|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | Aiham Anter | | **ID Number** | 25251a |  | **Class /**  **Group** | SWD6.1B |

**WRITTEN TASK FOR CHOSEN SORTING ALGORITHM IN ( C )**  
  
For part C (Sort by Delivery Date), I used the built-in List.Sort method and added my own comparer to sort by the delivery date.

**Sorting Algorithm Used:**

The built-in List.Sort uses IntroSort, which is a mix of QuickSort, HeapSort[3], and Insertion Sort.

**Time Complexities:**

* Best Case: O(n log n)
* Average Case: O(n log n)
* Worst Case: O(n log n)

**Comparison with Merge Sort and Quick Sort:**

**Merge Sort[1]:**

* Time: Always O(n log n)
* Stable, but needs O(n) extra memory.

**Quick Sort[2]:**

* Time: O(n log n) average, O(n²) worst
* Works in-place but not stable.

**Built-in Sort (IntroSort):**

* Keeps worst-case at O(n log n), like Merge Sort
* Usually faster than plain Merge Sort
* Sorts in-place like Quick Sort but avoids its worst-case**.**

**Why I Chose This One:**

* I think the built-in sort is the best option here because:
* It guarantees O(n log n) time
* It’s already optimized and tested in .NET
* It doesn’t need extra memory like Merge Sort
* It avoids the slow cases of Quick Sort
* And honestly, it’s easier and more reliable to use something that’s already built-in.

**C. WRITTEN TASK FOR PRNG CORRECTNESS AND INTRACTABILITY**

**1. Is your PRNG[4] implementation correct?**

Yes, I believe it’s correct based on these test results:

* All the numbers it generated were between 1 and 1000, like expected.
* The numbers didn’t come out sorted, either ascending or descending.
* I followed the SplitMix64 algorithm exactly as shown in the pseudocode.

**2. Is your PRNG implementation intractable?**

No, it’s not intractable. Here's why:

* The time it takes grows in a straight line with the number of random values I generate.
* The log-log graph shows a straight line with a slope around 1, which means it's linear (O(n)).
* Even when generating 1,000,000 numbers, it handled it fine without slowing down a lot.
* There’s no sign of exponential time, so the performance is predictable and practical.

**References:**

[1] Pankaj, “Merge Sort Algorithm - Java, C, and Python Implementation | DigitalOcean,” *www.digitalocean.com*, Aug. 04, 2022. <https://www.digitalocean.com/community/tutorials/merge-sort-algorithm-java-c-python>

[2]Tutorialspoint, “Data Structure and Algorithms - Quick Sort - Tutorialspoint,” *Tutorialspoint.com*, 2020. <https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm>

[3]Maaneth De Silva, “Heaps and Heapsort - Simply Explained,” *YouTube*, Jan. 26, 2023. https://www.youtube.com/watch?v=pY-cH7rti4U (accessed Jun. 09, 2025).

[4]Y. Singla, “Pseudo Random Number Generator (PRNG) - GeeksforGeeks,” *GeeksforGeeks*, Jun. 27, 2017. https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/

‌

‌ [5]HamzaAhmedPasha, “GitHub - HamzaAhmedPasha/IntroSort--Algorithm: INTRO SORT ALGORITHM IN C#,” *GitHub*, 2019. https://github.com/HamzaAhmedPasha/IntroSort--Algorithm (accessed Jun. 09, 2025).

‌