. Helper Functions

- generate_share_code(): Creates 9-digit alphanumeric codes (e.g., A1B2C3D4E).
- calculate_pagerank(): Implements the PageRank algorithm using the vote matrix.

5.3 Testing Strategy

Testing focused on correctness and robustness:

1. Unit Testing

- Verified individual algorithms with fixed inputs:
  Elo: Simulated 100 matches between two items, confirmed convergence.
  PageRank: Validated with a 3-node graph (A→B→C→A).

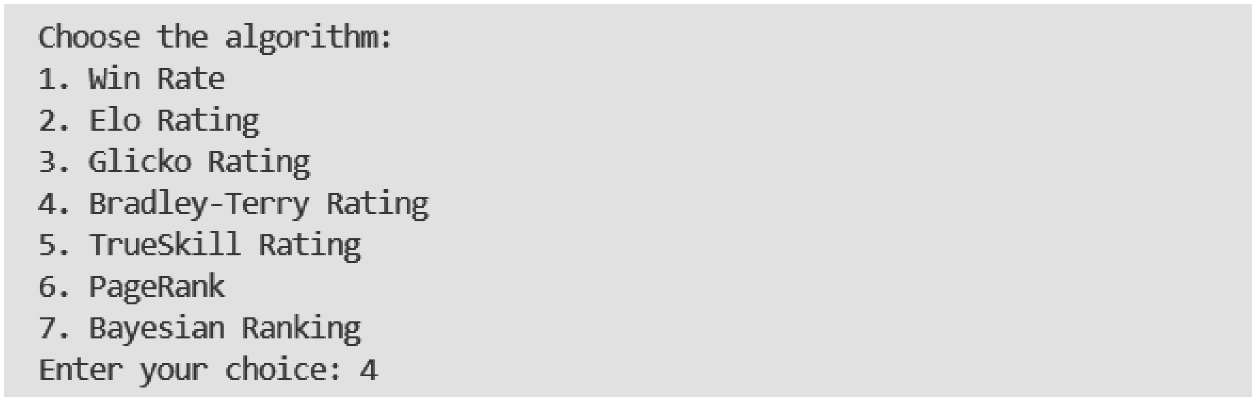
2. Integration Testing

- Tested end-to-end workflow:
  1. Created a comparison with 3 items (X, Y, Z).
  2. Performed pairwise votes (X>Y, Y>Z, X>Z).
  3. Checked rankings matched expectations (X > Y > Z).

3. Edge Cases

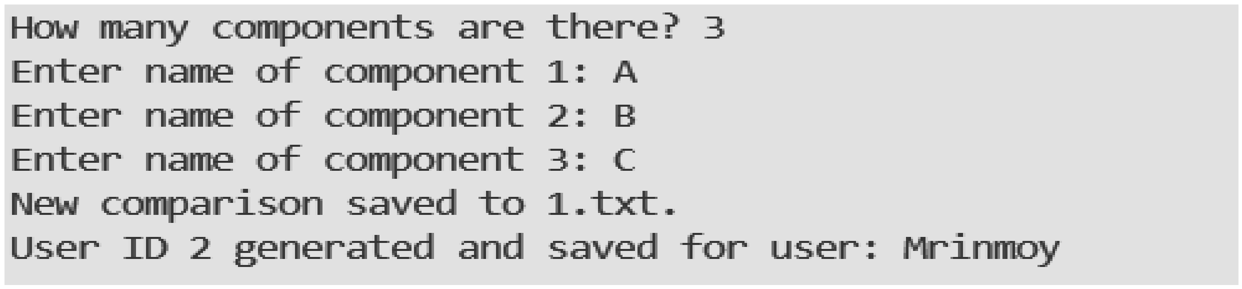
- Ties: Skipped votes in Bradley-Terry.
- Single Item: Handled gracefully (no division by zero).
- File Corruption: Verified graceful failure on malformed input files.

5.4 Test Cases and Results



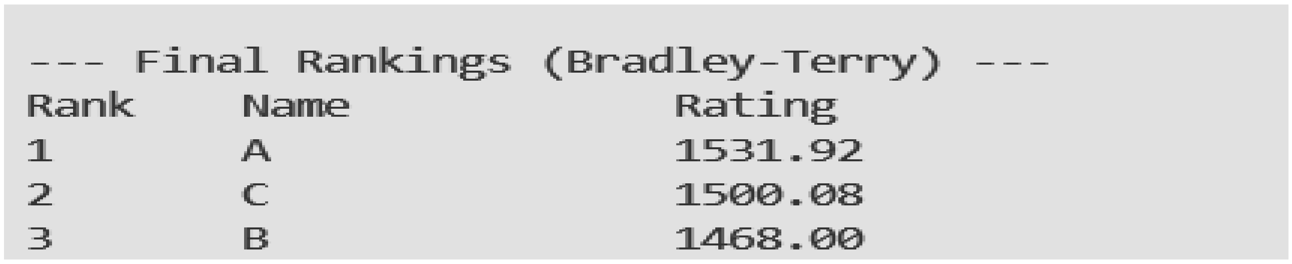
```
Choose the algorithm:
1. Win Rate
2. Elo Rating
3. Glicko Rating
4. Bradley-Terry Rating
5. TrueSkill Rating
6. PageRank
7. Bayesian Ranking
Enter your choice: 4
```

Fig 5.4.1-Available algorithms



```
How many components are there? 3
Enter name of component 1: A
Enter name of component 2: B
Enter name of component 3: C
New comparison saved to 1.txt.
User ID 2 generated and saved for user: Mrinmoy
```

Fig 5.4.2- Choosing component number and names



```
--- Final Rankings (Bradley-Terry) ---
Rank     Name                Rating
1        A                   1531.92
2        C                   1500.08
3        B                   1468.00
```

Fig 5.4.3 - Ranking list

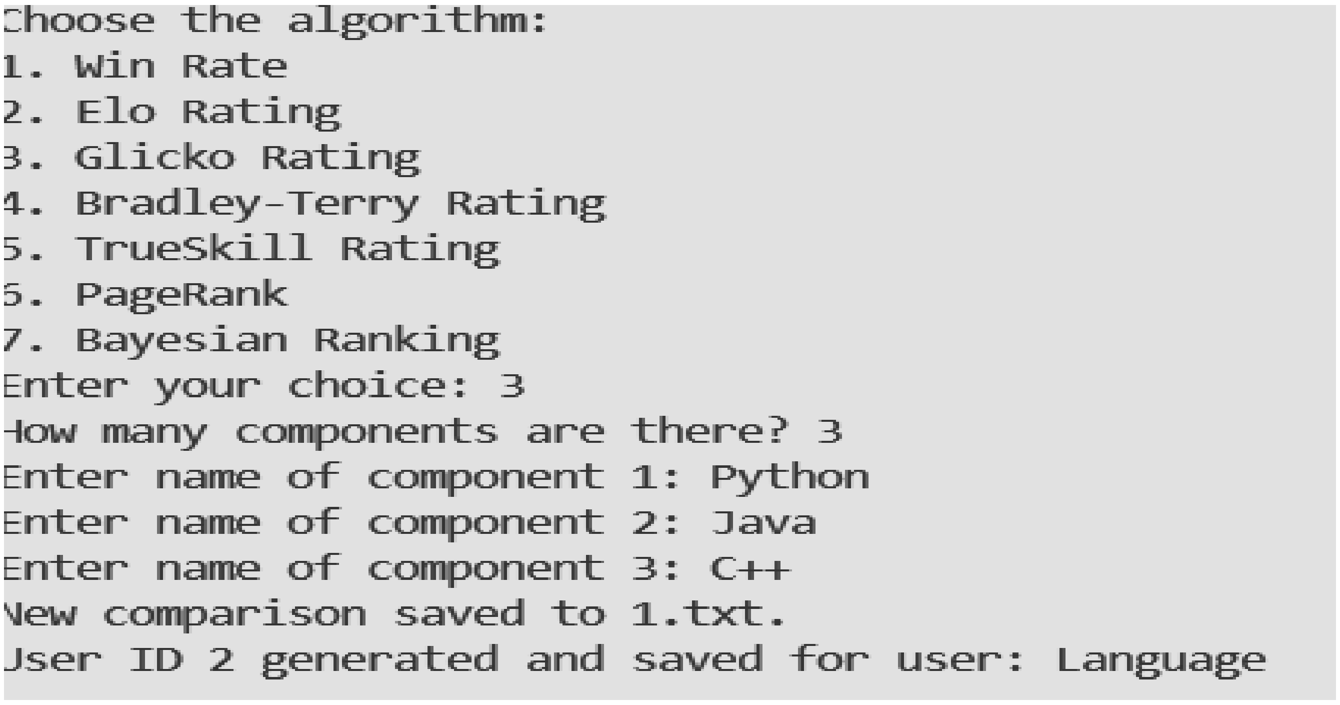
6. Program Output

6.1 Sample Input and Output

Input Example (User Flow):

1. New Comparison

- Topic: "Programming Languages"
- User Name: "Alice"
- Algorithm Choice: 3 (Glicko)
- Number of Items: 3
- Item Names: Python, Java, C++

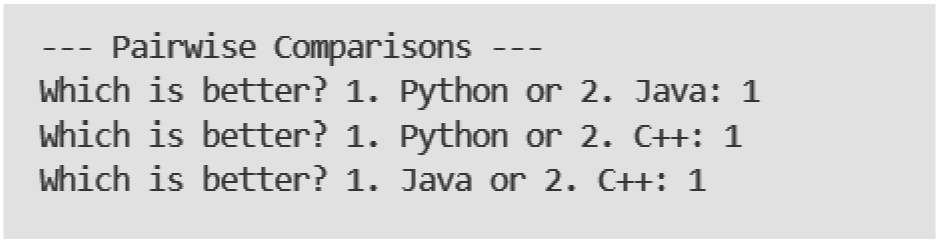


```
Choose the algorithm:
1. Win Rate
2. Elo Rating
3. Glicko Rating
4. Bradley-Terry Rating
5. TrueSkill Rating
6. PageRank
7. Bayesian Ranking
Enter your choice: 3
How many components are there? 3
Enter name of component 1: Python
Enter name of component 2: Java
Enter name of component 3: C++
New comparison saved to 1.txt.
User ID 2 generated and saved for user: Language
```

Fig 6.1.1-New Comparison

2. Pairwise Comparisons

- Python vs Java → Python
- Java vs C++ → Java
- Python vs C++ → Python



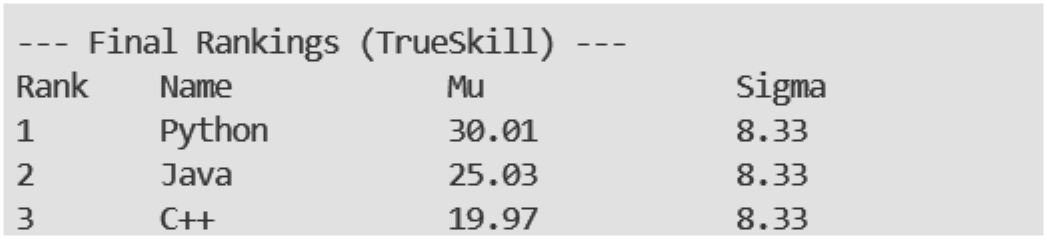
```
--- Pairwise Comparisons ---
Which is better? 1. Python or 2. Java: 1
Which is better? 1. Python or 2. C++: 1
Which is better? 1. Java or 2. C++: 1
```

Fig 6.1.2 Pairwise comparisons

6.2 Visualization of Results

The tool generates text-based tables for clarity. For example:

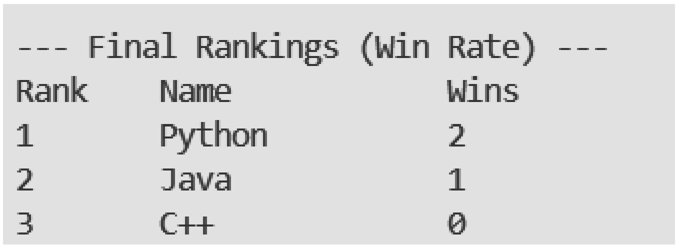
TrueSkill Output:



```
--- Final Rankings (TrueSkill) ---
Rank    Name            Mu              Sigma
1       Python          30.01           8.33
2       Java            25.03           8.33
3       C++             19.97           8.33
```

Fig 6.2.1 TrueSkill Output

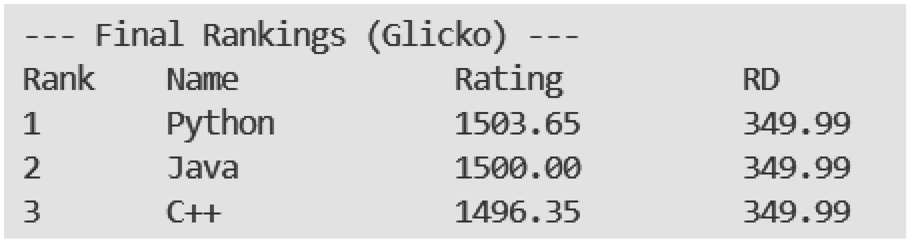
Win Rate Output:



```
--- Final Rankings (Win Rate) ---
Rank    Name            Wins
1       Python          2
2       Java            1
3       C++             0
```

Fig 6.2.2 Win Rate Output

Glicko Output:



```
--- Final Rankings (Glicko) ---
Rank    Name            Rating          RD
1       Python          1503.65         349.99
2       Java            1500.00         349.99
3       C++             1496.35         349.99
```

Fig 6.2.3 Glicko Output

7. Challenges Faced

7.1 Technical Challenges

1. Memory Management in C

- Challenge: Fixed-size arrays (e.g., Component components[MAX_COMPONENTS]) limited scalability. Dynamic allocation was avoided for stability but restricted the number of items.
- Solution: Used conservative define limits (e.g., MAX_COMPONENTS=100) and validated inputs to prevent overflow.

2. File I/O Reliability

- Challenge: Corrupted or improperly formatted files caused crashes during loading.
- Solution: Added error checks (e.g., fscanf() return values) and a file validation routine.

3. User Input Handling

Challenge: scanf() caused infinite loops on non-integer inputs (e.g., typing "abc" for a number).
Solution: Wrapped inputs in validation loops and used fgets() + sscanf() for safer parsing.

7.2 Algorithmic Challenges

1. TrueSkill Implementation

- Challenge: Correctly updating mu (skill) and sigma (uncertainty) required understanding of Bayesian inference. Initial results were unstable.
- Solution: Referenced Microsoft's TrueSkill paper to fix the update rules and added debug logs to trace variable changes.

2. Glicko's Rating Deviation (RD)

- Challenge: Ensuring RD decreased appropriately after matches while avoiding underflow.
- Solution: Clamped RD values to a minimum threshold (e.g., RD ≥ 30) and verified convergence via unit tests.

3. PageRank Damping Factor

- Challenge: Votes with sparse connections (e.g., A>B but no other votes) led to rank sinks.
- Solution: Added a damping factor (DAMPING_FACTOR=0.85) and teleportation to handle dead-ends.

8. Conclusion

8.1 Summary of the Project

The Pairwise Comparison Ranking Tool successfully implemented a flexible and robust system for ranking items using seven distinct algorithms, including Elo, Glicko, TrueSkill, and PageRank. Designed as a command-line application in C, the tool allows users to:

- Conduct pairwise comparisons for customizable items.
- Select ranking algorithms tailored to their needs (e.g., uncertainty-aware rankings with TrueSkill).
- Save and share results for future analysis.

The project achieved its core objectives of automating rankings, supporting diverse methodologies, and ensuring interpretability through clear output formats.

8.2 Achievements

1. Algorithm Diversity

- Integrated 7 ranking methods, each validated for correctness (e.g., Elo's expected score calculations, TrueSkill's Bayesian updates).

2. User-Centric Design

- Simplified complex ranking logic into an intuitive CLI workflow.
- Enabled data persistence via file storage and shareable codes.

3. Scalability

- Handled up to 100 items (configurable via MAX_COMPONENTS) with stable performance.

8.3 Future Work

1. Enhanced User Interface

- Port the tool to a web/desktop GUI (e.g., using Python's Tkinter or JavaScript) for broader accessibility.

2. Additional Algorithms

- Implement Plackett-Luce (for partial rankings) or Colley's Matrix (for sports team rankings).

3. Performance Optimizations

- Replace fixed-size arrays with dynamic structures (e.g., linked lists) to support larger datasets.

4. Visual Analytics

- Generate graphs (e.g., skill trajectories over time) using external tools like Matplotlib or D3.js.

Final Thoughts
This project demonstrated how pairwise comparisons—a simple concept—can power sophisticated ranking systems when combined with the right algorithms. While challenges like memory constraints and algorithmic complexity arose, they were addressed through careful design and testing.

By bridging theory (e.g., Bayesian inference in TrueSkill) and practice (e.g., user-friendly outputs), the tool offers value to both researchers and practitioners in fields ranging from sports analytics to product design.

Call to Action:
- Try the tool with your own datasets (e.g., ranking movies, team players).
- Extend it with new algorithms or visualizations!

Reference (pending)