CA1 – Project Report

Student: Guilherme Silveira
Student Number: X23413271
Program: HDCSDEV_INT
Module: Algorithms and Advanced Programming (AAP)
Project Part: Part 1 - Sorting and Searching Algorithm Analysis

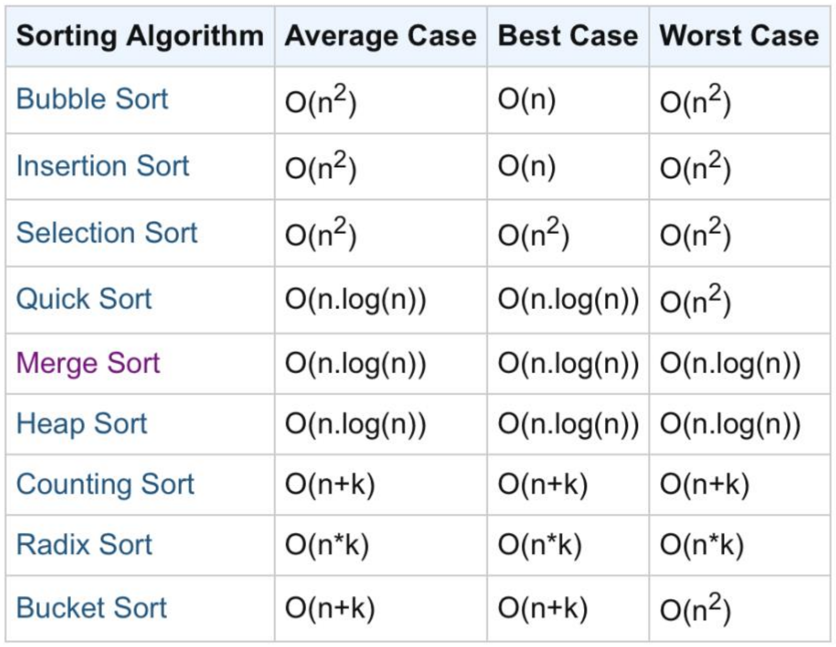
Introduction

This project involves working on a dataset of 10,000 films, implementing and analysing sorting and searching algorithms, as well as measuring their time complexity.

Q1. Sorting Algorithm Implementation

Algorithm Chosen: Merge Sort

The following table compares the most common sorting methods.



Sorting Algorithm	Average Case	Best Case	Worst Case
Bubble Sort	$O(n^2)$	O(n)	$O(n^2)$
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	O(n.log(n))	O(n.log(n))	$O(n^2)$
Merge Sort	O(n.log(n))	O(n.log(n))	O(n.log(n))
Heap Sort	O(n.log(n))	O(n.log(n))	O(n.log(n))
Counting Sort	O(n+k)	O(n+k)	O(n+k)
Radix Sort	O(n*k)	O(n*k)	O(n*k)
Bucket Sort	O(n+k)	O(n+k)	$O(n^2)$

Figure 1 - Comparison of Sorting Algorithms (Code Project, 2021)

Merge Sort was chosen because it is an efficient and stable sorting method, especially for large datasets, which is the case. Its time complexity remains O(n log n) for best, average and worst case scenarios. Therefore, it seemed to be the best option.

Films were sorted by Title. If two or more films had the same title, their `filmID` was used as a secondary criterion for the sorting.

Q2. Sorting Time Complexity Analysis

The Merge Sort algorithm was tested with datasets of the following sizes: 10, 100, 1,000, 5,000 and 10,000 films. Each size was tested three times and the average time (in nanoseconds) was calculated.

Merge Sort Results:

Number of Records	Average Time (ns)
10	1041.0
100	7736.0
1000	76805.0
5000	374972.0
10000	976861.0

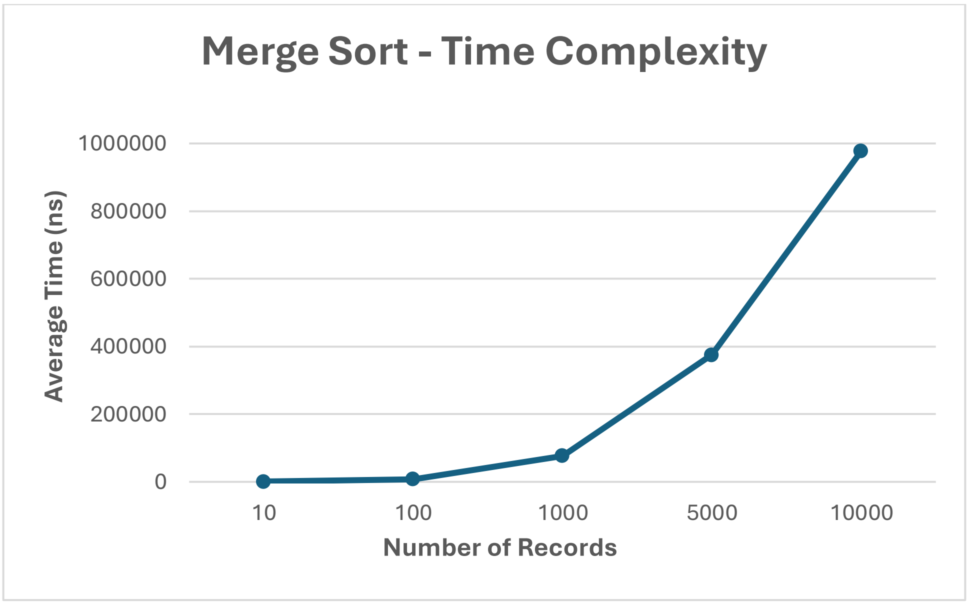


Figure 2 - Merge Sort Time Complexity. Results plotted using Excel charts

Observation: The trend confirms the expected O(n log n) time complexity.

Q3. Searching Algorithm Implementation

Algorithm Chosen: Binary Search

A modified binary search was implemented to find all films matching a specific title in the sorted array. Once the match is found, the algorithm expands left and right from the match to capture any duplicates. If the title is not found, the program displays "`Not an existing film title`". It is assumed that the dataset is clean and sorted before performing search operations.

This method was chosen because it can be much faster than the linear search, since it does not have to traverse the whole array to find the targeted element.

Q4. Searching Time Complexity Analysis

Just like the Merge Sort method, this algorithm was also tested with various datasets of increasing sizes (10,100, 1,000, 5,000 and 10,000). Each test was run three times and the average time was calculated.

Binary Search Results:

Number of Records	Average Time (ns)
10	6583.0
100	4166.0
1000	4111.0
5000	6875.0
10000	7194.0

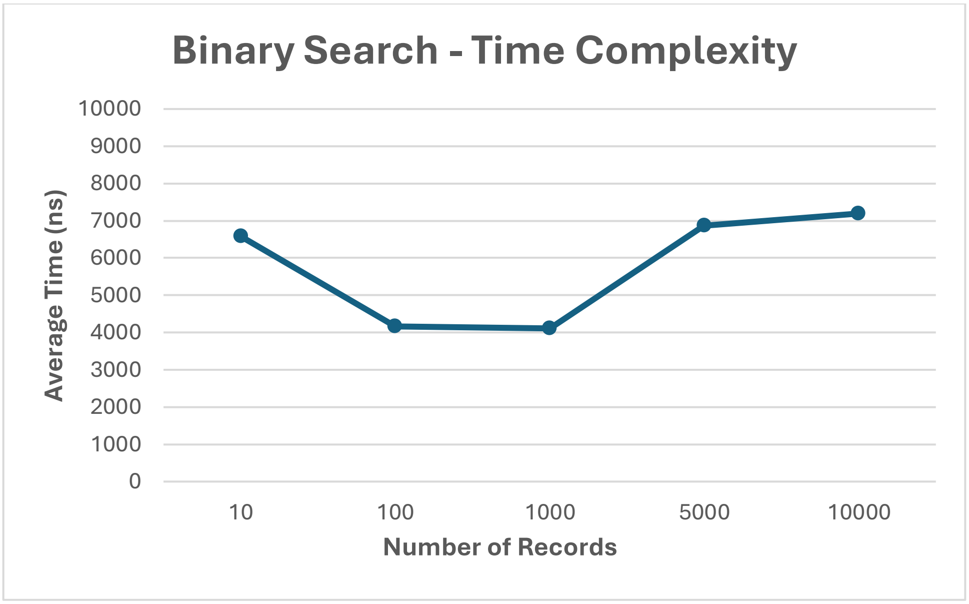
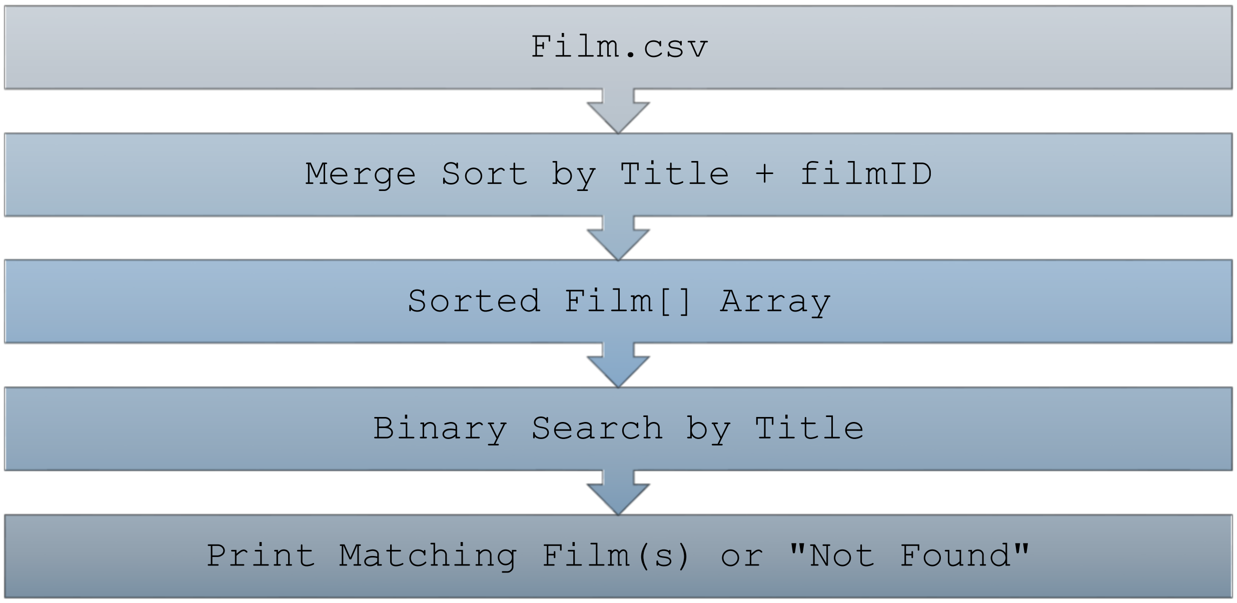


Figure 3 – Binary Search Time Complexity. Results plotted using Excel charts

IPO Diagram

Step	Description
Input	Film Dataset (10,000 records in CSV format)
Process	Merge Sort by title, then Binary search by title
Output	Sorted film list and printed search results



Film.csv
Merge Sort by Title + filmID
Sorted Film[] Array
Binary Search by Title
Print Matching Film(s) or "Not Found"

Appendix

Java Classes included:

- `Film.java`
- `FilmSorter.java`
- `FilmSearcher.java`
- `ReadFilmData.java`
- `Main.java`

Dataset: `Film.csv`

Bibliography

Code Project, 2021. *Comparison of Sorting Algorithms*. [Online]
Available at: https://www.codeproject.com/Articles/5308420/Comparison-of-Sorting-Algorithms
[Accessed March 2025].