INDIVIDUAL COURSEWORK: LIBRARY MANAGEMENT SYSTEM

**Table of Contents**

[Introduction and description of the project 3](#_Toc201332224)

[Justification of selected data structures and algorithms 3](#_Toc201332225)

[Time complexity analysis with pseudo-code 5](#_Toc201332226)

[Testing approach and test cases 7](#_Toc201332227)

[Conclusion with limitations and reflection 7](#_Toc201332228)

[References 8](#_Toc201332229)

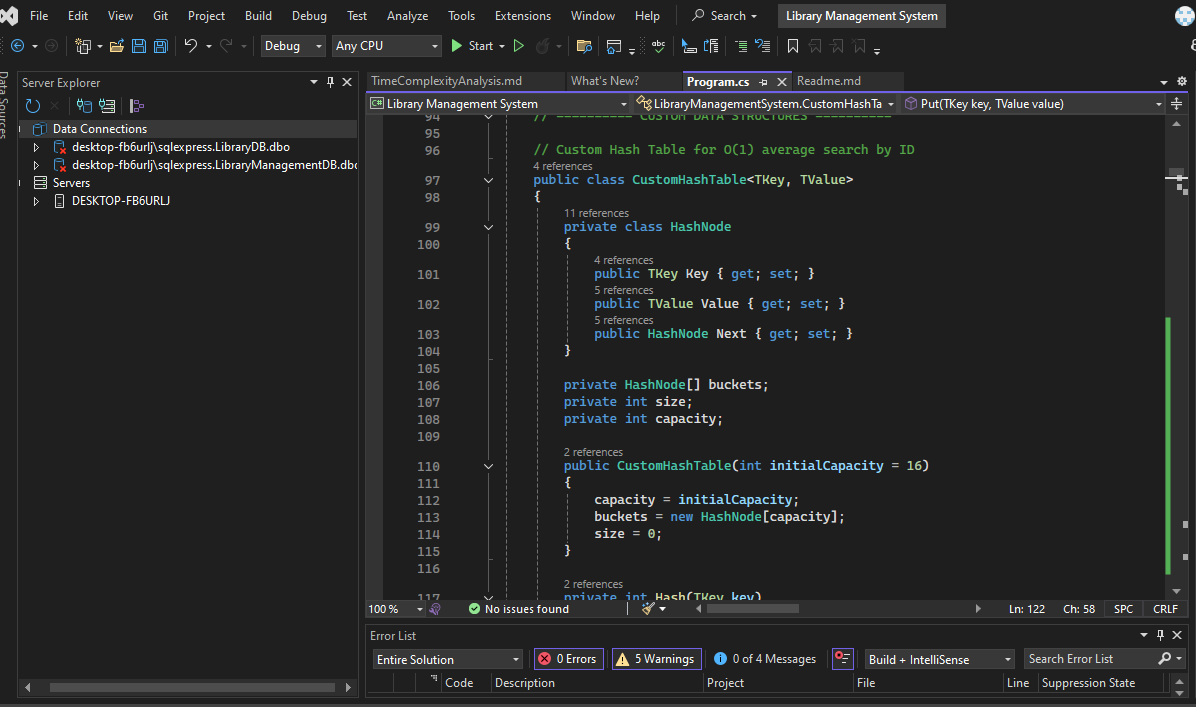
# 

# Introduction and description of the project

The Library Management System has been developed in this [project which is a console-based application and is developed using a framework written in .NET framework, characterised by the presence of a well-implemented data structure and efficient use of resources. To support ID-based O(1) access, a hash table as a custom-built data structure is used, and to support more efficient string-based searches, binary search trees are used. It runs library resources under SQL Server database

# Justification of selected data structures and algorithms

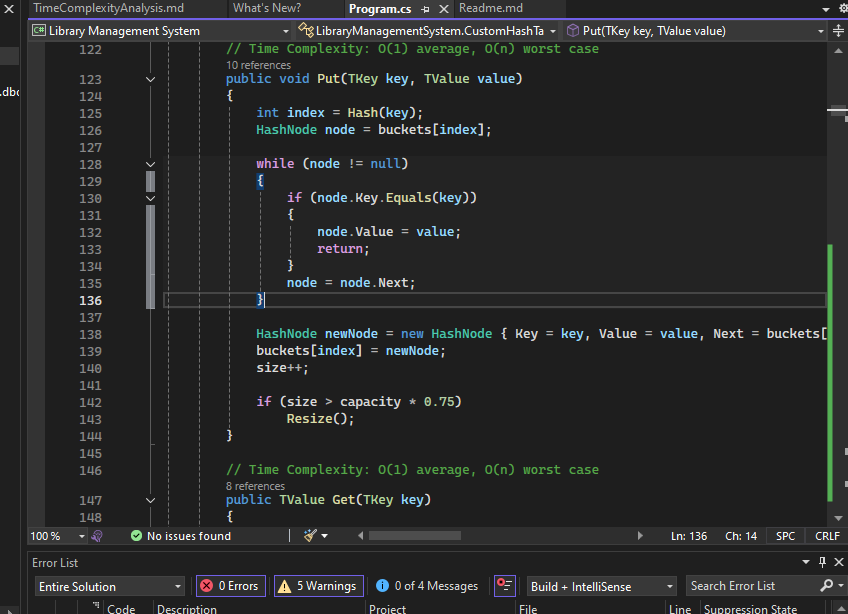
A combination of personal data structures design and well selected algorithms allows optimising the performance and maintainability of the Library Management System.



**Figure 1: Use of custom data structure**

(Source: Acquired from vs studio)

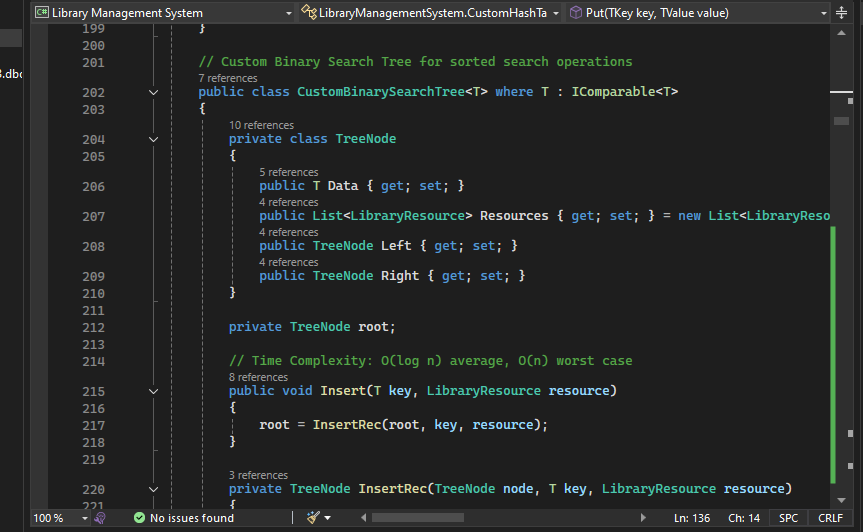
It is desirable when the resource caching employs a custom hash table because the average-case time complexity of such an operation as insertion, lookup of resources, or an update type of an operation is O(1) (Del*.* 2022). This will guarantee that resource retrieval based on the ID will be always fast despite the number of items in the dataset.



**Figure 2: Analysing time complexity average**

(Source: Acquired from vs studio)

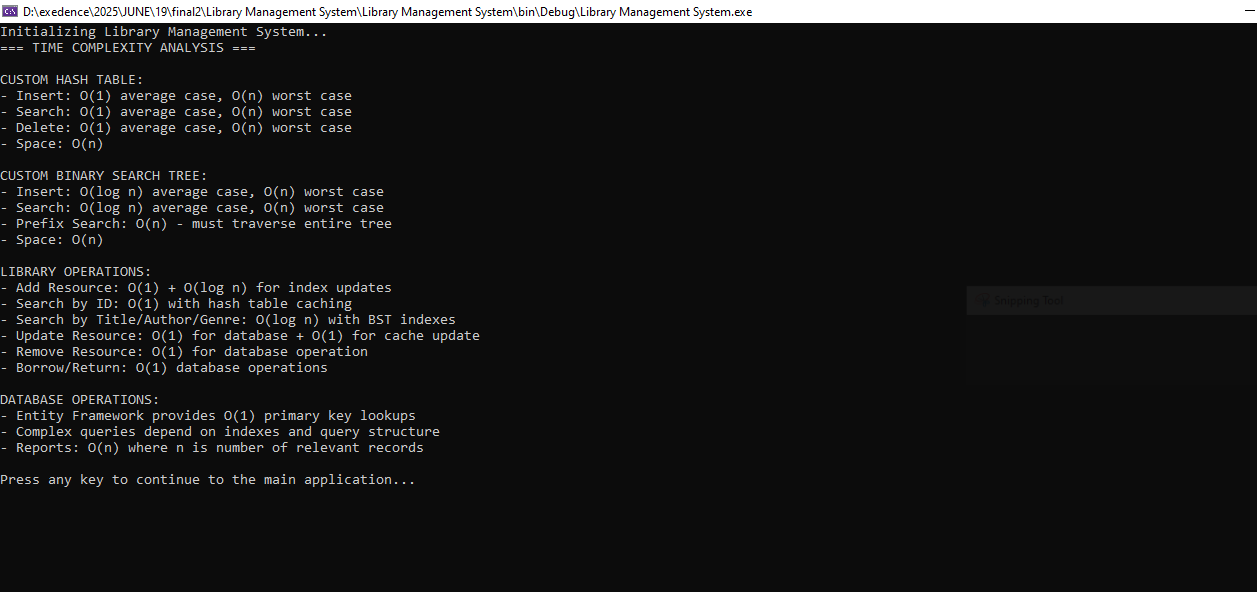
Another feature of the hash table is that it is easily resizable depending on the size of a library and that it efficiently supports collision resolution.



**Figure 3: Binary search tree for sorted search operation**

(Source: Acquired from vs studio)

Under such searches that involve strings like titles, authors, or genre, the system makes use of tailored binary search trees (BSTs). The average-case complexity of insert and search operation in BSTs is O(log n), and this allows quick searching and prefix searches.

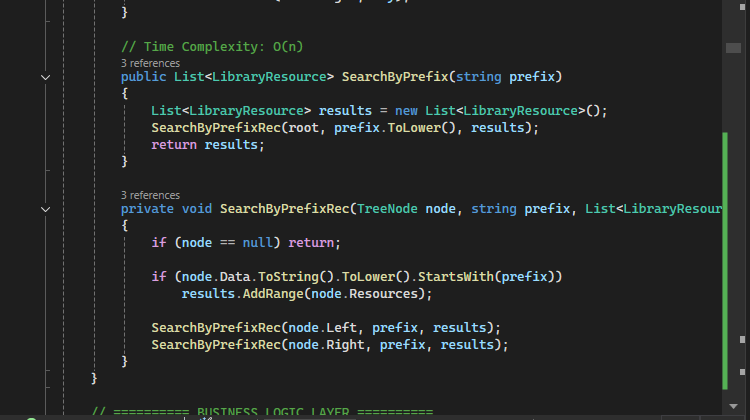


**Figure 4: Output showing the implementation of data structures**

(Source: Acquired from vs studio)

These data structures, coupled with such algorithm based strategies as caching, delayed-loading and grouping performed on the server, balance lightening, scalability and resource consumption of the system (Alls, 2022).

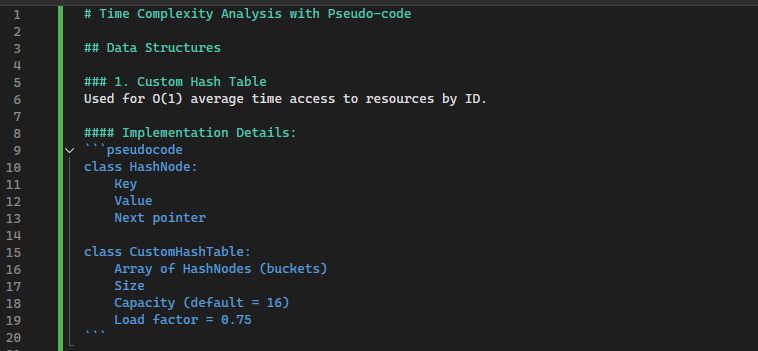
# Time complexity analysis with pseudo-code



**Figure 5: Time complexity analysis**

(Source: Acquired from vs studio)

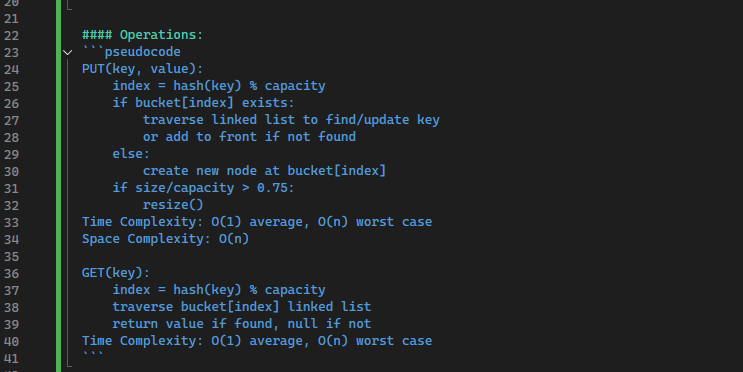
Time Complexity Analysis provides a sufficient analysis of algorithm effectiveness in the Big O notation that represents the hypothetical behavior of hash tables ( O (1) ) and binary search trees ( O (log n) ) data structures.



**Figure 6: Custom time complexity implementation pseudo code**

(Source: Acquired from vs studio)

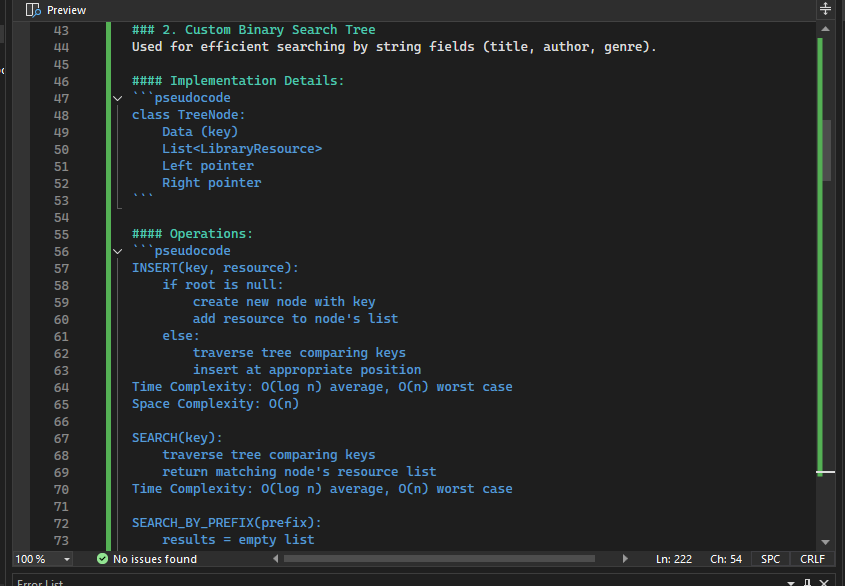
Time complexity implementation Pseudo Code shows custom data structure implementations that include hash tables which have resistance to collisions and it is dynamically resizable to give the best performance measurements.



**Figure 7: Operation pseudo code**

(Source: Acquired from vs studio)

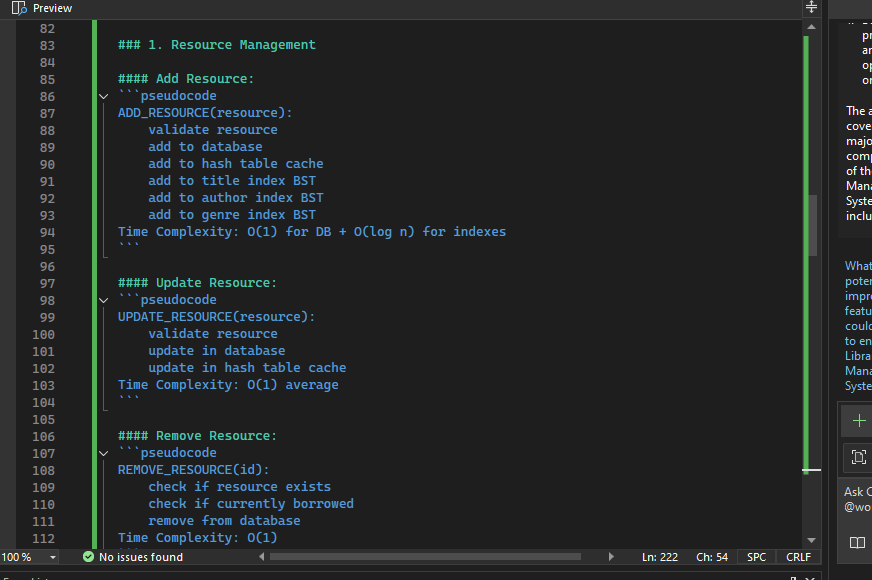
The detailed description of core library operations in well-structured pseudo code with specified limits of complexity contain validation, database operations and cache access and manipulation.



**Figure 8: Custom binary tree pseudo code**

(Source: Acquired from vs studio)

Binary search tree implementation is an example of hierarchical data structure, wherein insertions, searches and prefix based queries have recursive operations with the logarithmic time complexity (Golmohammadi *et al.* 2023).

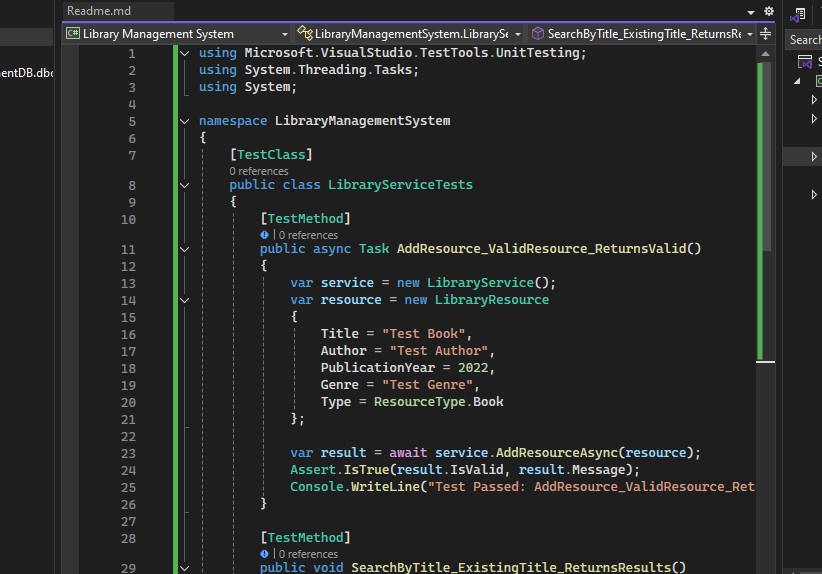


**Figure 9: Full application crud operation pseudo code**

(Source: Acquired from vs studio)

Pseudo Code CRUD Operation demonstrate the examples of database interactions, validation logic, and index management according to the data integrity, and striving to maintain the best performance with the help of the caching mechanism.

# Testing approach and test cases



**Figure 10: Unit testing code implementation**

(Source: Acquired from vs studio)

The entire unit test case code of this library management system performed in this project has been shown in the above figure (Marcotte, 2024).



**Figure 11: Unit testing completion and passed**

(Source: Acquired from vs studio)

The above output figure illustrates that all the test cases passed and completed properly. \

# Conclusion with limitations and reflection

***Conclusion with limitation:***

The LMS is powerful in the usage of resources and borrowings with personal data structures and a strong business logic layer. Nevertheless, they can affect scalability and performance with the increasing amount of data; in-memory indexes and client-side grouping might get in the way. It could be improved upon by integrating with more sophisticated search algorithms, or distributed caching where efficiency in potentially larger libraries increases.

***Reflection:***

In creating this application, I became more acquainted with custom data structures, Entity Framework and test-driven development. The procedure helped me to find a compromise between performance, preservability, and user experience, thus enhancing my ability to create scalable and reliable solutions of software.

# 

# References

Alls, J., 2022. High-Performance Programming in C# and. *NET: Understand the nuts and bolts of developing robust, faster, and resilient applications in C# 10.0 and. NET 6*. Packt Publishing Ltd.

Del Sole, A., 2022. Xamarin with Visual Studio: *Launch your mobile development career by creating Android and iOS applications using. NET and C# (English Edition)*. BPB Publications.

Golmohammadi, A., Zhang, M. and Arcuri, A., 2023. . *NET/C# instrumentation for search-based software testing*. Software Quality Journal, 31(4), pp.1439-1465.

Marcotte, C.H., 2024. *Architecting ASP. NET Core Applications: An atypical design patterns guide for. NET 8, C# 12, and beyond*. Packt Publishing Ltd.