Marking Criteria Sheet

Student ID: **Student Name:** **Marker:**

Total Marks (%):

ADT Implementations (50%)

ADT/Methods	Full marks (%)	Marks (%)
Movie		
• *CompareTo*	**3**	
• *ToString*	**3**	
MovieCollection		
• *IsEmpty*	**2**	
• *Insert*	**8**	
• *Delete*	**8**	
• *Search*	**8**	
• *NoDVDs*	**8**	
• *ToArray*	**8**	
• *Clear*	**2**	

Technical Report (50%)

Criteria	Full marks (%)	Marks (%)
Algorithm design		
• Finding the total number of DVDs in the movie collection (in the library)	**10**	
Algorithm analysis		
• Finding the total number of DVDs in the movie collection (in the library)	**20**	
Software test plan, test data and test results	**18** (9 methods; 2% each)	
Presentation	**2**	

Marking Criteria

ADT method implementation	• Preconditions: Preconditions must be met before an operation can be performed on an ADT. • Postconditions: Postconditions must hold true after an operation is performed on an ADT. • Invariants: Invariants must hold true for an ADT at all times, regardless of the operations that are performed on it. • Performance: The class method should be designed with performance in mind, avoiding unnecessary overhead and using efficient algorithms and data structures where possible. • Behaviour: The behaviour of the class method should be consistent with its underlying algorithm, if appliable
Algorithm design	• Use of pseudocode: Correct use of the pseudocode notations • Correctness: The algorithm should be designed to solve the problem it is intended to solve, and it should produce the correct output for all possible input values. This requires careful consideration of the problem domain and an understanding of the desired outcomes. • Efficiency: The algorithm should be designed to be as efficient as possible in terms of time and space complexity. This means that it should be optimized to run quickly and use minimal memory resources. • Robustness: The algorithm should be designed to be robust and tolerant of unexpected inputs or errors. This means that it should handle errors gracefully, provide informative error messages, and allow for easy recovery from errors. • Simplicity: The algorithm should be designed to be as simple and easy to understand as possible. This means that it should avoid unnecessary complexity or optimization that could make it difficult to understand or modify over time. • Reusability: The algorithm should be designed to be reusable in different contexts or applications. This means that it should be

	general enough to be applied to similar problems, and modular enough to be integrated into larger systems or workflows.
Algorithm analysis	• Experimental setup: The experimental setup should be designed to eliminate biases and ensure reproducibility. It should include a clear specification of the hardware and software used, the methodology for generating input data, and the criteria for measuring performance. • Algorithm implementation: The algorithm is implemented correctly. • Output: The experimental results is presented in the report. • Analysis result: The empirical analysis result is correct.
Software test plan and test results	• Test coverage: The test plan should ensure that all aspects of the software system are covered by the testing, including functional requirements, non-functional requirements, and boundary cases. • Test objectives: The test plan should have clear objectives that are aligned with the project goals and objectives. The test objectives should be measurable, specific, achievable, and relevant to the project. • Test methods: The test plan should specify the methods to be used for testing, such as manual testing, automated testing, exploratory testing, or a combination of these methods. • Test environment: The test plan should define the test environment, including the hardware, software, and network configurations, as well as any necessary test data and test cases. • Test results: The test plan should document the results of testing, including any defects found, their severity, and their resolution status. The test results should be analysed to identify any trends or patterns and to inform decisions about software quality and readiness for release.

Presentation	• Clarity: The report should be written in clear and concise language, using appropriate technical terminology and notation. • Discussion and interpretation: The report should include a discussion and interpretation of the results, explaining their significance and implications in the context of the project or study objectives. The discussion should be based on sound reasoning and supported by evidence. • Structure: The structure of the report should be logical and easy to follow, with clearly defined sections and headings, and should include a cover page, a table of contents and references.