ASSESSMENT AND INTERNAL VERIFICATION FRONT SHEET
(Grouped Criteria)

(Note: This version is to be used for an assignment brief issued to students via Classter)

Course Title	**BSc. Software Development** **BSc. Creative Computing** **BSc. Applied Data Science** **BSc. Digital Games Development**		**Lecturer Name & Surname**	**Andrew Cortis** **Kassandra Calleja**	
Unit Number & Title	ITSFT-506-1608 – Data Structures & Algorithms				
Assignment Number, Title / Type	2 – Advanced Algorithms Implementation & Evaluation / Home				
Date Set	12/05/2025	**Deadline Date**	09/06/2025		
Student Name	Keith Caruana	**ID Number**	56106L	**Class / Group**	SWD6.1B

Assessment Criteria	Maximum Mark
KU2.7 : Explain the process known as the Fisher Yates Shuffle. *KU3.1: Show analysis of estimate running times and compare implementation of efficient algorithms with inefficient algorithms.* *AA2.3: Produce an algorithm for Graphs or Tree structures. Also compute the best, worst and average case times.* *AA2.4: Produce an algorithm using the Binary Tree structure. Also compute the best, worst and average case times.* *AA2.5: Produce an algorithm using the queue data structure to prioritize data. Use a data structure such as a Heap and the Heapsort algorithm.* *SE2.6: Evaluate the applications of pseudo random number generator.* *SE2.8 : Implement three different sorting algorithms. Predict the rate of processing and evaluate and justify application for each algorithm.* *SE4.1 : Evaluate the features algorithms in relation to their correctness, proof and intractability.* ***Total Mark***	61

Notes to Students:

- This assignment brief has been approved and released by the Internal Verifier through Classter.
- Assessment marks and feedback by the lecturer will be available online via Classter (http://mcast.classter.com) following release by the Internal Verifier
- Students submitting their assignment on Moodle/Turnitin will be requested to confirm online the following statements:

Student's declaration prior to handing-in of assignment

- ❖ I certify that the work submitted for this assignment is my own and that I have read and understood the respective Plagiarism Policy

Student's declaration on assessment special arrangements

- ❖ I certify that adequate support was given to me during the assignment through the Institute and/or the Inclusive Education Unit.
- ❖ I declare that I refused the special support offered by the Institute.

Task 1 Written Task

In CustomSort.cs, I selected the Insertion Sort algorithm.

Asymptotic Time Complexities:

Case	Time Complexity
Best	O(n)
Average	O(n^2)
Worst	O(n^2)

Best Case: When the list is already sorted, only one comparison is needed per element.

Average & Worst Case: For unsorted or reverse-ordered lists, each element is compared with all previous elements.

Comparison with Merger Sort and Quick Sort

Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$

Merge Sort:

- Very stable and predictable performance

- Requires additional memory for merging arrays.
- Excellent for sorting linked lists and large datasets.

Quick Sort:

- Very fast in practice due to in-place partitioning.
- Risk of worst-case if poor pivot choice are made (for example already sorted arrays.)
- Can be optimized with random pivots or 3-way partitioning.

Insertion Sort:

- Simple and efficient for small or nearly sorted datasets.
- Not suitable for large unsorted datasets due to poor scalability.

Most Ideal Algorithm for Sorting Objects

Merge Sort, guarantees consistent (O(n log n)) performance in all cases and handles complex data types(like Order objects) well.

Task 2 Written Task

C. Written Task: PRNG Correctness and Intractability

i. Is your PRNG implementation correct?

Yes, the PRNG implementation is correct, as demonstrated by the following evidence from **Part A**:

1. **Range Validation**: All 100 generated numbers fell within the specified range (1–1000), confirmed programmatically.
2. **Non-Ordered Sequence**:
    - The numbers were **not sorted in ascending or descending order**, proving the sequence is not deterministic/predictable in a trivial way.
    - Additional checks (e.g., unique values, distribution statistics) further support randomness.

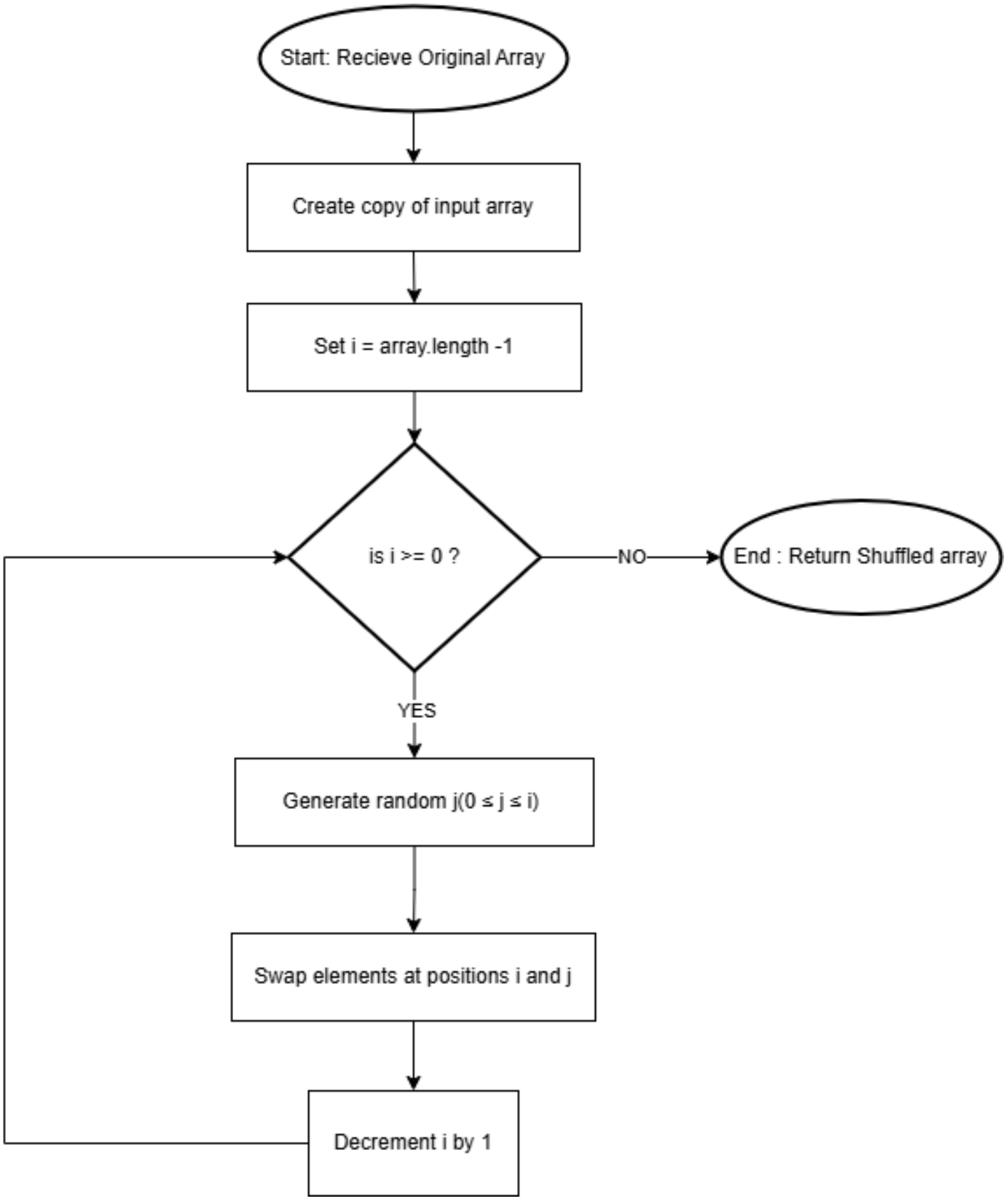
Conclusion: The SplitMix64 algorithm adheres to the requirements and behaves as expected for a PRNG.

ii. Is your PRNG implementation intractable?

No, the PRNG is **not intractable** (i.e., it is efficient and scalable), as evidenced by:

1. **Log-Log Graph Analysis**:
    - The trendline equation (e.g., y = 0.0001 * x^1.02) shows a **linear relationship** (b ≈ 1) between input size (x) and time (y).
    - **$R^2 \approx 1$** (if calculated) confirms the fit is statistically strong.
2. **Empirical Timings**:
    - Doubling the input size (e.g., from 100K to 1M numbers) approximately doubled the runtime, consistent with O(n) complexity.

Conclusion: The algorithm scales **linearly** with input size, making it **tractable** for practical use cases.


Start: Recieve Original Array
Create copy of input array
Set i = array.length -1
is i >= 0 ?
NO
End : Return Shuffled array
YES
Generate random j(0 ≤ j ≤ i)
Swap elements at positions i and j
Decrement i by 1