**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | HDIP in Science in Computing |
| **Assessment Title:** | Algorithms and Constructs CA-2 |
| **Lecturer Name:** | Dr Muhammad Iqbal |
| **Student Full Name:** | Davaabayar Usukhbayar |
| **Student Number:** | 2024217 |
| **Assessment Due Date:** | 07th April 2025 |
| **Date of Submission:** | 14th 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

[Why I Picked These Algorithms: Sorting & Searching 3](#_Toc198114385)

[GITHUB LINK 3](#_Toc198114386)

# 

# **Why I Picked These Algorithms: Sorting & Searching**

For my hospital staff management system, I decided to use **recursive insertion sort** to organise the list of employees alphabetically and **binary search** to find staff by name. I chose both based on the size of the data I’m working with, how the program is structured, and what makes sense for a real-world hospital system.

I went with **recursive insertion sort** because the staff list in this app isn’t very big—maybe 20 or 30 names max—so I didn’t need a complex sorting algorithm like merge sort or quicksort. Insertion sort is already great for small or almost-sorted lists, and the recursive version made the code shorter and cleaner. Since I wanted something that works well but is still easy to follow and maintain, this felt like a solid choice.

Another reason I chose it is because it doesn’t use extra memory—everything happens within the list itself. That might not be a big deal now, but it’s good practice if the system ever ends up on low-resource machines or gets reused elsewhere. Compared to bubble sort or selection sort, it performs better and isn’t just comparing or swapping things randomly. Merge sort was tempting, but it adds more complexity than I needed, and the goal was to keep the solution simple and efficient.

For **searching**, I used **binary search**, mainly because I’m already sorting the list. It would’ve been a waste to go through the list one by one (like linear search does) when I could just split it in half each time and find what I need faster. Binary search is perfect when the data is sorted, and it makes the app feel quicker and more responsive—especially if the list of employees grows bigger in future versions.

This kind of speed matters in a hospital setting. If someone needs to find a doctor or nurse quickly based on their name, the system shouldn’t lag. Binary search helps with that. Plus, it’s a good example of choosing the right algorithm based on the data and setup. Linear search might’ve been easier to write, but binary search was the smarter choice in this case.

In the end, I chose both algorithms because they match the size and purpose of my program. They’re easy to understand, they keep things efficient, and they allowed me to practice important concepts like recursion and performance-based decision-making. For a small hospital staff system like this one, recursive insertion sort and binary search gave me the perfect balance between simplicity and speed.

# GITHUB LINK

<https://github.com/Dvk822/HospitalSystemCA_2.git>