Performance Comparison of HashSet and TreeSet for Storing and Processing Election Data in Java

By Jerrico Garcia

1. Introduction

In modern software systems, especially those that handle large datasets, choosing the right data structure is essential for performance and scalability. This is also true for election systems, where large XML files must be loaded, cleaned, validated, and processed to generate reports.

In our election application, we work with parties, candidates, and vote totals taken from the Dutch Kiesraad. During development, we needed to determine which Java data structure performs better when storing and processing thousands of party records: HashSet or TreeSet. Both belong to the Java Collections Framework and both ensure that elements are unique, but they work very differently internally. As Baeldung (n.d.) explains, HashSet is much faster, while TreeSet automatically sorts elements but at a performance cost.

---

2. Problem and Need

- What is the reason? While processing TK2023 election data (8,560 party records), we noticed clear performance differences depending on whether HashSet or TreeSet was used.
- What is the problem? We do not know which data structure is most suitable for fast processing of election data—especially for adding, searching, and removing records.
- Who has this problem? Developers of election systems, ADS students, and any engineer who must choose a Set implementation in Java (GeeksforGeeks, 2025).
- When did the problem appear? The issue appeared during the integration of the XML parser and benchmark module in the backend.
- Why is it a problem? Using an inefficient data structure can make reports slow or cause delays in processing new vote totals.
- Where does the problem occur? In the Java backend's data service that loads municipalities, parties, and vote totals.

---

3. Main Research Question

What are the performance differences between HashSet and TreeSet when storing and processing election data in Java?

---

4. Sub-questions

- *How do HashSet and TreeSet differ in time complexity for adding, searching, and removing election data?*
- *How do HashSet and TreeSet differ in memory usage when storing the same election data?*
- *How does TreeSet's automatic sorting influence its usefulness for processing and presenting election data?*
- *In which situations in an election system is HashSet more suitable than TreeSet, and vice-versa?*

---

5. Analysis

5.1 Time Complexity and Benchmark Results

HashSet uses a hash table and performs basic operations in O(1) on average (GeeksforGeeks, 2025). TreeSet uses a Red-Black Tree, which performs operations in O(log n) (Baeldung, 2024). According to CodeGym (2025), Red-Black Trees stay balanced by using special color rules, which allows predictable but slower performance.

The benchmark with real TK2023 data showed:

Operation HashSet (ms) TreeSet (ms) Add 1.838 3.600 Contains 0.691 3.011 Remove 0.888 2.296

These results match the theory: TreeSet must keep the tree sorted, making it slower. HashSet is faster because it only relies on hashing.

StackOverflow contributors (2009) even recommend adding elements to a HashSet first and only converting to a TreeSet later if sorting is required.

5.2 Memory Usage Differences

HashSet stores elements in a hash table with buckets, which requires relatively little extra memory. TreeSet stores nodes containing references to child and parent nodes, which increases memory use (Baeldung, 2024). TreeSet typically uses 20–30% more memory than HashSet for similar datasets.

Because HashSet allows one null element and uses hashing internally (GeeksforGeeks, 2025), it is more memory-efficient than TreeSet, which must maintain a complete tree structure.

5.3 Effect of Automatic Sorting in TreeSet

TreeSet automatically sorts all elements based on natural order or a custom comparator (Baeldung, 2024). This makes TreeSet useful when:

alphabetical order is required

data must be exported to reports

user interfaces need consistent ordering

duplicates must be removed in sorted form

fast access to the smallest or largest element is needed (first(), last())

CodeGym (2025) explains that TreeSet's Red-Black Tree guarantees sorted order without manual sorting.

HashSet cannot do this because it does not maintain any order.

5.4 When to Use HashSet or TreeSet

HashSet is better for:

fast searching and validating IDs (GeeksforGeeks, 2025)

performance-critical operations

large datasets that don't need sorting

situations where memory use matters

TreeSet is better for:

automatically sorted elections data (Baeldung, 2024)

alphabetical party lists

UI components where stable order is required

exporting sorted data to reports (Trn, 2024)

As Trn (2024) states, HashSet is best when performance is the priority, while TreeSet is ideal when sorted output is required.

---

6. Conclusion

HashSet is significantly faster than TreeSet when processing election data, especially for searching and inserting. Its constant-time performance makes it suitable for large datasets. TreeSet is slower because it must maintain a sorted Red-Black Tree, but sorting can be a major advantage when presenting or exporting election data.

Therefore:

- Use HashSet for speed-critical logic and deduplication
- Use TreeSet for sorted output and organized presentation
- The correct choice depends on whether speed or ordering is more important in the election system.

---

7. References (APA 7)

Baeldung. (2024). Guide to TreeSet in Java. https://www.baeldung.com/java-tree-set

Baeldung. (n.d.). HashSet vs TreeSet in Java. https://www.baeldung.com/java-hashset-vs-treeset

CodeGym. (2025). TreeSet in Java. https://codegym.cc/groups/posts/treeset-in-java

GeeksforGeeks. (2025). HashSet in Java. https://www.geeksforgeeks.org/java/hashset-in-java/

StackOverflow. (2009). In which cases should I use a HashSet over a TreeSet? https://stackoverflow.com/questions/1463284

Trn, A. (2024). Top 10 Key Differences Between HashSet and TreeSet in Java. https://dev.to/anh_trntun_4732cf3d299/top-10-key-differences-between-hashset-and-treeset-in-java-49f3