**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | **Algorithms & Constructs** |
| **Assessment Title:** | **System Modelling & Build** |
| **Lecturer Name:** | **Muhammad Iqbal** |
| **Student Full Name:** | **Iveelt Batjargal** |
| **Student Number:** | **2024569** |
| **Assessment Due Date:** | **10 May 2025** |
| **Date of Submission** | **09 May 2025** |

Declaration

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

Table of Contents (Sorting and Searching Algorithm)

[Sorting Algorithm: Recursive Insertion Sort 3](#_Toc197378163)

[Searching Algorithm: Recursive Binary Search 4](#_Toc197378164)

[GitHub link 4](#_Toc197378165)

[References 5](#_Toc197378166)

Sorting and Searching Algorithms

# Sorting Algorithm: Recursive Insertion Sort

In the Tech Company management system, I chose recursive Insertion Sort to sort the list of employees by name because, during classroom comparisons with other sorting methods alongside my instructor, it performed faster. Additionally, through further research, I learned that this algorithm is particularly well-suited for small to medium-sized datasets, specifically those with fewer than 100 elements. The dataset from my organization, stored in the Applicants\_Form.txt file, aligned perfectly with this requirement. The simplicity of recursive Insertion Sort makes the code easy to read and maintain, which is critical for a prototype that prioritizes development speed and clarity over handling large datasets. This factor also influenced my decision.

When comparing recursive Insertion Sort to Bubble Sort and Selection Sort, I selected Insertion Sort for the following reasons: Recursive Insertion Sort outperforms Bubble Sort on small datasets because it requires fewer comparisons and shifts for partially sorted lists. Bubble Sort’s performance is hindered by its constant pairwise comparisons, making it slower. Similarly, Selection Sort always searches for the minimum element, requiring more comparisons. Moreover, the recursive implementation of Insertion Sort clearly demonstrates the divide-and-conquer principle, enhancing code readability. Bubble Sort’s iterative loops, while simple, fail to meet the recursive requirement, and Selection Sort is unstable, potentially affecting data integrity.

In summary, recursive Insertion Sort excels in performance for small datasets, stability, and meeting the recursive requirement, making it superior to Selection Sort and Bubble Sort. Its simplicity, adherence to recursive requirements, and efficiency for small datasets also make it a better choice than Merge Sort’s memory demands and Quick Sort’s complexity for this prototype.

# Searching Algorithm: Recursive Binary Search

For searching an employee by name in the sorted list, I chose recursive Binary Search because I used recursive Insertion Sort to maintain a sorted ArrayList, making recursive Binary Search the most suitable for searches with optimal time complexity. In classroom experiments comparing Linear Search and Binary Search, Binary Search performed faster with fewer operations. This method seamlessly integrates with the sorted list, halving the search space with each recursive call to ensure consistent logarithmic performance.

Linear Search, while simple to implement, is impractical for multiple searches as it sequentially checks each element. The recursive nature of Binary Search aligns with the system’s recursive sorting, creating a cohesive algorithmic framework. The implementation in MyArrayList provides a clean interface for both recursive and iterative searches, but the recursive variant was chosen for its clarity and compliance with the recursive requirement. Support for partial name matching, achieved through a secondary linear scan, enhances user experience without compromising the efficiency of the primary recursive Binary Search for exact matches. For a prototype requiring efficient, recursive, and maintainable search functionality on a sorted list, recursive Binary Search outperforms Linear Search’s inefficiency.

# GitHub link:

<https://github.com/Iveelt-B/Algor_Constr_CA2_2024569/tree/master>

# References

* GeeksForGeeks (25 Apr, 2025). *Merge Sort - GeeksforGeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/merge-sort/>.
* GeeksforGeeks (17 Apr, 2025). *Quick Sort Algorithm*. [online] GeeksforGeeks.

Available at: <https://www.geeksforgeeks.org/quick-sort-algorithm/>.

* [Muhammad Iqbal]. (2025). Algorithms and Constructs: Bubble Sort Algorithm. [CCT College Dublin].
* [Muhammad Iqbal]. (2025). Algorithms and Constructs: Insertion Sort Algorithm. [CCT College Dublin].
* [Muhammad Iqbal]. (2025). Algorithms and Constructs: Binary Search and Linear Search [CCT College Dublin].

‌