CCT College Dublin Continuous Assessment

<table>
<tr><td>Programme Title:</td><td colspan="3">H.Dip. in Computing</td></tr>
<tr><td>Delivery Mode:</td><td colspan="3">FT</td></tr>
<tr><td>Cohort Details:</td><td colspan="3">Feb 2025 cohort</td></tr>
<tr><td>Module Title(s):</td><td colspan="3">Algorithms & Constructs
Software Development Fundamentals</td></tr>
<tr><td>Assignment Type:</td><td>Integrated / Individual</td><td>Weighting(s):</td><td>Algorithms & Constructs – 60%
Software Development Fundamentals – 55%</td></tr>
<tr><td>Assignment Title:</td><td colspan="3">System Modelling & Build</td></tr>
<tr><td>Lecturer(s):</td><td colspan="3">Ken Healy / Taufique Ahmed</td></tr>
<tr><td>Issue Date:</td><td colspan="3">28th Oct 2025</td></tr>
<tr><td>Submission Deadline Date:</td><td colspan="3">Saturday 29th Nov at 11.59pm</td></tr>
<tr><td>Late Submission Penalty:</td><td colspan="3">Late submissions will be accepted up to 5 calendar days after the deadline. All late submissions are subject to a penalty of 10% <u>of the mark awarded</u>.
Submissions received more than 5 calendar days after the deadline above <u>will not</u> be accepted and a mark of 0% will be awarded.</td></tr>
<tr><td>Method of Submission:</td><td colspan="3">This assignment is submitted via Moodle.</td></tr>
<tr><td>Instructions for Submission:</td><td colspan="3"><u>Algorithms & Constructs</u>
A GitHub repo must be created. The Netbeans project and report Word document must be put into a GitHub repo for version control. The lecturer must be added to the repository as a collaborator. The GitHub repo link will be added at the end of the report. There should be at least 10 to 15 commits throughout the time worked on the project. Submit your Netbeans .java files individually, and make sure all files contain the same package name, CA_2.
If your submission does not load or run, then you may receive a zero mark! Include a section (max 500 words) in your Report submission that clearly explains your choices for the sorting & searching algorithms that you have used.

<u>Software Development Fundamentals</u>
Your design diagrams, user stories, etc. must be submitted in one document in PDF format to the SDF Moodle page. You do not need to include this in your GitHub Repo.
There will be an IN-CLASS Q&A held on Tues 2nd Dec. You may be asked to explain any part of your submission. Late submissions will be required to attend a separate Q&A in week commencing 8th Dec.</td></tr>
<tr><td>Feedback Method:</td><td colspan="3">Results posted in Moodle gradebook</td></tr>
<tr><td>Feedback Date:</td><td colspan="3">Following release of approved results by the college</td></tr>
</table>

Introduction

The project deals with designing a system known as the Department Store Employee Management System, using the Java programming language in the NetBeans development environment. The system shall be capable of storing, organizing, and retrieving employee information like the employee's name, type of manager, department, and every piece of relevant information in an effective and reliable way.

In large and complex systems, such as department store chains, efficient management of employee information is the key to rapid decision-making, manpower planning, and administrative report generation. To this end, sorting and searching algorithms are used to ensure that performance will be consistent despite the growth of databases.

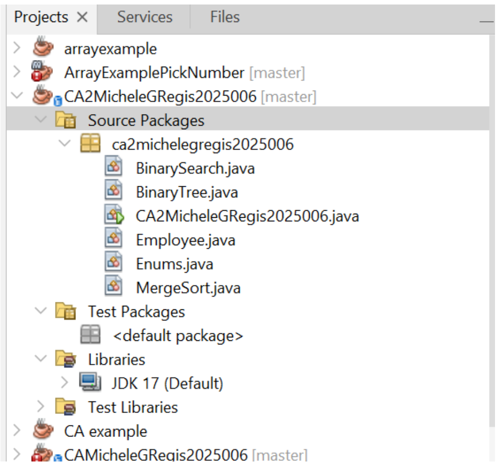
Version control is also used on this project through GitHub, which provides tracking of changes, facilitates collaboration, and offers a record of the history of development. This report describes the chosen algorithms-Merge Sort for sorting and Binary Search for searching-and gives justifications for each selection based on performance, scalability, and reliability. The system was designed to be expandable, allowing for future implementations such as secondary sorting, range queries, and graphical or reporting interfaces.

GitHub-

https://github.com/MicheleRegis/CA2MicheleGRegis2025006.git

Algorithms and Constructs

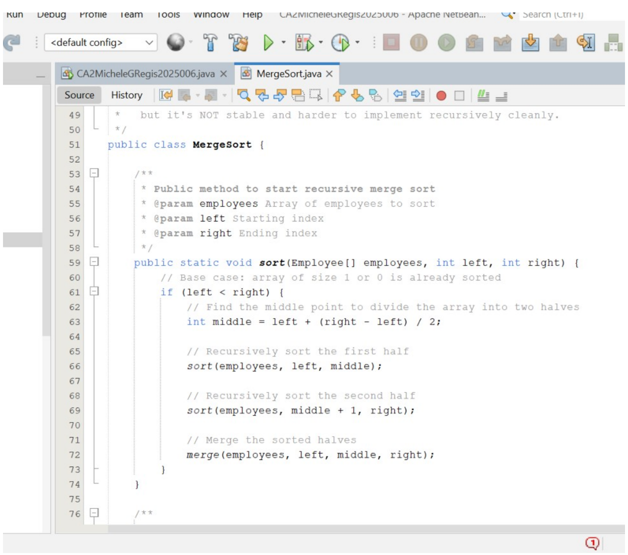
The Department Employee Management System

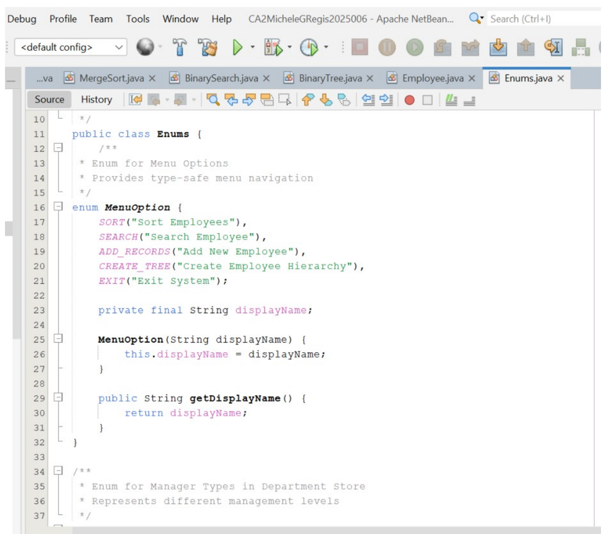


Sorting Algorithm: Merge Sort

Main Reason

I chose Merge Sort for the recursive sorting algorithm for this department store system because it guarantees an O(n log n) time complexity in all cases: best, average, and worst. Unlike Quick Sort, it doesn't degrade to O(n²) with poorly chosen pivots or already-sorted data. Merge Sort exhibits predictable, reliable performance irrespective of input patterns, which is very important in a production environment serving department store managers.





Key Advantages

Another important aspect is stability. The Merge Sort algorithm is a stable sort, meaning that equal elements remain in their original order. This becomes useful later if we extend our system to include secondary sorting criteria, such as employee ID or hire date.

The problem fits perfectly with the divide-and-conquer recursive approach. Merge Sort naturally splits the problem into smaller subproblems, sorts them recursively, and then merges the results. This is a very elegant recursive structure that makes the code maintainable, with proper algorithm design principles in mind.

Scalability Considerations

With the increase of department stores, employee databases grow from hundreds to possibly thousands of records. This $O(n \log n)$ complexity of Merge Sort makes the system scale well. If the number of employees is doubled, the time taken increases a bit more than double-not that quadratic explosion as in the simpler $O(n^2)$ algorithms like Bubble or Insertion Sort.

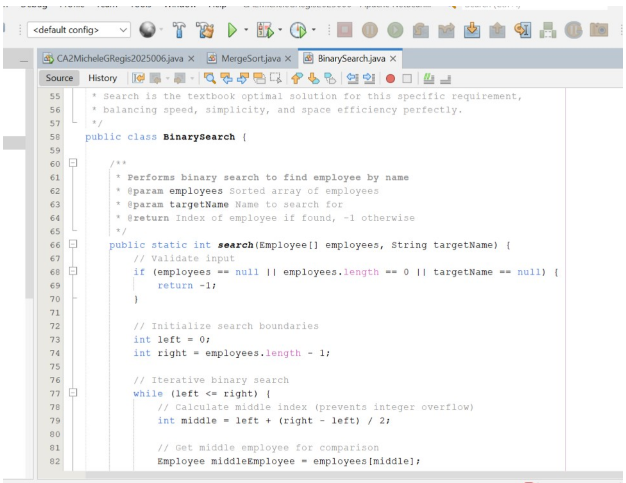
Trade-Offs Acknowledged

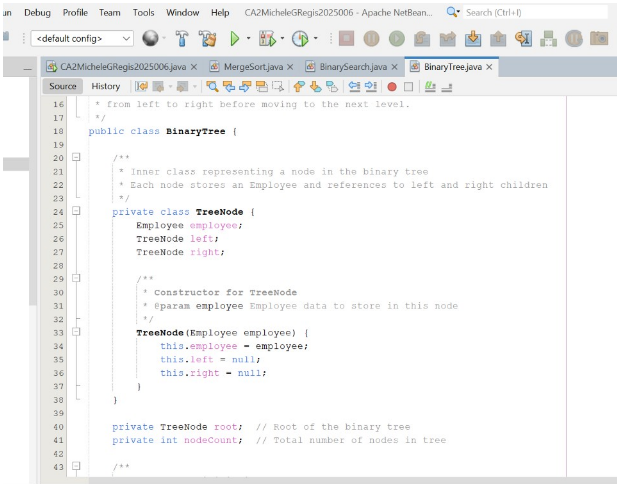
The main trade-off here is space complexity: Merge Sort requires $O(n)$ auxiliary space for the merging operation. However, on modern systems, this is usually acceptable given the reliability benefits that come with it. I have taken the approach of sacrificing memory to keep up consistent performance. Besides, the extra memory consumption is temporary and freed right after the sorting is done.

Searching Algorithm:Binary Search

Core Reasoning

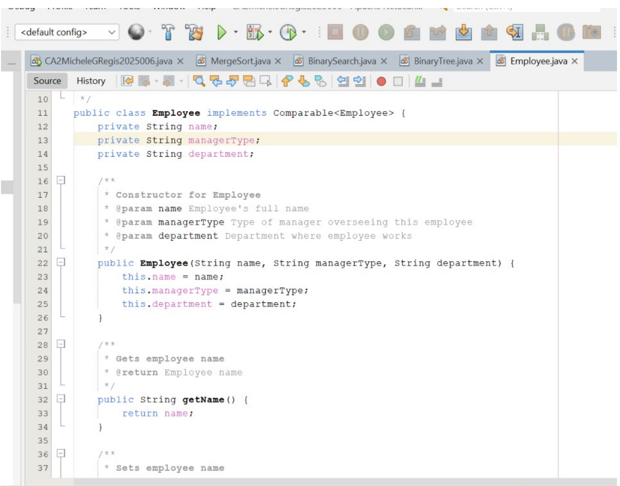
Binary search was the obvious choice for looking up employees, since it provides a time complexity of O(log n). Since our data is already sorted through Merge Sort, Binary Search takes advantage of this structure to locate employees with logarithmic efficiency. For 1,000 employees, it only takes roughly 10 comparisons versus the average of 500 comparisons in Linear Search.





Practical Impact

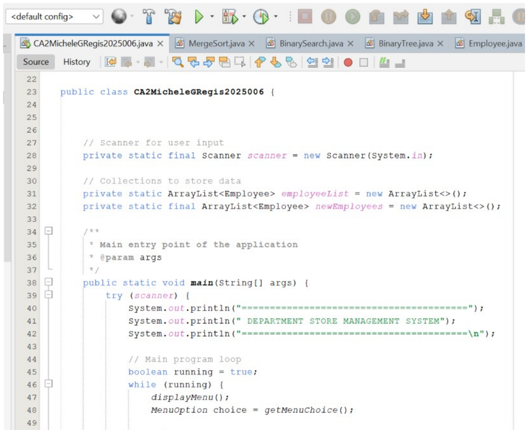
Speed counts in a busy department store where managers frequently query employee information for scheduling, performance reviews, or department assignments. Similarly, Binary Search provides instant results even as the number of staff grows, which again improves user experience and operational efficiency.



Why Not Alternatives?

Linear Search is rejected, though simple, because it wastes the sorted structure that we maintained with such care. For very large datasets, this would be noticeably slow.

Hash tables might seem faster, averaging O(1); but they require O(n) extra space for the hash structure and don't support features we may want in future versions-such as finding "closest matches" for partial name searches or doing range queries for alphabetical sections.



Algorithm Synergy

Binary Search and Merge Sort go together perfectly. Merge Sort creates the required sorted structure that Binary Search needs. This design shows an understanding of algorithm relationships and dependencies in data structures.

Conclusion

Both algorithms were chosen for reliability, scalability, and performance guarantees rather than clever optimizations that might fail under specific conditions. This conservative, production-ready approach ensures that the system serves department store staff effectively as the organization grows.

GitHub-

https://github.com/MicheleRegis/CA2MicheleGRegis2025006.git

Reference

Sehgal, K., 2018. *A Simplified Explanation of Merge Sort*. Medium, 25 January. Available at: https://medium.com/karuna-sehgal/a-simplified-explanation-of-merge-sort-77089fe03bb2 [Accessed 16 November 2025].

Codecademy Team, 2025. *Time Complexity of Merge Sort: A Detailed Analysis*. Codecademy. Available at: https://www.codecademy.com/article/time-complexity-of-merge-sort [Accessed 16 November 2025].

EnjoyAlgorithms, 2020. *Merge Sort – Algorithm, Source Code, Time Complexity*. HappyCoders.eu. Available at: https://www.happycoders.eu/algorithms/merge-sort/ [Accessed 16 November 2025].

Crio.Do, 2024. *Everything You Need To Know About Merge Sort*. Crio Blog, 9 September. Available at: https://www.crio.do/blog/merge-sort-algorithm-dsa [Accessed 16 November 2025].

Crio.Do, 2022. *Understanding Binary Search Algorithm*. Crio Blog. Available at: https://www.crio.do/blog/understanding-binary-search-algorithm/ [Accessed 16 November 2025].

EnjoyAlgorithms, 2023. *Binary Search Algorithm – EnjoyAlgorithms*. EnjoyAlgorithms Blog. Available at: https://www.enjoyalgorithms.com/blog/binary-search-algorithm/ [Accessed 16 November 2025].

InterviewKickstart, 2025. *Binary Search Algorithm: How It Works and Why It's Efficient*. InterviewKickstart Blog. Available at: https://interviewkickstart.com/blogs/learn/binary-search [Accessed 16 November 2025].