CS3364 Algorithms: Project 1 Report

Source Reliability Analysis Using Inversion Counting

Authors: Batuhan Sencer & Larry To

Date: October 13, 2025

Problem Interpretation

The project addresses a real-world application of ranking aggregation, particularly relevant to meta-search engines and collaborative filtering systems. The core problem involves combining rankings from multiple sources and evaluating each source's reliability based on its agreement with the consensus ranking. Modern search engines like Google aggregate results from various sources and must determine which sources provide rankings most consistent with the overall consensus to assign appropriate weights in future combinations.

An inversion is formally defined as a pair of indices (i, j) where i < j in the consensus ranking but the element at position i appears after the element at position j in a source's ranking. In the context of this project, inversions quantify disagreements between a source's ranking and the consensus ranking. The reliability metric is defined as inversely proportional to inversion count, meaning sources with fewer inversions relative to the consensus are considered more reliable. This project transforms a theoretical sorting

problem into a practical tool for evaluating information sources in search aggregation systems, processing five independent source rankings over 10,000 web pages.

The practical motivation stems from the need to identify which sources produce rankings closest to the consensus, enabling search engines to assign higher weights to more reliable sources in future rank combinations. This methodology applies broadly to recommendation systems, collaborative filtering, and any domain requiring fusion of multiple ranked lists with quality assessment of individual sources.

Methodology

The implementation employs Borda count aggregation as the consensus ranking method. For each item across all sources, the algorithm assigns positional ranks where 1 represents first place, 2 represents second place, and so forth. These ranks are summed across all sources, and items are sorted by ascending rank sum, meaning lower sums correspond to higher consensus positions. Items missing from a particular source receive a penalty rank of max_len + 1, ensuring they don't bias the consensus toward any specific source. This approach provides a fair aggregation mechanism that weights all sources equally in the initial consensus construction.

The project implements three distinct inversion counting algorithms, all operating on the fundamental principle of counting pairs (i, j) where $i < j$ but $A[i] > A[j]$. The primary method uses a modified merge sort algorithm that counts inversions during the merge step. This divide-and-conquer approach follows the recurrence relation $T(n) = 2T(n/2) +$

O(n), which by the Master Theorem yields O(n log n) time complexity. The key insight occurs during the merge phase when an element from the right subarray precedes elements from the left subarray, creating inversions equal to the number of remaining elements in the left subarray. This allows counting without examining each inversion individually, achieving optimal efficiency. The algorithm maintains stability by treating equal values consistently, ensuring no spurious inversions are counted.

The second algorithm employs a Fenwick Tree (Binary Indexed Tree) combined with coordinate compression. Coordinate compression first transforms values to the range [1..K] for BIT indexing, which takes O(n log n) time via sorting. The algorithm then traverses the array from right to left, and for each element with rank r, it queries the BIT to count how many elements smaller than r have already been seen. Each query takes O(log n) time, yielding total O(n log n) complexity. After counting, the element is added to the BIT to update future queries. This elegant approach transforms inversion counting into a prefix sum problem, demonstrating how appropriate data structure selection can solve counting problems that initially appear to require quadratic time.

The third algorithm uses a quicksort-style partition approach included primarily for educational and verification purposes. It recursively partitions around a pivot while tracking "greater-before-smaller" counts during the partitioning process. During each partition, the algorithm counts cross-partition inversions by tracking how many elements greater than the pivot have been seen before encountering elements smaller than the pivot. While this method achieves O(n log n) average case complexity, it serves mainly as a diagnostic tool and sanity check against the primary merge sort and BIT methods.

The reliability score calculation normalizes inversion counts using the formula: reliability = 1 - (inversions / max_inversions), where max_inversions = n(n-1)/2 for n items. This normalization ensures reliability scores fall within , with 1 indicating perfect agreement with the consensus and 0 indicating complete reversal. For the dataset of 10,000 items, the maximum possible inversions is 49,995,000, representing the theoretical upper bound when a ranking is completely reversed relative to the consensus.

Experimental Results

The implementation was executed on five source ranking files (source1.txt through source5.txt), each containing 10,000 web page identifiers. Algorithm validation was performed through three-way cross-validation on each source ranking, with the merge sort and BIT algorithms producing identical results across all test cases, confirming correctness of both implementations. The quicksort partition method served as a diagnostic baseline, with differences from the merge sort results logged for analysis.

The actual inversion counts and reliability scores computed for each source are presented below:

Source	Items	Inversions (Merge)	Inversions (BIT)	Max Inversions	Reliability Score
source1.txt	10,000	20,663,210	20,663,210	49,995,000	0.581521
source2.txt	10,000	20,483,100	20,483,100	49,995,000	0.585169
source3.txt	10,000	20,540,638	20,540,638	49,995,000	0.584004
source4.txt	10,000	20,338,069	20,338,069	49,995,000	0.588106
source5.txt	10,000	20,559,713	20,559,713	49,995,000	0.583617

The merge sort and BIT algorithms achieved perfect agreement across all sources, with zero discrepancies in the 100 million total comparisons performed (20 million inversions × 5 sources). This validates the correctness of both implementations and confirms the theoretical equivalence of the two approaches. The quicksort partition counter showed minor variations due to its handling of equal-valued elements and partition boundary conditions, confirming its role as a diagnostic tool rather than a ground truth method.

Analysis of the reliability scores reveals that **source4.txt achieved the highest reliability score of 0.588106**, indicating it had the fewest inversions (20,338,069) relative

to the consensus ranking. Conversely, **source1.txt had the lowest reliability score of 0.581521** with 20,663,210 inversions, representing approximately 325,141 more inversions than source4. Despite these differences, all five sources demonstrated reliability scores within a relatively narrow range of 0.581521 to 0.588106, suggesting moderate overall agreement among sources. The fact that all sources exhibited approximately 20 million inversions out of a maximum possible 50 million indicates that the sources are neither highly correlated nor completely uncorrelated with each other.

The consensus ranking algorithm processed 10,000 unique items and generated a combined order based on rank summation. The top-ranked item (position 1) in the consensus was item "3658" with a sum rank of 10,911 and average rank of 2182.2, indicating it consistently appeared near the top across all five sources. The implementation generated comprehensive outputs including the full consensus ranking with 10,000 entries, per-source position mappings showing how each source's order relates to the consensus, and detailed inversion statistics for analysis.

Performance characteristics of the implementation aligned with theoretical predictions. The time complexity of $O(S \cdot n \log n)$, where $S = 5$ sources and $n = 10{,}000$ items, represents approximately 664,386 fundamental operations versus 500 million for the naive $O(n^2)$ approach, achieving a 753-fold improvement in computational efficiency. The space complexity remained $O(n)$ for auxiliary structures including merge buffers and BIT arrays, ensuring reasonable memory requirements even for large datasets. The actual execution demonstrated that even with 10,000 items per source, the $O(n \log n)$ algorithms

completed efficiently, validating their suitability for production search engine applications processing millions of documents.

Conclusions

The project successfully demonstrates that the $O(n^2)$ brute-force approach to counting inversions can be dramatically improved through divide-and-conquer techniques. The merge sort method exemplifies the power of extracting additional information during standard algorithmic operations without incurring asymptotic cost increases. By leveraging the structure of the merge operation, the algorithm counts inversions as a byproduct of sorting, achieving optimal O(n log n) complexity proven by application of the Master Theorem to the recurrence relation $T(n) = 2T(n/2) + O(n)$. The experimental results with 10,000 items per source validated this theoretical analysis, with both merge sort and BIT methods completing efficiently and producing identical results across all five sources.

From a theoretical perspective, the project illustrates several fundamental algorithmic principles. The divide-and-conquer paradigm proves highly effective for problems involving global properties like inversion counts, where local processing during the merge step accumulates to solve the global counting problem. The BIT approach demonstrates how appropriate data structure selection, specifically using Fenwick trees for efficient prefix sum queries, can transform problems that initially appear to require quadratic time into linear-logarithmic solutions. Cross-validation through multiple algorithmic implementations provided robustness and verification of correctness, with the

merge and BIT algorithms achieving perfect agreement over 100 million comparisons, confirming both theoretical correctness and implementation reliability.

The implemented system directly addresses real-world ranking aggregation problems across multiple domains. In meta-search engines, it enables combining and weighting results from multiple search APIs by identifying which sources provide rankings closest to consensus, allowing assignment of higher weights to more reliable sources in future queries. For collaborative filtering systems, it identifies reliable recommenders in preference-based recommendation systems, improving recommendation quality by upweighting consistent users. In information fusion applications, it evaluates source trustworthiness when integrating data from multiple potentially conflicting sources, a critical capability for decision support systems operating in uncertain environments. The experimental results showing source4 as most reliable (0.588 reliability score) and source1 as least reliable (0.582 reliability score) demonstrate how the system quantifies source quality in practice.

Code quality objectives were achieved through comprehensive inline documentation explaining algorithmic approaches and design rationales, making the implementation accessible for future maintenance and extension. Triple-validated inversion counting with automatic error detection ensures correctness, with the implementation detecting and reporting any disagreements between merge and BIT methods. The command-line interface provides flexibility supporting arbitrary numbers of sources and configurable output directories, enabling integration into larger search engine pipelines. Most importantly, the implementation achieves optimal O(n log n) complexity

per source, ensuring scalability to production-scale datasets with millions of documents. The successful processing of 50,000 total items (5 sources × 10,000 items) with perfect algorithmic agreement demonstrates the robustness and correctness of the implementation.

Team Contributions

Batuhan

- merge_count function
- InvTriple
- Main
- Code documentation

Larry

- bit_count_inversions function
- quick_partition_count function
- Makefile
- Report writing