Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Documentation

(Group Work)

Student Information:

Surname:	**Given Name:**	**Student ID Number:**
Micky	Micky Malvino Kusandiwinata	2602174522
Christoffer	Christoffer Raffaelo Wijaya	2602177051
Louis	Louis Ruby Elsalim	2602183451

Course Code : COMP6048001 **Course Name :** Data Structure

Class **:** L2CC **Lecturer** **: Dr.MARIA SERAPHINA ASTRIANI , S.kom., M.T.I**

Type of Assignments: Final Project Documentation / Report

Submission Pattern

Due Date **:** 21 June 20223 **Submission Date** **:** 18 June 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Micky Malvino Kusandiwinata

Louis Ruby Elsalim

Christoffer Raffaelo Wijaya

Table of Contents

Background

Sorting is a fundamental process that brings order and efficiency to collections of data. In this project, we aim to develop a versatile sorting system that enables users to organize their data effectively. Our system allows for manual input or reading data from text files and implements bubble sort algorithms using regular arrays and linked lists.

Sorting plays a vital role in various applications, from managing lists to optimizing search results. It enhances searchability, improves user experience, and supports data analytics. By creating an intuitive sorting system, we empower users to take control of their data, enhance decision-making processes, and boost productivity.

Through comprehensive testing with multiple systems and evaluation using predefined test cases, we demonstrate the reliability and efficiency of our sorting system. The results contribute to the broader understanding and practical application of sorting algorithms in relation to data structures. We have provided evidence of this in the following pages of this report.

Overall, our project aims to provide a user-friendly sorting solution that enhances data organization and promotes efficient data manipulation.

Problem Description

The problem addressed by this project is the need to sort a sequence of numbers in ascending order. The goal is to develop a program that allows users to input a list of numbers or read them from a file and obtain the sorted output. The program should utilize the bubble sort algorithm to arrange the numbers efficiently.

The specific requirements of the problem can be defined as follows:

- The program should provide options for users to input the numbers manually or read them from a file.
- In the case of manual input, the program should prompt the user to enter a sequence of numbers separated by commas.
- In the case of reading from a file, the program should read the numbers from the specified file (.txt file).
- The program should implement the bubble sort algorithm to sort the numbers in ascending order.
- After sorting, the program should display the original list of numbers and the sorted list.
- The program should measure and display the execution time of the sorting algorithm.
- The program should be implemented in two versions: one using an array data structure (**BubbleSortArray**) and the other using a linked list data structure (**BubbleSortLinkedList**).

By examining both implementations, we aim to compare the efficiency and performance of Bubble Sort on LinkedList and array data structures. We will evaluate factors such as complexity analysis for different input scenarios. Additionally, we will give a clear cut winner among them both when pitted against each other.

Overall, our goal is to provide insights into the practical application of sorting algorithms using different data structures and their impact on performance, scalability, and user experience. Through this project, we aim to contribute to the understanding of sorting algorithms and facilitate informed decision-making when selecting data structures for sorting tasks.

Project Specifications

(Software, libraries, and data structures used in the program)

Software: IntelliJ IDEA: The integrated development environment (IDE) used for Java development. The program used was made with Java Development Kit (JDK) version 17.0.6

Libraries:

- java.io.BufferedReader: Used to read text from a character-input stream.
- java.io.IOException: Used for handling input/output exceptions.
- java.io.InputStream: Represents an input stream, used for reading files.
- java.io.InputStreamReader: Reads bytes and decodes them into characters.
- java.util.Scanner: Reads user input from the console.
- java.io.FileReader: Reads text from files.
- java.util.LinkedList: Represents a linked list data structure.

Data Structures:

- Array: Used in the first code BubbleSortArray.java to store and manipulate a sequence of integers.
- LinkedList: Used in the second code BubbleSortLinkedList.java to store and manipulate a sequence of integers in the form of a linked list.

Solution: (Proposed data structures to solve the problem and further analysis)

To solve the problem of sorting a sequence of numbers, two different data structures are proposed: an array and a linked list.

The first code file, **BubbleSortArray.java**, implements the Bubble Sort algorithm using an array data structure. This approach allows users to input the number sequence either from a text file or through manual input.
In BubbleSortArray, the numbers are stored in an array. Arrays are a contiguous block of memory that allows efficient access to elements using their indices. For sorting, arrays provide direct access to elements based on their positions, making it easy to compare and swap elements during the sorting process. The Bubble Sort algorithm iterates through the array, compares adjacent elements, and swaps them if necessary until the entire array is sorted.

The second code file, BubbleSortLinkedList.java, demonstrates the Bubble Sort algorithm using a LinkedList data structure. This approach also supports reading the input number sequence from a text file or manual input. LinkedList is a dynamic data structure that offers easy insertion and deletion of elements, making it suitable for sorting larger or continuously changing datasets. In BubbleSortLinkedList, the numbers are stored in a linked list (java.util.LinkedList). Linked lists consist of nodes, where each node holds a value and a reference to the next node in the sequence. Unlike arrays, linked lists do not require contiguous memory and provide flexibility in adding, removing, and rearranging elements. The Bubble Sort algorithm is adapted to work with linked lists by traversing the list, comparing adjacent elements, and rearranging their positions if needed.

Both the array and linked list data structures provide a solution to the problem of sorting a sequence of numbers. However, they have different characteristics and trade-offs. Arrays offer direct element access and efficient memory usage but have a fixed size, while linked lists provide dynamic resizing and flexibility but require extra memory for storing node references. The choice between the two data structures depends on the specific requirements and constraints of the problem at hand.

For reference, the program includes 15 default test cases in the form of text files (.txt). Each test case consists of a specific number of integers within a given range. You can also add your own test cases to evaluate the program's performance or modify the existing test cases by removing some of the larger numbers. The test cases cover a range of different sizes and ranges of integers to test the program's capabilities.

Here are the 15 default test cases along with the number of integers in a set and the range of integers:

- test1: 10 (range: -1000 to 1000)
- test2: 50 (range: -5000 to 5000)

- test3: 100 (range: -10000 to 10000)
- test4: 250 (range: -25000 to 25000)
- test5: 500 (range: -50000 to 50000)
- test6: 750 (range: -75000 to 75000)
- test7: 1000 (range: -100000 to 100000)
- test8: 1500 (range: -150000 to 150000)
- test9: 2000 (range: -200000 to 200000)
- test10: 2500 (range: -250000 to 250000)
- test11: 3000 (range: -300000 to 300000)
- test12: 3500 (range: -350000 to 350000)
- test13: 4000 (range: -400000 to 400000)
- test14: 4500 (range: -450000 to 450000)
- test15: 5000 (range: -500000 to 500000)

In order to have these 15 test cases, we decided to use a number generator called Random.org. From their website we learn that random.org offers a unique and professional approach to generating random numbers for testing purposes. Unlike traditional pseudo-random number generators, Random.org taps into atmospheric noise by aiming antennas into deep space. This noise, known as the "static" between radio stations, serves as a source of true randomness. Collected from multiple sites worldwide, the data undergoes careful processing to amplify statistical anomalies. By leveraging this approach, Random.org provides random numbers of unparalleled unpredictability and statistical independence. Incorporating these truly random numbers into our testing process enhances the integrity and accuracy of evaluations, enabling robust assessment of sorting algorithms.

So the first reason we chose this website would be for its high quality randomness. But apart from that, we also chose it for its broad range of values. The random.org algorithm generates random numbers within the specified range, ensuring that the test cases cover a wide spectrum of values. This range diversity enables comprehensive testing of our programs, allowing us to assess their efficiency and accuracy across different ranges of input data.

In addition, the random number generation algorithm done by random.org ensures unbiased sampling. Each generated number is statistically unrelated to the others, which mimics the randomness found in real-world scenarios. This helps in evaluating the robustness and effectiveness of our program under various data patterns. The convenience factor of using random.org is also a big one. Eliminating the need for us to create our test cases with thousands of integers manually. The website provides an automated and efficient way to generate multiple test cases with specific range and size parameters. This saved us a great deal of time and effort.

*How we filled out the prompt in the Random.org website for test case 15

Random Integer Set Generator

This form allows you to generate random sets of integers. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs.

Step 1: The Sets

Generate 1 set(s) with 5000 unique random integer(s) in each.

Each integer should have a value between -50000(and 500000 (both inclusive; limits ±1,000,000,000).

The total number of integers must be no greater than 10,000.

Random Integer Set Generator

You requested 1 set with 5000 unique random integers, taken from the [-500000,500000] range. The integers were not sorted.

Here is your set:

Set 1: 127159, 378428, -424744, 405116, 228357, 218520, -200679, -34090, -48107, -408523, 387814,
-348900, -495335, 471659, 5824, -60734, 312702, -217682, -321237, 476591, 425124, -394974,
455234, 192017, 181795, 53986, -365296, 375261, 487922, -154262, -489293, -455803, -36269,
-284279, 447225, -440433, -303783, -459738, -12494, -400122, -316941, -475102, 24271, -497260,
-363252, -359133, 484480, 357652, 356372, -446202, -304311, -19232, -205011, 281889, -415570,
-267430, 67580, -170724, -278807, -8410, 108486, 498503, -131690, -19577, -384773, -169315,
361985, -299145, 372703, -227845, 159750, 83832, -177220, 392996, -6115, -376878, -324410,
-311935, 382153, 330706, -122718, -279258, -102907, -141191, -306862, -411931, -231473, 113704,
210770, -455567, 421414, 17990, 293967, 122704, 423415, -306446, 276753, 367391, 419029, 147588,
425429, -165035, -385428, 35544, 196802, -491085, -123023, -147517, 363922, 96980, 397407,
-398568, -217524, -228404, -447428, -411535, 179498, 287104, 237537, -77743, -472772, -276305,
-308991, 469514, -201747, 408430, 356508, -80857, -214517, -317498, 480535, -416871, -373804,
-151876, -392718, -231194, -177959, 322565, 473353, 255783, -278275, -194084, -438363, -17318,
356482, -190032, 322616, -448791, -295161, 294929, 463747, 367621, 348563, 327069, 274836, 39151,
18030, 44267, -174074, 449754, -137309, 466770, -254032, 338049, 359661, -187278, 384438,
-382425, 54706, -338874, -458670, -110755, -356036, -280701, -183851, 301784, 186566, -154077,
-499538, 181928, -488691, 118584, 373999, 322050, 415188, 92996, -347834, -403428, 457798,
173346, 87030, 496613, -24714, 461797, 143670, -104248, -406752, -116568, -447292, -136815,
355910, -3792, -321384, 482186, 466630, -13447, 201989, -334859, 66050, -449460, 413218, 282581,
470810, -376500, 94025, -185152, 137880, -497531, 51602, 17037, -434055, -495602, 174526, 197903,
260353, -282673, 416940, -16657, 393992, -229083, -213057, -473371, -176372, -409358, -9980,
-317533, 357073, 291180, 80152, 132148, -493924, -470986, 460947, -425789, 22809, 139184, 282292,
31194, 363362, 102702, -322704, 467860, 148424, 290176, 212514, 260747, -370669, -468455, 105996,
155187, -320105, -439064, -367137, 278497, 208781, -113756, 192539, 421128, 128242, 43115,
-212136, 217008, -414528, 277008, 343046, -465293, 95169, 227238, 180254, -116572, 388479, 35262,
-42057, -114106, -13717, -58018, -267339, 436583, 348990, 313273, -357821, 245942, 160278,
348021, -300350, -328603, 143754, 300952, 431415, -250580, 104288, -82423, -23879, 56135,
-491433, -30568, 99848, -272231, -18985, -122644, -323236, -410979, -158229, 160745, 65237,
463782, 218753, 248299, 363182, 118924, 170145, -297644, -5063, -139147, -117612, 463208, 480149,

Implementations

Sorting Using LinkedList: The code gives the user the choice of manually entering the numbers into the terminal or using a text file. The program reads the numbers from a file called "test.txt" to "test15.txt" and adds them to the LinkedList if the file option is chosen. The user can enter the numbers by using commas to separate them if the manual option is selected. The sorting algorithm, created specifically for LinkedLists, is then used by the code. As it goes over the list, it compares nearby elements and, if necessary, switches them. The reported sorted LinkedList is followed by the execution time in milliseconds and nanoseconds.

Sorting Using Array: It is similar to the LinkedList implementation, it offers options for inputting the numbers via a text file or manual input. When the file option is chosen, the application loads the data from the file "test.txt" to "test15.txt" and places it in the array. The user can input the numbers by using commas to separate them if the manual option is selected. The sorting technique for arrays is then used, which iterates through the array and switching elements as necessary. The execution time in milliseconds and nanoseconds is shown alongside the sorted array.

The first file, "BubbleSortArray.java," implements the bubble sort algorithm for sorting an array of integers. It provides methods for reading the array either from user input or from a file, performing the bubble sort operation, and printing the sorted array. Breakdown of the implementation:

- The class BubbleSortArray declares a static integer array variable array to hold the numbers to be sorted.

- The bubbleSort method implements the bubble sort algorithm, which compares adjacent elements and swaps them if they are in the wrong order. It iterates over the array multiple times until the array is sorted in ascending order.
- The printArray method is a utility method that prints the elements of the array.

- The readArrayFromInput method reads the number sequence from user input. It prompts the user to enter a comma-separated list of numbers, splits the input into individual numbers, converts each number from a string to an integer, and stores them in the array.

- The readArrayFromFile method reads the number sequence from a file named "test.txt" located in the resources folder. It loads the file as an input stream, reads the content line by line, splits each line into individual numbers, converts them from strings to integers, and stores them in the array.

- The main method serves as the entry point of the program. It prompts the user to choose between reading the number sequence from a file or manually entering it. Based on the user's choice, it calls the corresponding input method (readArrayFromFile or readArrayFromInput) to populate the array. Then it prints the original array, performs the bubble sort operation using the bubbleSort method, prints the sorted array, and measures the execution time.

The second file, "BubbleSortLinkedList.java," implements the bubble sort algorithm for sorting a linked list of integers. It provides methods for reading the list either from user input or from a file, performing the bubble sort operation, and printing the sorted list. Breakdown of the implementation:

- The class “BubbleSortLinkedList” declares a LinkedList<Integer> variable numbers to hold the numbers to be sorted.

- The main method serves as the entry point of the program. It prompts the user to choose between reading the number sequence from a file or manually entering it. Based on the user's choice, it calls the corresponding input method (readNumbersFromFile or readNumbersFromManualInput) to populate the linked list. Then it prints the original list, performs the bubble sort operation using the bubbleSort method, prints the sorted list, and measures the execution time.
- The “readNumbersFromFile” method reads the number sequence from a file named "test.txt." It reads the file line by line, splits each line into individual numbers, converts them from strings to integers, and adds them to the linked list.

- The “readNumbersFromManualInput” method reads the number sequence from user input. It prompts the user to enter a comma-separated list of numbers, splits the input into individual numbers, converts each number from a string to an integer, and adds them to the linked list.

- The “bubbleSort” method implements the bubble sort algorithm for linked lists. It iterates over the list multiple times, comparing adjacent elements and swapping them if they are in the wrong order. The sorting is performed by swapping the values of the nodes in the linked list.

- The program measures the execution time by recording the start time before the bubble sort operation and the end time after the operation. It then calculates the duration and prints it out.

Evidence Of Working System [Micky's system]

Bubble Sort with **Array - manual** method

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent
Do you want to print the sorted number sequence via text (F)ile or (M)anually? m
Enter the number sequence (comma-separated numbers): 5, 6, 10, 1, 3, -2
Original array:
5 6 10 1 3 -2

Sorted array:
-2 1 3 5 6 10

Time taken: 5583 nanoseconds
Time taken: 0 milliseconds

Process finished with exit code 0
```

Bubble Sort with **Array - automatic** method via text file (**test.txt**)

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent
Do you want to print the sorted number sequence via text (F)ile or (M)anually? f
Original array:
-435 -667 225 -131 -666 840 -37 -173 -797 -321

Sorted array:
-797 -667 -666 -435 -321 -173 -131 -37 225 840

Time taken: 7333 nanoseconds
Time taken: 0 milliseconds

Process finished with exit code 0
```

How BubbleSortArray.java reads input from file, experiment with different test cases!

```
// Method to read array from a file
1 usage
private static int[] readArrayFromFile() {
    try {
        // Load the file from resources folder
        InputStream is = BubbleSortArray.class.getResourceAsStream( name: "test.txt");
        if (is == null) {
            System.out.println("Failed to locate the file.");
            return null;
        }
```

Handling user input for Array

```
Do you want to print the sorted number sequence via text (F)ile or (M)anually? 100
Invalid input. Exiting the program.
```

```
    if (input.equalsIgnoreCase( anotherString: "F")) {
        // Read array from file
        array = readArrayFromFile();
        if (array == null) {
            System.out.println("Error reading array from file.");
            return;
        }
    } else if (input.equalsIgnoreCase( anotherString: "M")) {
        // Read array from user input
        array = readArrayFromInput();
    } else {
        System.out.println("Invalid input. Exiting the program.");
        return;
    }
```

Bubble Sort with **LinkedList - manual** method

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Appli
Do you want to print the sorted number sequence via text (F)ile or (M)anually?
m
Enter numbers separated by commas:
0, 1, 10, 100, -50, 6, -3
Original list: [0, 1, 10, 100, -50, 6, -3]

Sorted list: [-50, -3, 0, 1, 6, 10, 100]

Time taken: 55125 nanoseconds
Time taken: 0 milliseconds

Process finished with exit code 0
```

Bubble Sort with **LinkedList - automatic** method via text file (**test.txt**)

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Appli
Do you want to print the sorted number sequence via text (F)ile or (M)anually?
f
Original list: [-435, -667, 225, -131, -666, 840, -37, -173, -797, -321]

Sorted list: [-797, -667, -666, -435, -321, -173, -131, -37, 225, 840]

Time taken: 118208 nanoseconds
Time taken: 0 milliseconds

Process finished with exit code 0
```

How BubbleSortLinkedList.java reads input from a file, put different test cases here!

```
private static LinkedList<Integer> readNumbersFromFile() {
    LinkedList<Integer> numbers = new LinkedList<>();

    try {
        InputStream is = BubbleSortLinkedList.class.getResourceAsStream( name: "test.txt");
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        String line;
```

Error handling aspect for BubbleSortLinkedList.java

```
Do you want to print the sorted number sequence via text (F)ile or (M)anually?
G
Invalid option selected. Exiting the program.
```

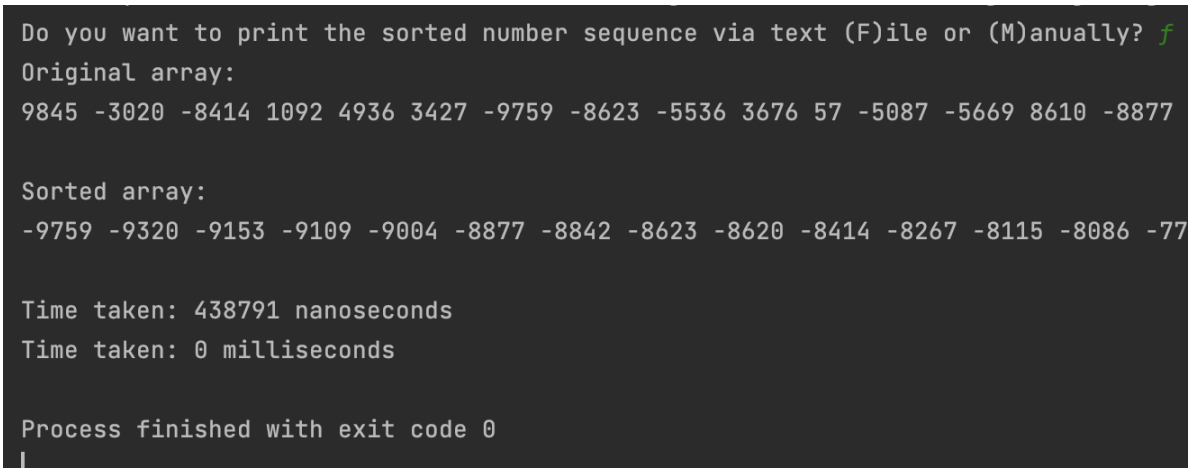
```
switch (option) {
    case 'F':
    case 'f':
        numbers = readNumbersFromFile();
        break;
    case 'M':
    case 'm':
        numbers = readNumbersFromManualInput();
        break;
    default:
        System.out.println("Invalid option selected. Exiting the program.");
        System.exit( status: 0);
}
```

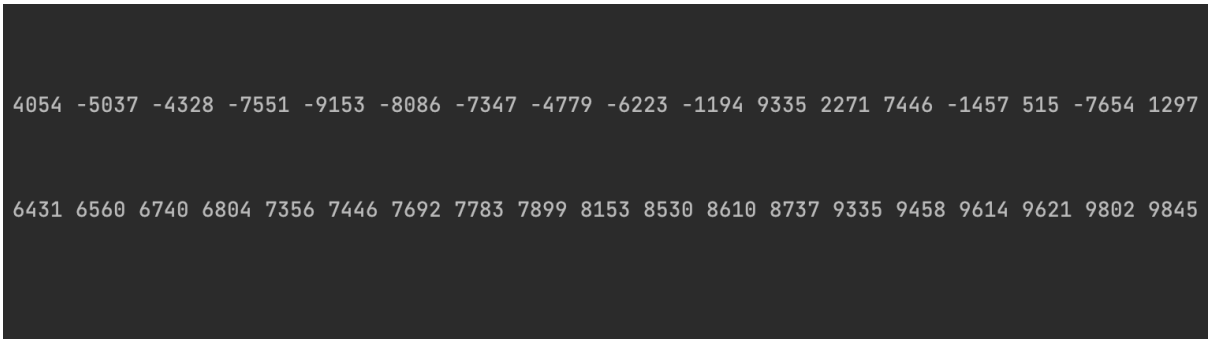
Measuring execution time, Both implementations use **System.nanoTime()** to display the results in nanoseconds, then divide the result by 1,000,000 to display milliseconds

```
// Measure the execution time of bubble sort
long startTime = System.nanoTime();
bubbleSort(array);
long endTime = System.nanoTime();

System.out.println("\nSorted array:");
printArray(array);

long duration = endTime - startTime;
long milliseconds = duration / 1_000_000; // Convert nanoseconds to milliseconds
```

*The choice of measuring execution time in both nanoseconds and milliseconds serves different purposes. Nanoseconds provide a high-resolution measurement of time, allowing us to capture precise details of the algorithm's performance. Sorting algorithms can have varying time complexities, and nanoseconds help us analyze and compare the efficiency between the two files into the finer details. On the other hand, milliseconds provide a more human-readable and intuitive representation of time. It allows us to communicate the execution time in a format that is easily understood by users. By presenting both nanoseconds and milliseconds, we cater to different perspectives and requirements.

Evidence of successful sorting for test case 3 (test3.txt):





As you can see, it gets sorted from the smallest value(negative) which is -9759 to the largest value which in this case is 9845 perfectly!

Complexity Analysis

We did 15 test cases with 10 trials per case on 3 different systems. The 3 different systems being Chris, Louis, and Micky's systems.

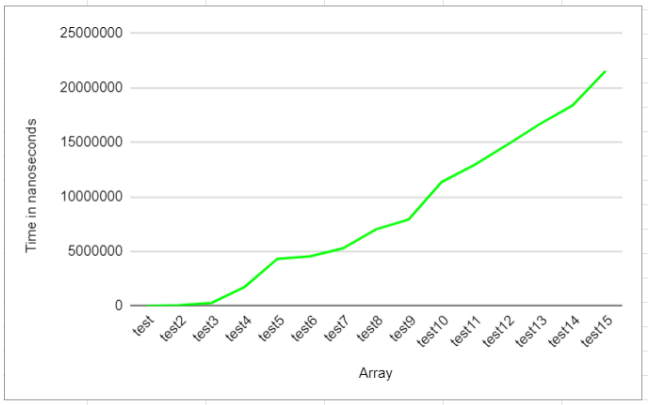
[Chris] = 2021 Macbook Pro 14 Inch M1 Chip

[Louis] = ROG Zephyrus Duo 16 GX650RM_GX650RM

[Micky] = 2021 Macbook Pro 16 Inch M1 Chip

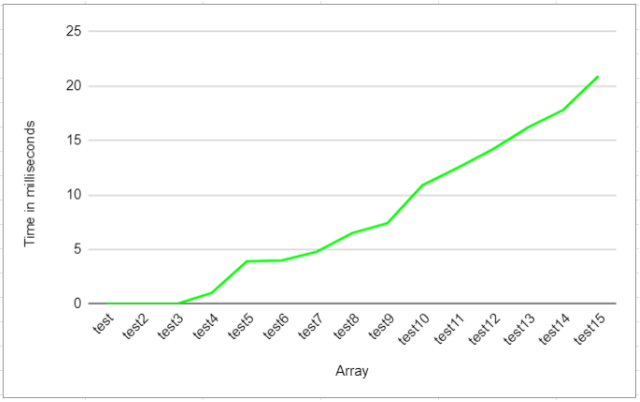
Chris's system results in an Array in nanoseconds.

Array	Time (in nanoseconds) for attempts 1 to 10										Average
test	5167	8708	5959	5791	6417	6000	6542	6250	6417	6250	6350.1
test2	72542	59542	77666	60666	87792	76167	92972	83292	69125	79000	75876.4
test3	316042	259334	284834	271833	352042	396000	312292	230083	289667	247417	295954.4
test4	1563709	1796459	1782583	1658000	1996083	1533792	1701917	1731583	1909500	1810875	1748450.1
test5	5213083	5317291	3644542	4090875	4092334	3822459	3892167	4225542	4149583	4825500	4327337.6
test6	4757458	4493541	4982208	4731708	4699708	4315458	4147334	4174042	4616958	4574875	4549329
test7	4815166	6065500	4760375	4850958	5492250	5107416	5484750	5212333	5109042	5949000	5284679
test8	7467083	6531500	7339125	6617000	8466958	5670250	7221583	6586333	6398542	7967750	7026612.4
test9	8164292	6813834	8056667	7375209	8536792	8928875	7738500	7370667	7621292	8655000	7926112.8
test10	11743208	11337833	11186666	11573958	11222750	12067250	10774459	11176125	10712875	11725792	11352091.6
test11	12151208	14502791	12873250	13088000	12042834	12655125	13776542	13003375	12524291	12582208	12919962.4
test12	14583792	15104250	13763375	14665750	14668250	14533708	15580958	14182125	14591250	15931750	14760520.8
test13	17369667	16975625	17029333	16440792	16843917	15339500	17153542	17375666	14978792	17336500	16684333.4
test14	19358708	22396875	17326875	19445958	17787167	18612125	16944417	17571042	17541167	16878125	18386245.9
test15	21256750	20771875	20344625	19958708	19993125	20510792	24631417	24964666	22412041	20452916	21529691.5



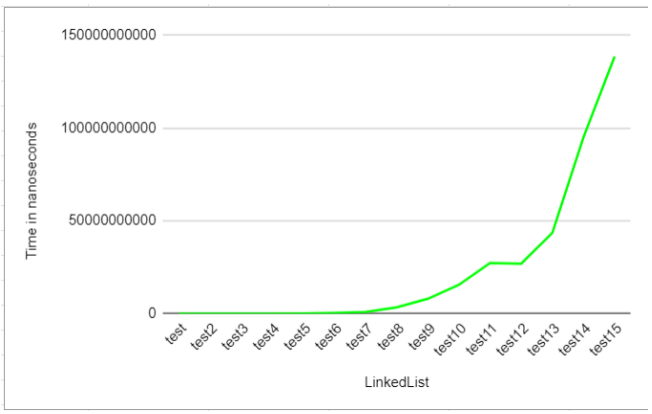
Chris's system results in an Array in milliseconds.

Array	Time (in milliseconds) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	0	0	0	0	0	0	0	0	0	0	0
test3	0	0	0	0	0	0	0	0	0	0	0
test4	1	1	1	1	1	1	1	1	1	1	1
test5	5	5	3	4	4	3	3	4	4	4	3.9
test6	4	4	4	4	4	4	4	4	4	4	4
test7	4	6	4	4	5	5	5	5	5	5	4.8
test8	7	6	7	6	8	5	7	6	6	7	6.5
test9	8	6	8	7	8	8	7	7	7	8	7.4
test10	11	11	11	11	11	12	10	11	10	11	10.9
test11	12	14	12	13	12	12	13	13	12	12	12.5
test12	14	15	13	14	14	14	15	14	14	15	14.2
test13	17	16	17	16	16	15	17	17	14	17	16.2
test14	19	22	17	19	17	18	16	17	17	16	17.8
test15	21	20	20	19	19	20	24	24	22	20	20.9



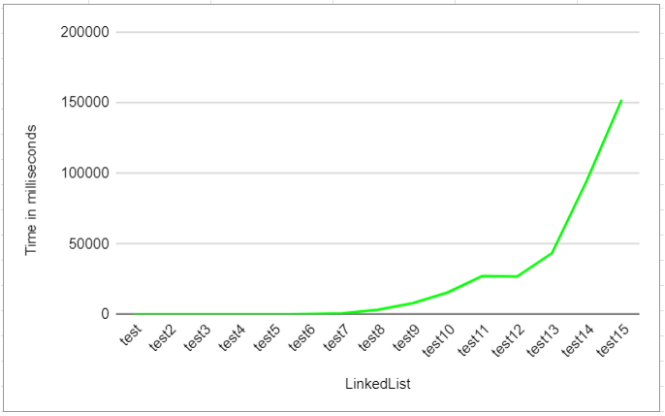
Chris's system results in LinkedList in nanoseconds.

LinkedList	Time (in nanoseconds) for attempts 1 to 10										Average
test	161959	83291	68792	89250	76041	75167	84667	98250	135333	84458	95720.8
test2	1715041	2789417	3097291	3080666	4595125	3839083	4764542	4797750	3797416	4217042	3669337.3
test3	10370958	6884583	5799167	10363041	6342000	5507875	4403625	5838916	4886666	5203500	6560033.1
test4	33650750	43315000	47544125	30689875	45564292	36529500	46695959	29323459	34424750	35937667	38367537.7
test5	141242959	134725125	137273334	140042834	143754583	132990708	133662500	131851084	136807542	134969125	136731979.4
test6	423305584	419234625	412285709	427738583	416037042	417080667	414134125	415883458	420820958	437597084	420411783.5
test7	970747209	957330584	964312125	957480209	955267166	989760583	952737458	971344792	966056041	961703958	964674012.5
test8	3402759709	3402494500	3380756875	3383072167	3398995417	3420204625	3383688625	3389742125	3414200917	3379409292	3395532425
test9	7925098500	8010314625	8119384583	7997759167	8030387500	8042512208	8059976125	8068297167	8051552458	8123782125	8042906446
test10	15654732125	15817860042	15582214750	15504099500	15601291416	15661276875	15491561209	15529297458	15484663750	15658851166	15598584829
test11	27067400666	27060846666	27001570541	27230162500	27587535917	27576897041	27451559917	27345129250	27269815208	27318099959	27290901767
test12	27593580500	27428326000	27640227542	27509594833	27531440208	27202398208	27257174167	27072664625	27130656208	27006228917	26964292292
test13	43589551417	43372167000	43649366292	43633945000	43506399459	43480865625	43433590541	43565549291	43539239083	43583534041	43535420775
test14	92445499959	94836473334	95912972750	94883529875	96975886708	95988491040	96596849214	95387235012	94340489310	92344792194	94971221940
test15	149511740583	157987034916	145792137667	157979283500	145789481024	159789482104	167858217401	149904194919	134539471977	15354065896	138450510999



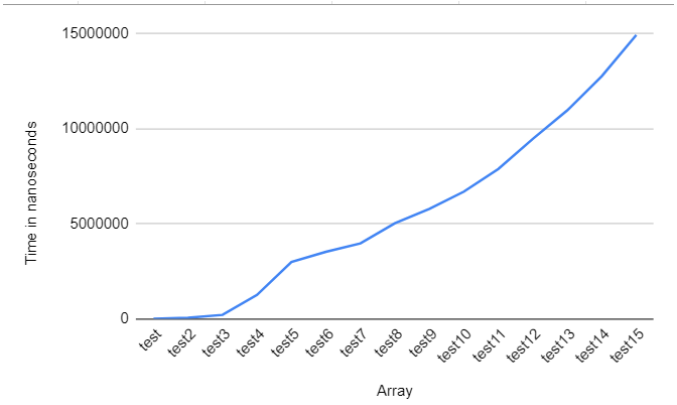
Chris's system results in LinkedList in milliseconds.

LinkedList	Time (in milliseconds) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	1	2	3	3	4	3	4	4	3	4	3.1
test3	10	6	5	10	6	6	4	5	4	5	6.1
test4	33	43	47	30	45	36	46	29	34	35	37.8
test5	141	134	137	140	143	132	133	131	136	134	136.1
test6	423	419	412	427	416	417	414	415	420	437	420
test7	970	957	964	957	955	989	952	971	966	961	964.2
test8	3402	3402	3380	3383	3398	3420	3383	3389	3414	3379	3395
test9	7925	8010	8119	7997	8030	8042	8059	8068	8051	8123	8042.4
test10	15654	15817	15582	15504	15601	15661	15491	15529	15484	15658	15598.1
test11	27067	27060	27001	27230	27587	27576	27451	27345	27269	27318	27290.4
test12	27593	27428	27640	27509	27531	27202	27257	27072	27130	27006	26964
test13	43589	43372	43649	43633	43506	43480	43433	43565	43539	43583	43534.9
test14	92445	94836	95912	94883	96975	95988	96596	95387	94340	92344	94970.6
test15	149511	157987	145792	157979	145789	159789	167858	149904	134539	153540	152268.8



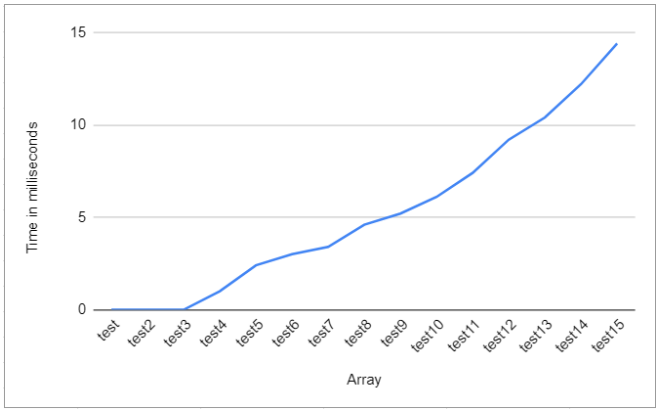
Louis's system results in an Array in nanoseconds.

Array	Time (in nanoseconds) for attempts 1 to 10										Average
test	5200	4000	4900	4800	4600	6900	4800	4200	4400	8800	5260
test2	64500	47500	57000	52600	47800	53100	80100	53000	51000	51800	55840
test3	198300	194000	207500	193600	207000	198900	200300	193000	200200	247400	204020
test4	1224400	1232900	1289700	1319100	1246000	1227699	1230199	1273200	1249801	1255301	1254830
test5	2857100	3144601	2936700	3199200	2977700	3127400	2996800	3015200	2694500	2932701	2988190.2
test6	3410599	3529201	3604899	3590399	3379401	3534300	3422200	3756600	3434100	3540701	3520240
test7	4010201	3938200	3920600	4019100	3845001	3932600	3973299	4023501	3937800	4024101	3962440.3
test8	4937900	4626901	5010001	5150200	5058600	5210999	4912601	4845899	5410099	5061399	5022459.9
test9	6131200	5992899	5398001	5220100	5975599	5819901	5692700	5633400	6037500	5795699	5769699.9
test10	6810501	6294001	7036000	6510399	6697300	6630799	6727900	6777099	6764100	6547400	6679549.9
test11	7910900	8061400	7570401	7628900	8057101	8019599	8348000	7602899	7742700	7797800	7873970
test12	9106000	9343700	9121000	9196500	9316799	9202800	10283900	10142701	9440900	9365401	9451970.1
test13	10532900	10644200	11481300	10824800	11139600	10671300	10554300	10822999	12265100	10596099	10953259.8
test14	12736099	12730700	12686200	13159600	12831800	12450300	12523701	13319700	12199599	12834300	12747199.9
test15	14613900	15042100	14756500	14311200	15104300	15154299	14579200	15826501	14943999	14762900	14909489.9



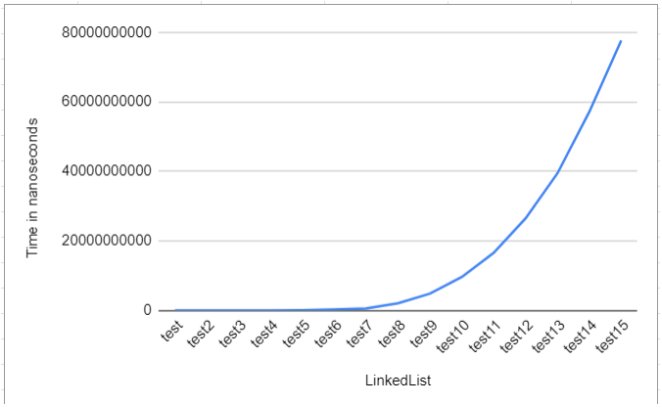
Louis's system results in an Array in milliseconds.

Array	Time (in milliseconds) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	0	0	0	0	0	0	0	0	0	0	0
test3	0	0	0	0	0	0	0	0	0	0	0
test4	1	1	1	1	1	1	1	1	1	1	1
test5	2	3	2	3	2	3	2	3	2	2	2.4
test6	3	3	3	3	3	3	3	3	3	3	3
test7	4	3	3	4	3	3	3	4	3	4	3.4
test8	4	4	5	5	5	5	4	4	5	5	4.6
test9	6	5	5	5	5	5	5	5	6	5	5.2
test10	6	6	7	6	6	6	6	6	6	6	6.1
test11	7	8	7	7	8	8	8	7	7	7	7.4
test12	9	9	9	9	9	9	10	10	9	9	9.2
test13	10	10	11	10	11	10	10	10	12	10	10.4
test14	12	12	12	13	12	12	12	13	12	12	12.2
test15	14	15	14	14	15	15	14	15	14	14	14.4



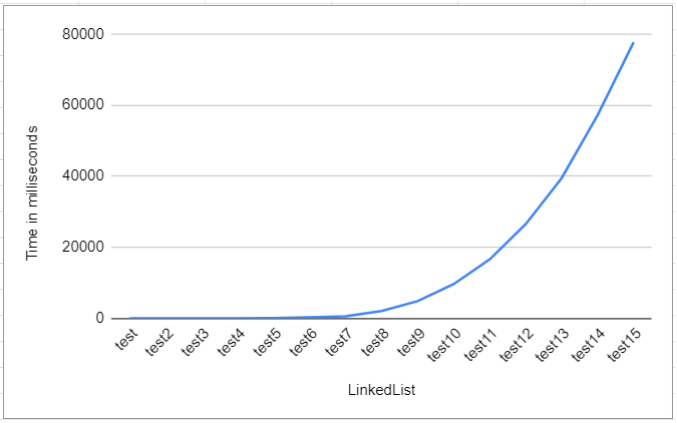
Louis's system results in LinkedList in nanoseconds.

LinkedList	Time (in nanoseconds) for attempts 1 to 10										Average
test	47700	46200	46100	69000	69500	45000	46200	45100	66900	63300	54500
test2	698700	831200	893100	662700	629000	632500	843900	915800	683400	954000	774430
test3	1887900	1980000	2133900	1941900	2105100	1932200	2390400	1919700	1902400	1983400	2017690
test4	13095400	13229600	13805800	13482800	14329700	13563400	14120600	14700600	14533400	14048000	13890930
test5	79016900	79577100	84381600	82612100	82664600	83078300	84486900	79913500	81893600	81698000	81932260
test6	249484800	250241100	253938800	252897400	255372500	253179600	255712800	257761300	256911100	256513900	254201330
test7	593183500	572658500	584634900	575333500	581834600	584254100	580856600	579282000	587626100	573455000	581311880
test8	2038695400	2017167900	2023199300	2031184000	2021257700	2039478600	2030045500	2033805500	2040827200	2036588600	2031224970
test9	4834290100	4830993700	4836007100	4807513400	4831614400	4857881000	4849986800	4840600700	4830043000	4839755100	4835868530
test10	9587377800	9655872300	9585603800	9636712600	9522850200	9602174200	9701498300	9643737200	9682573300	9540732000	9615913170
test11	16567057300	16932374800	16664709800	16290427400	16622362400	17028374100	16563113400	16248294700	16538422600	16222384600	16567752110
test12	26637922600	25982572900	26439462800	26789328400	26298462800	26174372900	27018284900	26365364100	26394371800	26693629300	26479377250
test13	39580042900	40129374600	39193264800	39412462900	39732817300	38919372700	39401329700	39277329400	39621842100	39284234800	39455207120
test14	57457133500	57191956800	56891378200	57284294100	57394102400	57294124900	56974280400	57129472900	57240217400	56980794800	57183775540
test15	78176583300	77921847200	78294104200	78937294800	78495274100	76284592300	75018249300	77680416400	77398327400	78964525600	77717121460



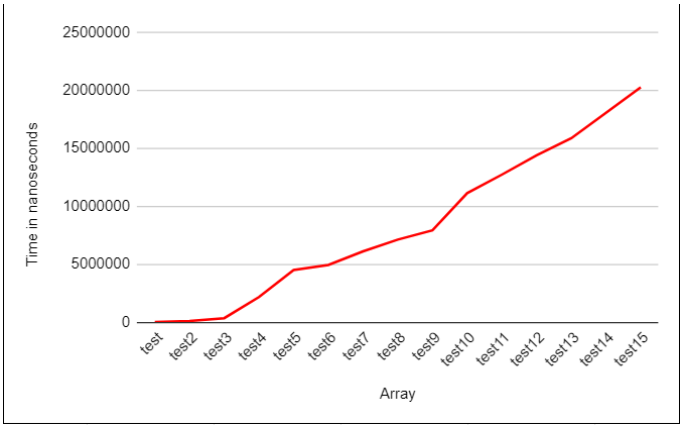
Louis's system results in LinkedList in milliseconds.

LinkedList	Time (in millisecond) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	0	0	0	0	0	0	0	0	0	0	0
test3	1	1	2	1	2	1	2	1	1	1	1.3
test4	13	13	13	13	14	13	14	14	14	14	13.5
test5	79	79	83	82	82	83	84	79	81	81	81.3
test6	249	250	253	252	255	253	255	257	256	256	253.6
test7	593	572	584	573	581	584	580	579	587	573	580.6
test8	2038	2017	2023	2031	2021	2039	2030	2033	2040	2036	2030.8
test9	4834	4830	4836	4807	4831	4857	4849	4840	4830	4839	4835.3
test10	9587	9655	9585	9636	9522	9602	9701	9643	9682	9540	9615.3
test11	16567	16932	16664	16290	16622	17028	16563	16248	16538	16222	16567.4
test12	26637	25982	26439	26789	26298	26174	27018	26365	26394	26693	26478.9
test13	39580	40129	39193	39412	39732	38919	39401	39277	39621	39284	39454.8
test14	57457	57191	56891	57284	57394	57294	56974	57129	57240	56980	57183.4
test15	78176	77921	78294	78937	78495	76284	75018	77680	77398	78964	77716.7



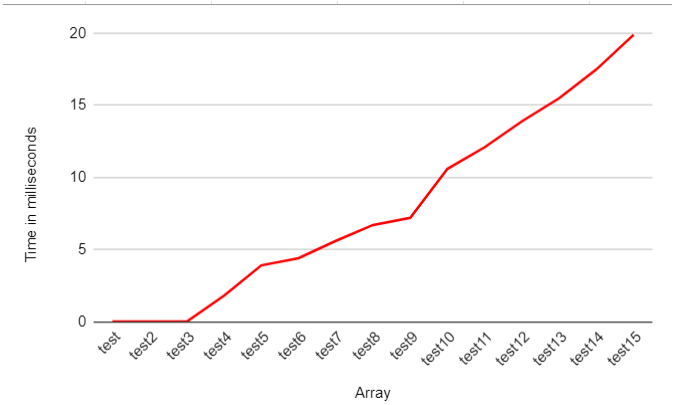
Micky's system results in an Array in nanoseconds.

Array	Time (in nanoseconds) for attempts 1 to 10										Average
test	6917	6500	6625	7333	7042	6584	10250	8917	5208	7084	7246
test2	80667	96334	82000	105416	83750	86750	90292	98666	100959	86334	91116.8
test3	373917	370417	395209	296042	278166	343792	334000	335375	318125	336750	338179.3
test4	1851625	2352750	2076208	2438834	2076875	1779750	2215209	2235750	2467792	2148750	2164354.3
test5	4352500	4717083	4915041	4766750	4421167	4199291	3703167	4639334	4789541	4453583	4495745.7
test6	4542541	5422792	5209000	4952417	4358500	5470625	4645583	4361958	5414083	4937416	4931491.5
test7	5573042	7066792	5613750	6156250	5679792	5578667	5385167	6800666	7278958	5895583	6102866.7
test8	6382959	6558709	7418292	7707167	7082375	7557959	7203834	6913375	7015041	7350167	7118987.8
test9	8534167	7889375	7837750	7762000	7715583	7675125	7520833	7617417	8747750	7895791	7919579.1
test10	11994834	11293083	11816000	11501000	11988500	8562125	11358417	9523709	11273625	12024250	11133554.3
test11	12490542	12614584	12375583	12427708	12991625	12959292	12960167	13020125	12749084	12722125	12731083.5
test12	13655875	14257208	14583958	13786084	14918500	14580333	15070292	13136167	15292250	14660000	14394066.7
test13	15551959	16687000	15093584	17060250	16022625	15203834	16000334	16661916	14709084	15801083	15879166.9
test14	17671625	17096042	17838041	17637083	18283792	18987708	17743000	19493000	17519834	18479458	18074958.3
test15	19986250	22360084	20272375	20138500	19364417	20174917	19887916	20389167	20192667	20012083	20277837.6



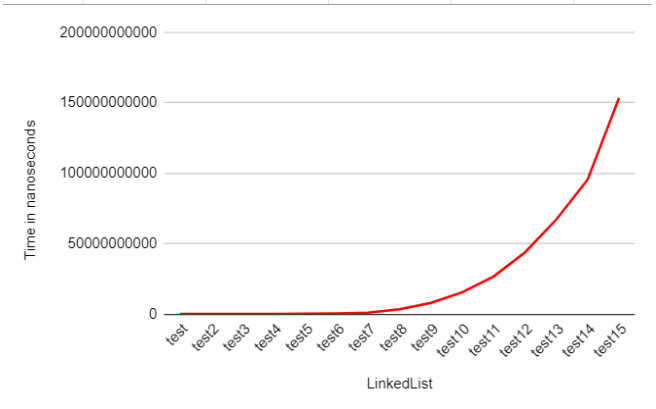
Micky’s system results in an Array in milliseconds.

Array	Time (in millisecond) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	0	0	0	0	0	0	0	0	0	0	0
test3	0	0	0	0	0	0	0	0	0	0	0
test4	1	2	2	2	2	1	2	2	2	2	1.8
test5	4	4	4	4	4	4	3	4	4	4	3.9
test6	4	5	5	4	4	5	4	4	5	4	4.4
test7	5	7	5	6	5	5	5	6	7	5	5.6
test8	6	6	7	7	7	7	7	6	7	7	6.7
test9	8	7	7	7	7	7	7	7	8	7	7.2
test10	11	11	11	11	11	8	11	9	11	12	10.6
test11	12	12	12	12	12	12	12	13	12	12	12.1
test12	13	14	14	13	14	14	15	13	15	14	13.9
test13	15	16	15	17	16	15	16	16	14	15	15.5
test14	17	17	17	17	18	18	17	19	17	18	17.5
test15	19	22	20	20	19	20	19	20	20	20	19.9



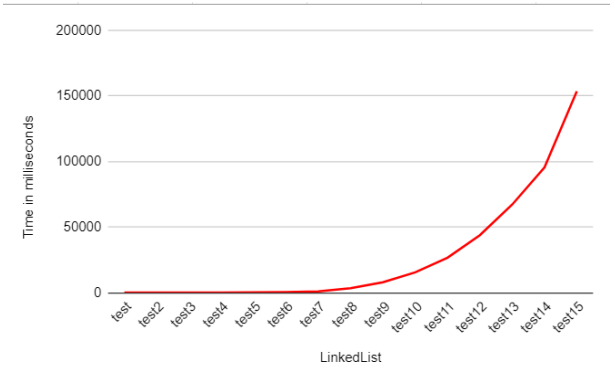
Micky’s system results in LinkedList in nanoseconds.

LinkedList	Time (in nanosecond) for attempts 1 to 10										Average
test	102166	102416	83250	110750	80792	126541	155917	108625	145166	100042	111566.5
test2	5251541	3044583	3501834	3005084	3634709	3642583	5114209	3046791	2873959	6553458	3966875.1
test3	4898209	4636917	6252417	7273750	5622041	5915166	6454291	8591084	5292333	8180083	6311629.1
test4	49495500	24884541	37844833	64818292	46006750	42294833	31480291	60478417	47771292	42303458	44737820.7
test5	145269916	137663333	134929833	145665041	141779833	149235041	147414125	141338625	145840792	162380583	145151712.2
test6	424365541	419530209	439253166	418757375	426741833	414963292	413716792	424051041	412479292	419825500	421368404.1
test7	956979458	951313334	956585458	969175000	958359542	953097166	950069541	974509167	950982334	958890167	957996116.7
test8	3327217708	3366354875	3407004042	3361084167	3331800875	3331531500	3411624208	3399248541	3376370958	3349155833	3366139271
test9	7859454958	7909108250	7948067167	7952101041	7922036667	7971656167	7886106167	7850862833	7890516250	7879602166	7906951167
test10	15363388791	15567677292	15343320833	15338793375	15391034458	15368865833	15608776667	15506881834	15371457125	15370593417	15423078963
test11	26682429292	26739158542	26711054667	26687577167	26842428391	26593492394	26948174932	26319392303	26381392031	26492817421	26639791714
test12	43418943417	43414461792	43552756209	43382301832	44928387212	43910374021	43847294175	44283917498	43649274018	43817439173	43820514935
test13	66407445250	67470231416	67192635709	66268739875	67821057333	67733750500	66855859459	67107102792	67403841943	67289371042	67155003532
test14	95782915292	95636473334	95982972750	95993529875	95422886708	95628491040	96012849214	95375235012	94729489310	95294792194	95585963473
test15	151117373000	151417034916	155592137667	156109283500	154829481024	155829482104	152948217401	151274194919	154829471977	152374065896	153632074240

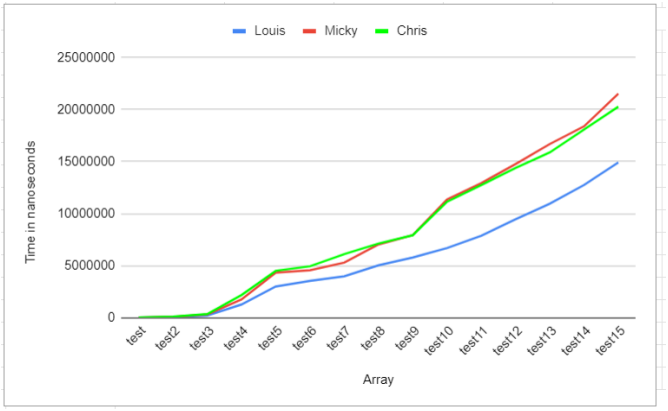


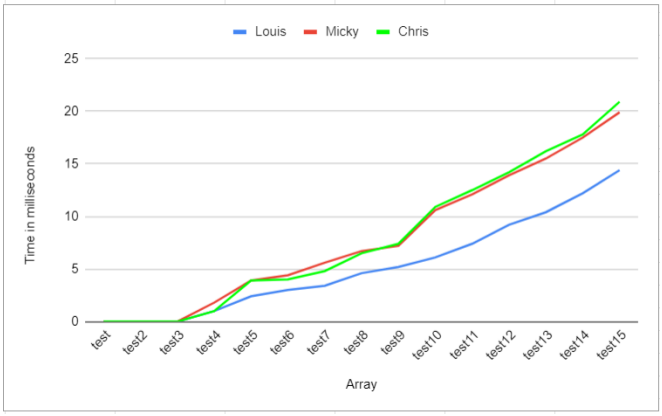
Micky’s system results in LinkedList in milliseconds.

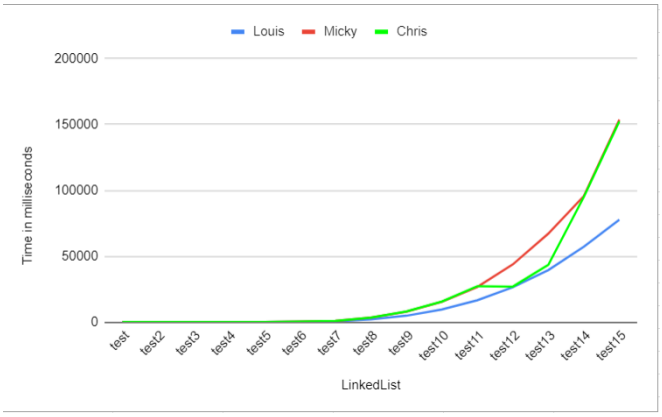
LinkedList	Time (in millisecond) for attempts 1 to 10										Average
test	0	0	0	0	0	0	0	0	0	0	0
test2	5	3	3	3	3	3	5	3	2	6	3.6
test3	4	4	6	7	5	5	6	8	5	8	5.8
test4	49	24	37	64	46	42	31	60	47	42	44.2
test5	145	137	134	145	141	149	147	141	145	162	144.6
test6	424	419	439	418	426	414	413	424	412	419	420.8
test7	956	951	956	969	958	953	950	974	950	958	957.5
test8	3327	3366	3407	3361	3331	3331	3411	3399	3376	3349	3365.8
test9	7859	7909	7948	7952	7922	7971	7886	7850	7890	7879	7906.6
test10	15363	15567	15343	15338	15391	15368	15608	15506	15371	15370	15422.5
test11	26682	26739	26711	26687	26842	26593	26948	26319	26381	26492	26639.4
test12	43418	43414	43552	43382	44928	43910	43847	44283	43649	43817	43820
test13	66407	67470	67192	66268	67821	67733	66855	67107	67403	67289	67154.5
test14	95782	95636	95982	95993	95422	95628	96012	95375	94729	95294	95585.3
test15	151117	151417	155592	156109	154829	155829	152948	151274	154829	152374	153631.8

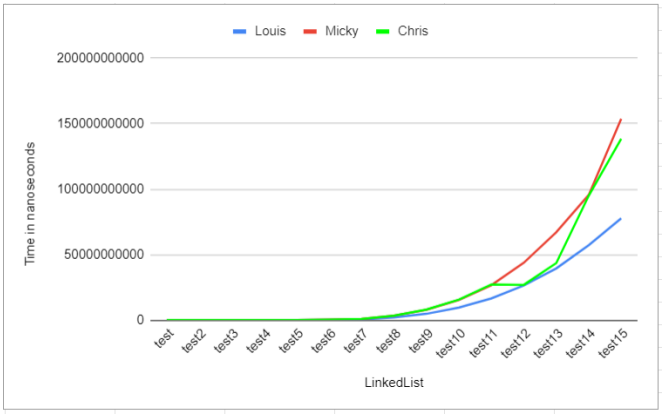


After we came up with each and every of the tables and graphs gathered from our documentation, we decided to make a graph to further show the clarity of the difference between the average we got on different systems.









We found out that the gaming laptop or in this case, the ROG Zephyrus Duo 16 performs a better job in handling our program. The very reason for the gaming laptop's better performance is the hardware specifications. Gaming laptops often come equipped with powerful processors, such as in this case the gaming laptop has a AMD Ryzen 7 6800H processor. This component is designed to handle demanding gaming applications that require high computational power and graphics processing abilities. On the other hand, macbook prioritizes portability and energy efficiency, leading to comparatively lower-end hardware specifications.

Another notable discovery we found out is that even though both the program has a time complexity of O(n^2), Array considerably runs alot faster than LinkedList especially when there is a vast amount of data. Bubble sort is generally more efficient with Arrays compared to LinkedLists due to the different nature of their data structure, like how they’re organized and accessed. Traversing a LinkedList to access elements requires following the links from one node to the next. Bubble sort needs to compare and swap adjacent elements, which can involve traversing the LinkedList multiple times. This results in a higher time complexity for bubble sort on LinkedList.

Bubble sort involves the swapping adjacent elements to sort the Arrays. With arrays, swapping elements is pretty straightforward and can be done by swapping their values in memory. This process is efficient in terms of memory access and data movement. LinkedList, however, does not store elements in contiguous memory locations. Swapping elements in a LinkedList requires adjusting the links between nodes, which involves more complex pointer manipulation. This additional overhead makes bubble sort less efficient on LinkedList compared to Arrays.

However, different data structures have their own strengths and weaknesses, and their performance can vary depending on the specific use case and the algorithm being employed. It's important to consider that the purpose of this project is not to demonstrate the superiority of one data structure over another, but rather to provide an implementation of Bubble Sort using both array and linked list to compare their performance. The goal is to showcase the differences and allow users to observe the impact of data structure choice on sorting time.

Appendix

User Manual:

- Find the Java file on GitHub: **https://github.com/mmalvino/DS_Sorting_FP**
- Download the repository as a zip file.
- Extract the zip file in your local system
- Open your IDE(intellij) and make sure you have installed JDK(program was made with version 17.0.6)
- Select the appropriate folder(src) from the extracted file and select it
- Open either BubbleSortLinkedList.java or BubbleSortArray.java and run the program
- The program provides two options for inputting the number sequence: text file(f) or manually(m)
- Default setting for text file input (f) is test.txt (first test case), you can always change this in the code
- For manual input (m), You can manually enter the number sequence by separating the numbers with commas. For example: "10, 5, -2, 8, 3".

Link to Github: **https://github.com/mmalvino/DS_Sorting_FP**

Link to the presentation file:
https://www.canva.com/design/DAFlEHLI-Jw/_D5He9JTesG_QvJayK3SXw/edit?utm_content=DAFlEHLI-Jw&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Link to our findings throughout different systems (spreadsheet containing 4 sheets, one for each member and last for combined findings):

https://docs.google.com/spreadsheets/d/1cCPV0NII8JvTuK391REeifm5A66U09PZLUsaLHObHUQ/edit?usp=sharing

Link to random number generator (RANDOM.org) which we used for our test cases:
https://www.random.org/integer-sets/