# TEB1113/TFB2023:Algorithm and Data Structure

**September 2024**

REPORT

Group:

1. See Tho Soon Yinn (24000187)
2. Avinash Kumar A/L Jayaseelan (24000113)
3. Ashwin A/L Ravichandran (22012188)
4. Syukri Fadhli bin Ahmad (24000074)
5. Hamzah Muhsin Bin Hafiz Al-Asadi (22001057)
6. Aiman Naim bin Shaiful Zahrin (22005653)

# Performance Comparison of Unity Drone Flocking Program Across Three Devices

## 1. Introduction

This report aims to compare the performance of a Unity drone flocking program executed on three different devices. The key criteria analysed include device specifications, the number of drones, and the response times of various functions tested during the program's execution. Understanding the relationship between hardware capabilities and program performance is crucial for optimising future implementations.

## 2. Device Specifications

The following table summarises the specifications of the devices used for testing:

| Device Name | Processor | RAM | System Type |
|---|---|---|---|
| MephilesVictus | AMD Ryzen 5 5600H (3.30 GHz) | 8 GB | 64-bit OS, x64 CPU |
| GODMODE | AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz | 32.0 GB | 64-bit operating system, x64-based processor |
| DESKTOP-TGD37EC | AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz | 16.0 GB | 64-bit operating system, x64-based processor |

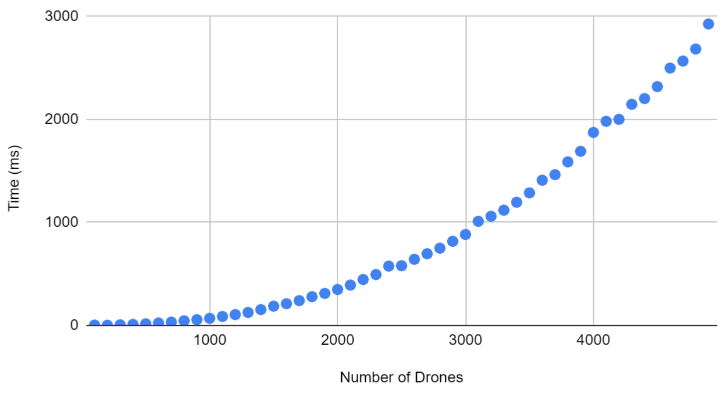
The device specifications highlight the differences in processing power and memory that could influence the execution of the drone simulation program.

Ranking: GODMODE > DESKTOP-TGD37EC > MephilesVictus

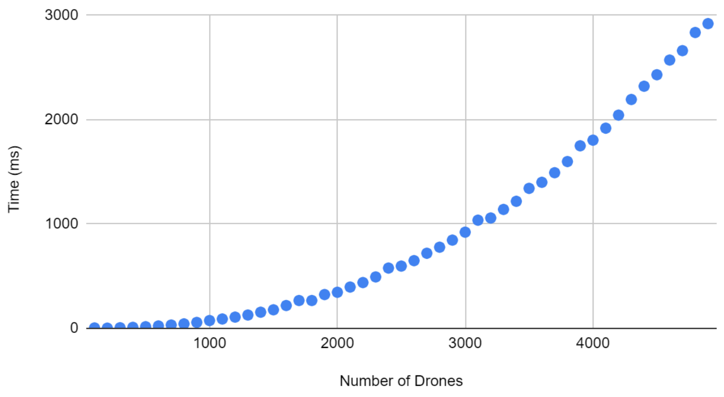
# 3. Data Comparison: Six Different Sorting Functions

## 3.1 *Device I - MephilesVictus*

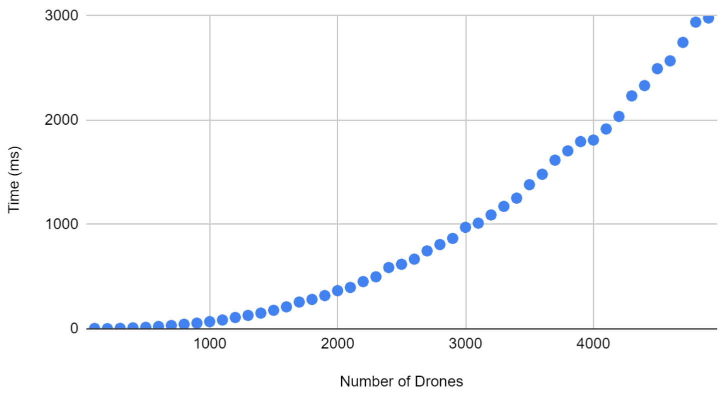
**Bubblesort**



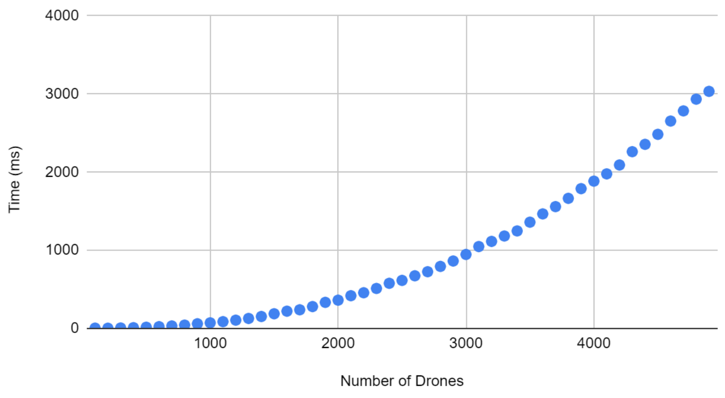
**Append**



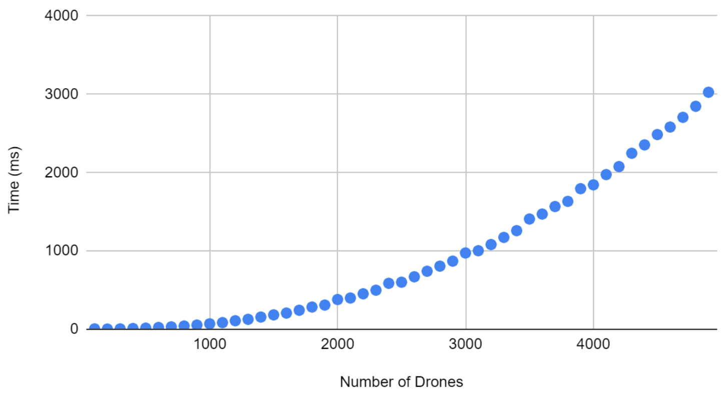
**AppendFront**



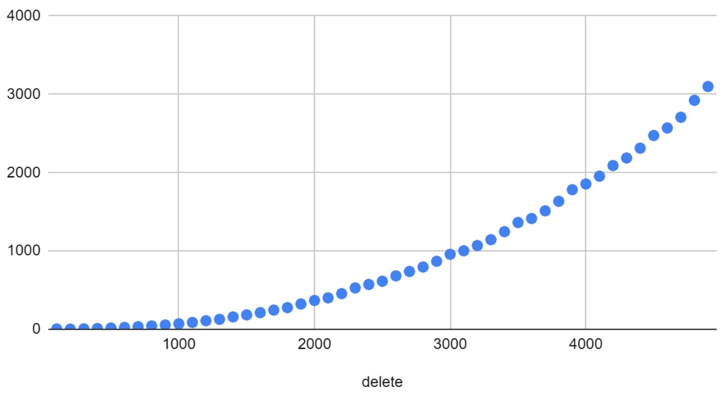
**Insert**



**DeleteFront**

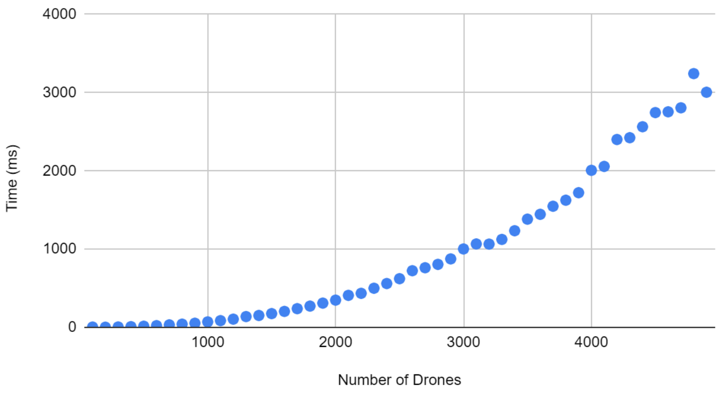


**Delete**

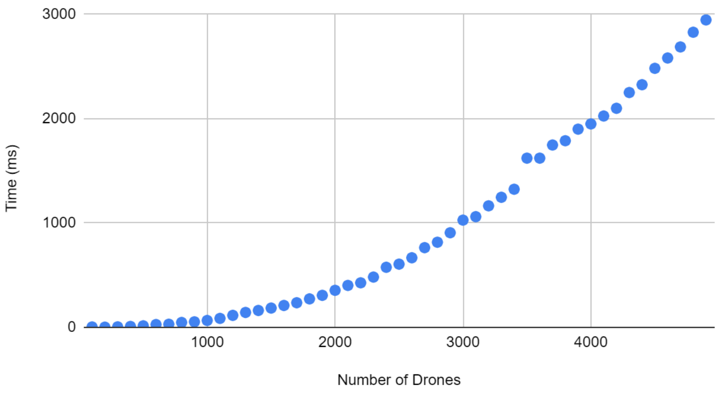


## 3.2 *Device II - GODMODE*

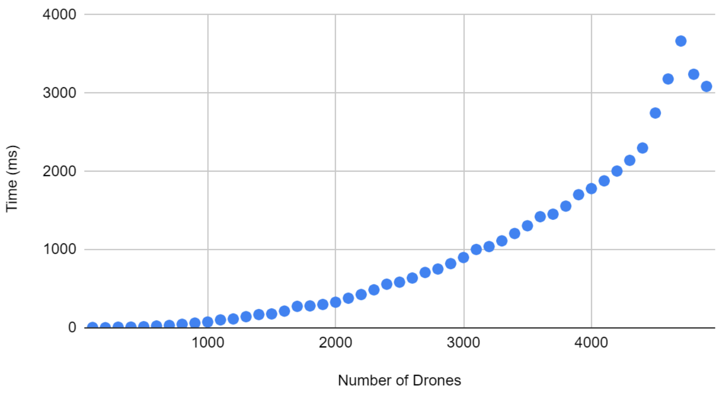
**Bubblesort**



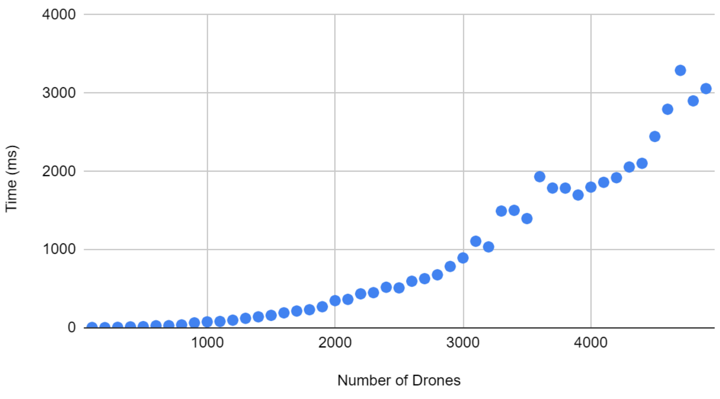
**Append**



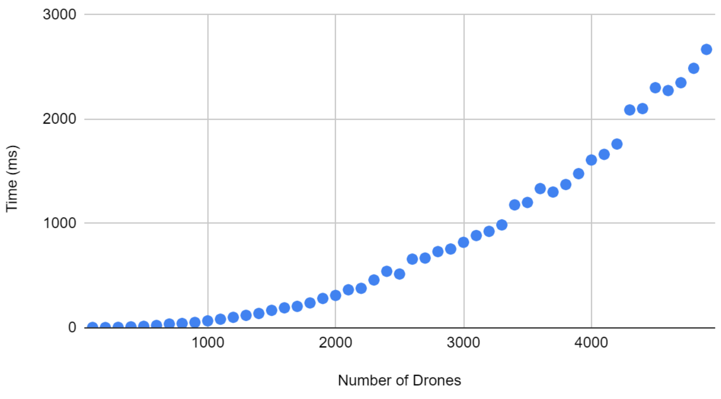
**AppendFront**



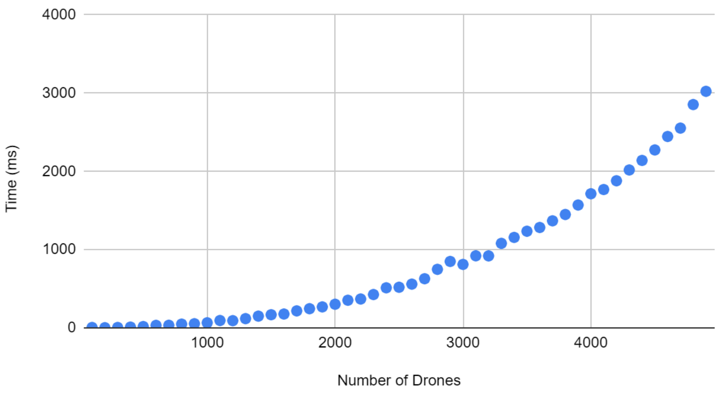
**Insert**



**DeleteBack**

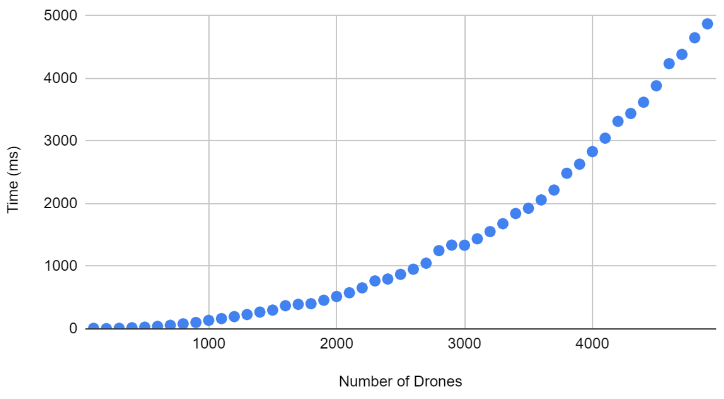


**DeleteBack**

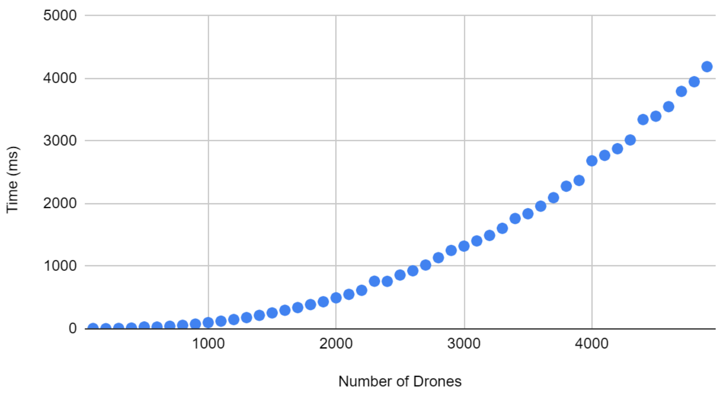


## 3.3 *Device III - DESKTOP-TGD37EC*

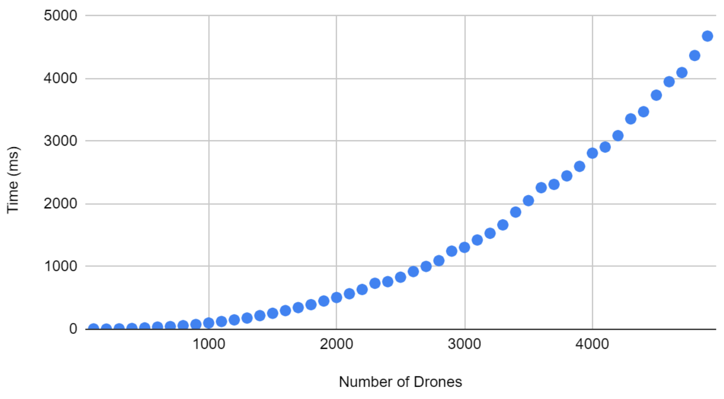
**Bubblesort**



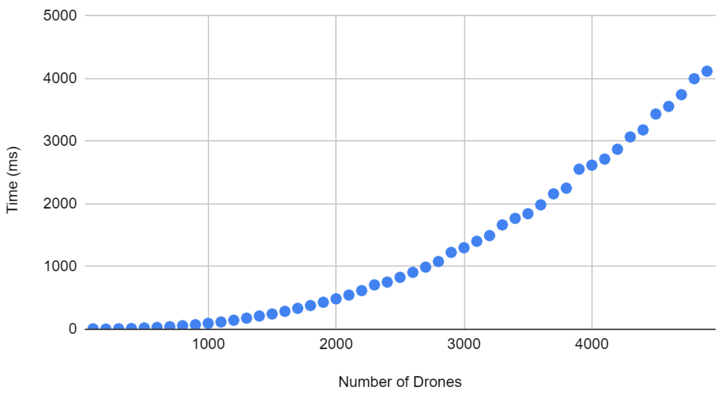
**Append**



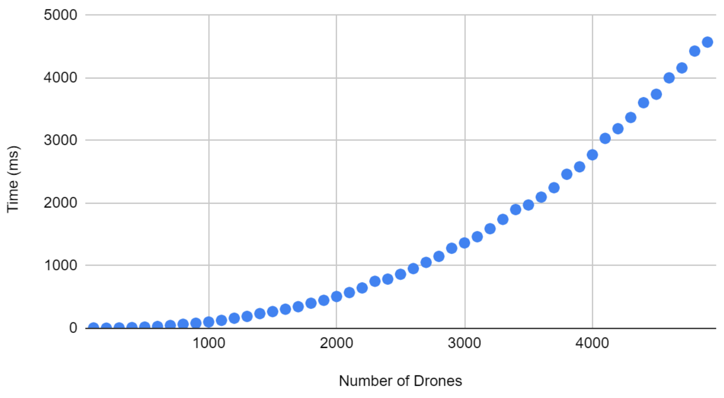
**AppendFront**



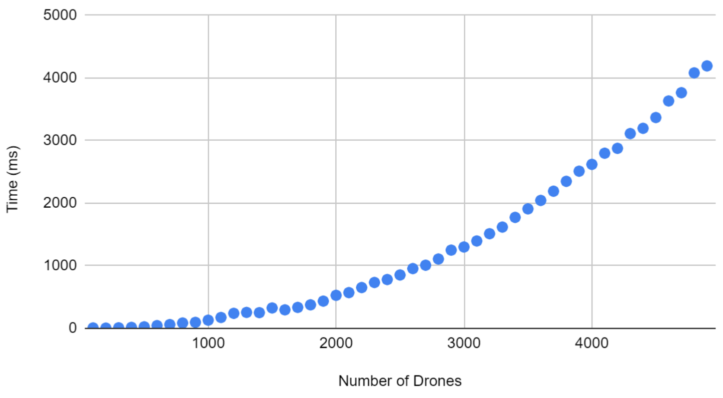
**Insert**



**DeleteFront**



**DeleteBack**



# 4. Discussion and Analysis

## *Analysis Device 1 - MephilesVictus*

BubbleSort should be the slowest yet it is showing similar times as other sorting functions despite its $O(N^2)$ complexity, there are several potential reasons:

1. Small Input Size: BubbleSort's quadratic time complexity becomes noticeable only with large input sizes. For small arrays, the difference between $O(N^2)$ and other algorithms (like $O(N\log N)$) might not be significant, especially given overheads like function calls or memory operations.
2. Hardware & Performance Overhead: Performance timings in environments like Unity can be influenced by factors like background processes, garbage collection, or how Unity manages updates, making differences between sorting algorithms less apparent.
3. Optimised Input: If the arrays being sorted are already partially sorted or have fewer elements out of order, BubbleSort can perform faster than its worst-case time complexity. This could explain why the difference in execution time is not as large as expected.
4. Performance Noise: If other parts of the code (like the flock behaviour simulation) are consuming significant time, the time taken by sorting might get masked, making all sorting algorithms seem to take roughly the same time.

Isolating the sorting function and testing it on larger datasets will portray a more pronounced difference in performance.

## *Analysis Device 2 - GODMODE*

In summary, the performance of different operations on the **GODMODE** device shows minimal differences with small datasets due to overhead factors like memory management. System processes and background tasks may impact the timing of operations like Insert and AppendFront, making them appear less efficient.

1. Small Input Size:For smaller datasets, the differences in performance between operations like BubbleSort (O(N2)O(N^2)O(N2)) and Append (O(1)O(1)O(1)) are less noticeable. This is likely due to overheads like memory allocation and function calls, which diminish the observable time complexity differences.

2. Hardware & Performance Overhead:System processes such as garbage collection and memory management can obscure performance. Operations like Insert and AppendFront may seem less efficient due to these environmental factors, rather than their inherent complexity.
3. Optimised Input: If the dataset is already partially sorted, operations like Insert and DeleteBack may perform better than expected. For instance, BubbleSort might not show its worst-case behaviour if the input is already close to being ordered.
4. Performance Noise:Irregularities in operations like Insert suggest external factors (e.g., background processes) are affecting performance. This noise can mask the true cost of each operation, making different algorithms appear to scale similarly as input size increases.

   Larger datasets and isolated testing of each operation are needed to better observe the true performance differences between these algorithms.

## *Analysis Device 3 - DESKTOP-TGD37EC*

Here's a analysis for Device III - DESKTOP-TGD37EC based on the performance charts:

1. Small Input Size:For small numbers of drones, time differences between operations like BubbleSort, Insert, and DeleteBack are minimal. Overheads like memory management and function calls obscure the actual time complexity for small datasets.

2. Hardware & Performance Overhead:System processes such as garbage collection and memory allocation affect operations like Insert and AppendFront, making them appear less efficient due to background memory management tasks.

3. Optimised Input: If the dataset is partially sorted, operations like Insert and DeleteBack perform better than expected. BubbleSort may not reach its worst-case behaviour with optimised input, narrowing the time differences between operations.

4. Performance Noise:Irregularities in operations like Insert and DeleteFront, especially around 3000-4000 drones, are due to external factors like memory reallocation and background processes, causing performance spikes.

Conclusion:Performance on DESKTOP-TGD37EC is influenced by system overheads and optimised input. Larger datasets and isolated testing are needed to reveal the true performance of each operation.

## 5. Conclusion

Putting it in  a nutshell, the dataset is too small to visualise the true complexities of the functions. Although all the graphs show an increase in processing time as the number of drones increases, BubbleSort in particular should show the highest gradient as it has the highest complexity O(N^2), while functions like append and deleteFront should not show any increase. However, the complexity of the Start() function itself when taking into account when the watch starts and stops. Would plot a graph of exponential increase for all functions for this code.