Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Group Work)

Student Information:	Surname:	Given Name:	Student ID:
1.	Djunaedy	Vania Agnes	2602158531
2.	Nasima	Azza	2602158166
3.	Huang	Jessica Angela	2602213031

Course Code : COMP6048001 **Course Name :** Data Structures and Algorithm

Class **:** L2BC **Lecturer :** Nunung Nurul Qomariyah, S.Kom., M.T.I., Ph.D.

Type of Assignment : Final Project Report

Submission Pattern

Due Date **:** 20 June 2023 **Submission Date :** 20 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 Students:



Vania Agnes Djunaedy Jessica Angela Huang Azza Nasima

Table of Contents

I. Introduction

A. Background

The Flower Management System is a user-friendly software solution designed to streamline the process of managing and tracking flower inventory. This system aims to provide an efficient and reliable means of organizing flower data, including their names, prices, and quantities, within a data structure. By offering a centralized platform for florists, garden centers, or any business dealing with flowers, this system enhances their ability to maintain accurate inventory records, make informed decisions, and optimize their operations. In this, we also added a time complexity benchmark to assess the efficiency and performance in handling various operations. By conducting time complexity analysis, we can understand how the system's execution time scales with the size of the input data.

B. Problem Description

The florists/ sellers find it hard in effectively managing and tracking flower inventory. Without a proper system, they struggle with manual processes, leading to inaccuracies, inefficiencies, overstocking or understocking of certain types of flowers, and resulting in revenue losses and missed sales opportunities. As flower businesses grow and expand, their inventory management needs become more complex. Without a scalable and adaptable system in place, businesses face difficulties in accommodating increasing product ranges, managing multiple store locations, or integrating with online platforms. To address these, there is a need for a Flower Management System that focuses on name, price, and quantity tracking, offering a user-friendly interface and robust functionality. By conducting time complexity analysis, we can understand and deter,ome how the system's execution time scales with the size of the input data and therefore get the best data structure to implement in our program.

C. Objective

Our main objective in this project is to provide floral businesses a good system to perform their needs as well as determining the most efficient data structure in our Flower Management System. By using this Flower Management System, you will be able to:

- Input New Flower: to add a new flower. It allows users to enter the necessary attributes such as the flower name, price, quantity, of a flower that is not currently present in the system.
- View Flower List: responsible for displaying a table of flowers available in the system.

- View Flower Details: responsible for displaying the details of a specific flower. It takes a flowerName parameter to identify the flower whose details should be displayed.
- Update Flower: provides the capabilities to make changes attributes such as the flower name, price, quantity, to the existing flower in the list.
- Remove Flower: remove the existing flower name with their price and quantity in the list.
- Search Flower: to find the details of the flower like name, price, quantity of the flower inputted in the list.

For this, we want to provide a user-friendly interface for managing flower information to enable easy adding, updating, searching and removing of flowers within the inventory to ensure tracking of prices and quantities of flowers. In addition, we want to provide the users with reports on flower inventory, including name, price, and quantity details to enhance operational efficiency.

II. Project Specification

- Software used:
  - IntelliJ IDEA
  - JDK (Java Development Kit)
- Library:
  - java.util.Scanner;
  - java.util.*;
  - java.util.ArrayList;
  - java.util.LinkedList
  - java.util.Queue;
  - java.util.Stack;
  - java.util.TreeMap;
  - java.io.BufferedReader;
  - java.io.FileReader;
  - java.io.IOException;
- Input:
  - displayMenu() method: to display the system menu
  - inputFlower() method: to input new flower data to the system such as flower name, price, and quantity
  - viewFlowerList() method: to view the flower list in a table
  - viewFlowerDetails() method: to view the details of a specific flower
  - updateFlowerDetails() method: to update the details of a specific flower by entering the flower name
  - removeFlower() method: to remove a flower from the list
  - searchFlowerByName() method: to search a flower by name

- Output
    - View Flower List:
        - If no data is entered, the table is still presented with the following columns: name, price, and quantity, but with no data to fill the table
        - If data is entered, the table will be displayed with the following columns: name, price, and quantity, which will be filled with the data entered.
    - View Flower Details:
        - If users enter an existing flower, the system will print out the flower's name, price, and quantity.
        - If the user does not enter an existing flower in the system, the message "Flower not found: (flower name)" will be displayed.
    - Update Flower Details:
        - If users do not enter an existing flower in the system, the message "Flower not found: (flower name)" will be displayed.

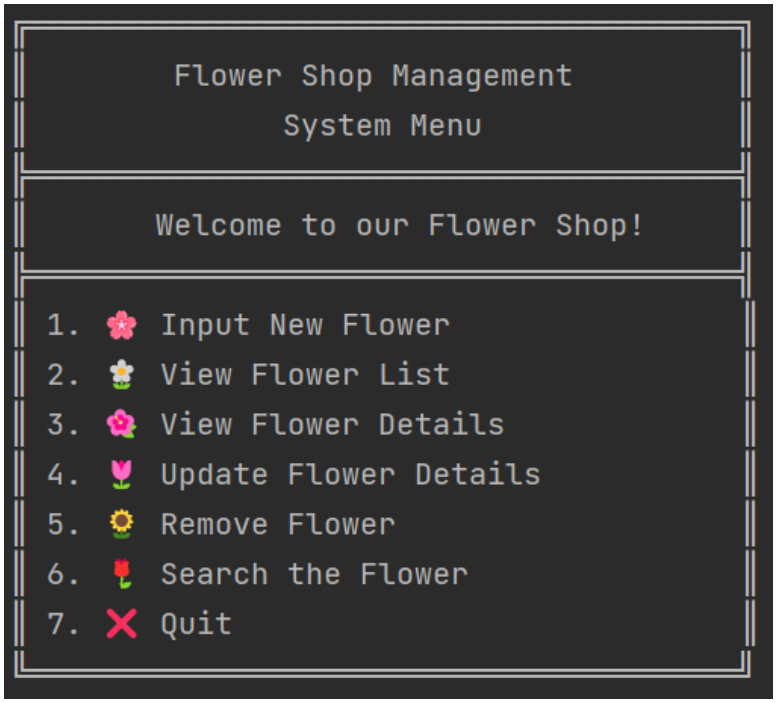
    - Remove Flower:
        - If users successfully delete a flower using the data they entered, the system will print: "Flower not found in the system."

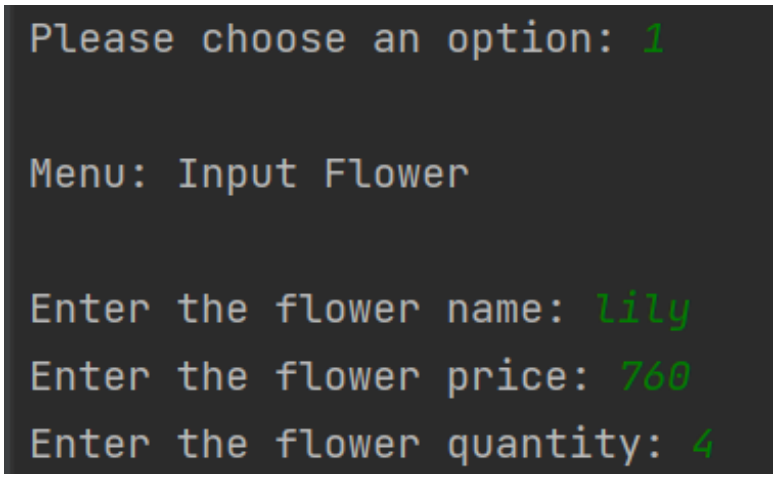
    - Search Flower:
        - If the flower is found in our system, it will print "Flower found:" and display the details for the entered flower: name, price, and quantity.
        - If the flower does not exist in our system, the message "Flower not found: (inputted flower)" will be displayed.
    - Exit:
        - When a user exits the system, the system prints "Thank you for using the Flower Shop Management System. Goodbye !"
    - Default:
        - If users enter an option that is not in the given range of 1- 7, the system will display “Invalid command ! Please try again.”

III. Proof of working system

We will show the FlowerShopArrayList implementation to demonstrate the effectiveness of our functioning program. Despite the fact that there are five different files with different data structures (FlowerShopQueue, FlowerShopTreeMap, FlowerShopStack, FlowerShopLinkedList, and FlowerShopArrayList), we have chosen to focus on FlowerShopArrayList because it generates the same output as the others. This choice assures that the provided evidence is representative of our program's full functioning.

A. FlowerShopArrayList





```
Please choose an option: 2

Menu: View Flower List

Flower Table
+------------------+------------------+------------------+
| Name             | Price            | Quantity         |
+------------------+------------------+------------------+
| lily             | 760.0            | 4                |
+------------------+------------------+------------------+
```

```
Please choose an option: 4

Menu: Update Flower

Enter the flower name: lily
Enter new price [Press ENTER to skip]:
Enter new quantity [Press ENTER to skip]: 50
```

```
Please choose an option: 2

Menu: View Flower List

Flower Table
+------------------+------------------+------------------+
| Name             | Price            | Quantity         |
+------------------+------------------+------------------+
| lily             | 760.0            | 50               |
+------------------+------------------+------------------+
```

```
Please choose an option: 6

Menu: Search Flower

Enter the flower name to search: lily
Flower found:
Name: lily
Price: $760.0
Quantity: 50
```

```
Please choose an option: 5

Menu: Remove Flower

Enter the flower name to remove: lily
```

```
Please choose an option: 2

Menu: View Flower List

Flower Table
+-----------------+-----------------+-----------------+
| Name            | Price           | Quantity        |
+-----------------+-----------------+-----------------+
```

```
Please choose an option: 7

Thank you for using the Flower Shop Management System. Goodbye!

Process finished with exit code 0
```

B. BenchmarkDS

```
╔═══════════════════════════════╗
║      Flower Shop Benchmark    ║
║           System Menu         ║
╠═══════════════════════════════╣
║    Welcome to our Flower Shop!║
╠═══════════════════════════════╣
║ 1. 🌸 Input New Flower        ║
║ 2. 🌷 Update Flower Details   ║
║ 3. 🌻 Remove Flower           ║
║ 4. 🌹 Search the Flower       ║
║ 5. ❌ Quit                    ║
╚═══════════════════════════════╝

Please choose an option: 1

The Result of the Benchmark for different data structure:
Enter the number of flowers (x) in the benchmark: 100
```

```
Array List
============
Benchmark time: 25.0 ms

Stack
============
Benchmark time: 7.0 ms

Tree Map
============
Benchmark time: 59.0 ms

Linked List
============
Benchmark time: 12.5 ms

Queue
============
Benchmark time: 4.6 ms
```

```
Please choose an option: 2

The Result of the Benchmark for different data structure:
Enter the number of flowers (x) in the benchmark: 100

Array List
============
Benchmark time: 514.1 ms

Stack
============
Benchmark time: 2365.9 ms

TreeMap
============
Benchmark time: 40.2 ms
```

```
Linked List
============
Benchmark time: 424.1 ms


Queue
============
Benchmark time: 130.8 ms
```

```
Please choose an option: 4

The Result of the Benchmark for different data structure:

Array List
============
Flower found:
Name: Rose
Price: $98.0
Quantity: 7
Benchmark time: 3153.3 ms

Tree Map
============
Flower found:
Name: Rose
Price: $98.0
Quantity: 7
Benchmark time: 240.0 ms
```

```
Stack
=============
Flower found:
Name: Rose
Price: $98.0
Quantity: 7
Benchmark time: 205.8 ms

Linked List
=============
Flower found:
Name: Rose
Price: $98.0
Quantity: 7
Benchmark time: 201.3 ms
```

```
Queue
============
Flower found:
Name: Rose
Price: $98.0
Quantity: 7
Benchmark time: 40.0 ms
```

```
Please choose an option: 3

The Result of the Benchmark for different data structure:
Enter the number of flowers (x) in the benchmark: 10

Array List
============
Benchmark time: 185.5 ms

Stack
============
Benchmark time: 2107.7 ms

TreeMap
============
Benchmark time: 19.6 ms
```

```
Linked List
============
Benchmark time: 1154.8 ms


Queue
============
Benchmark time: 1990.0 ms
```

```
Please choose an option: 5
Closing the System ...

Process finished with exit code 0
```

IV. Solution

A. Proposed alternative data structures to solve the problem

- Array List
- Stack
- Tree Map
- Linked List
- Queue

B. Theoretical analysis of how the choice of data structures can work to solve the problem.

i. Array List

For the first data structure, we picked arraylist to help solve our problem. ArrayList is a dynamic array-like data structure in Java that allows for resizable and ordered collections of elements. It provides simplicity and ease of use, as well as dynamic sizing capabilities, which are essential for managing the fluctuating inventory of flowers. The ability to efficiently add, update, and access elements based on their index, combined with the automatic resizing feature, which makes ArrayList a suitable choice for this project. Additionally, its widespread usage, community support, and familiarity makes it a reliable option for implementing an efficient flower management system.

ii. Stack

The choice of using Stack for a flower management system with only name, price, and quantity fields is not the most suitable selection. The Stack data structure follows the Last-In-First-Out (LIFO) principle, which means that the last element added to the stack is the first one to be removed. In the context of a flower management system, we need a data structure that allows efficient addition, retrieval, and modification of flower records based on specific attributes such as name, price, and quantity. A Stack does not provide direct access to elements in the middle or beginning of the stack, which makes it unsuitable for searching or updating specific flower records. Instead, data structures like ArrayList or HashMap would be more appropriate for this scenario. But we did do this to show an example of how inefficient as well as why using stack is not the most suitable for our system.

iii. Tree Map

The TreeMap is a type of map in Java that organizes key-value pairs in a specific order. It is also organized and has fast access to flower records based on their names and ensures that the data are sorted either based on their natural order or using a custom order defined by the programmer. It helps to keep the records sorted by the names of the flowers, which makes searching and retrieving information easy and fast. With TreeMap, we can associate each flower name with its price and quantity, making it easy to update and modify the details as needed. The best part is that TreeMap handles all these tasks efficiently, even when the number of flowers increases.

iv. Linked List

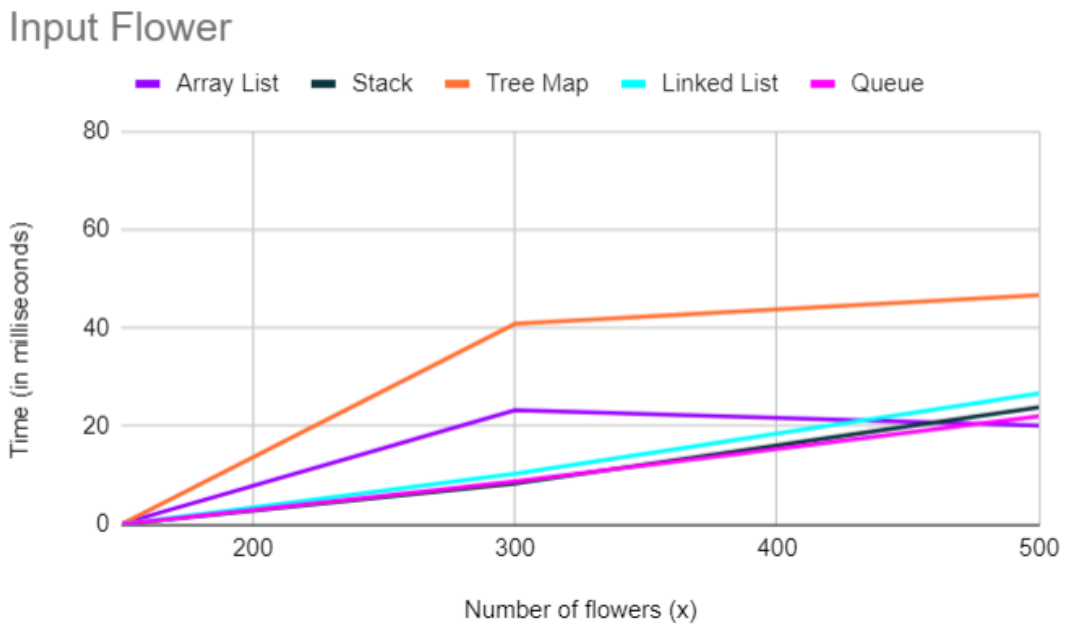
LinkedList is a useful data structure that organizes elements in a specific order. We chose LinkedList because it's great for adding and removing flower records, which is important when dealing with a constantly changing flower inventory. With LinkedList, we can easily adjust the size of the inventory by adding or removing flowers as needed. It also makes it simple to go through the flower records one by one. While it may not be as fast for directly accessing specific elements like an ArrayList, LinkedList's strength lies in its efficient insertion and deletion operations, which align with the demands of managing a dynamic inventory. Overall, LinkedList is a suitable choice for efficiently managing a flower management system with name, price, and quantity information.

v. Queue

Queue is a useful data structure that follows the "First-In-First-Out" principle, which means that the first item to enter is the first one to be processed. It's a great choice for managing the flow of flower records in the order they were added. With a Queue, we can easily handle adding and removing flower records, ensuring that the oldest entry is processed first. This is particularly helpful when dealing with scenarios like receiving new flower stocks or selling/restocking existing ones. By using a Queue, we can keep the flower records in chronological order, which allows us to effectively manage inventory updates and accurately track the quantities of flowers over time. The simplicity and efficiency of the Queue in handling the addition and removal of items make it a practical option for managing the dynamic nature of a flower management system, where new flowers are added and existing ones are sold or restocked.

C. Complexity analysis

- Input Flower



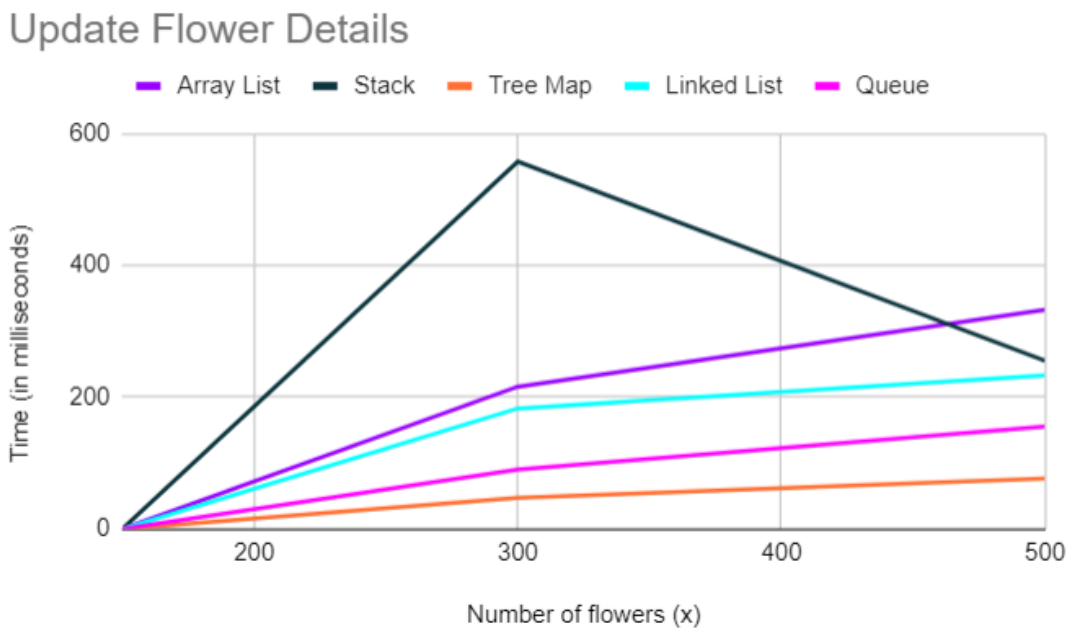
Input Flower (x = 0)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	0.2	0.2	0.2	0.2	0.4	0.24
Stack	0.1	0.1	0.1	0.1	0.1	0.1
Tree Map	0.1	0.2	0.2	0.1	0.1	0.14
Linked List	0.1	0.2	0.1	0.1	0.1	0.12
Queue	0.1	0.1	0.1	0.1	0.1	0.1

Input Flower (x = 150)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	34.8	11.3	15.2	14.3	40.9	23.3
Stack	8.2	7.7	8.3	8.7	8.9	8.36
Tree Map	46.5	33.1	33.8	44.4	46.4	40.84
Linked List	12.9	7.2	7.7	15.7	8.4	10.38
Queue	17.7	6.1	7.2	6.1	6.9	8.8

Input Flower (x = 300)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	17.6	15.1	31.7	16.5	19.9	20.16
Stack	23.7	15.6	17	33.2	29.7	23.84
Tree Map	62.5	69.3	33.1	35.5	32.8	46.64
Linked List	16.2	72.1	14.9	15.4	14.8	26.68
Queue	13.9	13.5	12.4	15.1	55.2	22.02

Input Flower (x = 500)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	30.4	26.3	36.1	23.1	38.9	30.96
Stack	19.8	18.5	23.1	17.8	30.1	21.86
Tree Map	103.8	56	71.1	49.7	42.6	64.64
Linked List	24.1	24.9	25.3	29.8	22.7	25.36
Queue	17.7	20.1	17.7	18.1	62.1	27.14

- Update Flower details



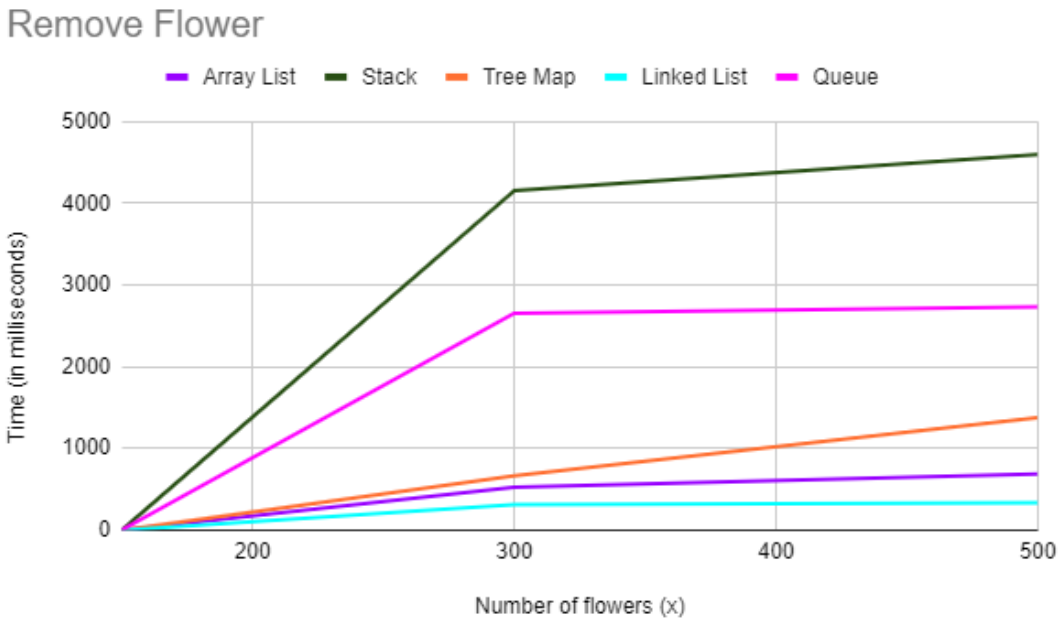
Update Flower (x = 0)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	0.3	0.3	0.1	0.4	0.2	0.26
Stack	0.2	0.1	0.2	0.2	0.2	0.18
Tree Map	0.1	0.1	0.1	0.1	0.2	0.12
Linked List	0.1	0.1	0.1	0.1	0.2	0.12
Queue	0.1	0	0.1	0.1	0.2	0.1

Update Flower (x = 150)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	615	247.5	72.8	73.3	70.8	215.88
Stack	2459.3	102.2	83.9	70.2	71.2	557.36
Tree Map	60	49.6	33.5	23.5	69.5	47.22
Linked List	581.9	79.9	153.4	46.8	52.9	182.98
Queue	112.7	82.4	114.8	68.8	71.4	90.02

Update Flower (x = 300)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	125.3	20.8	20.7	1124.5	372.4	332.74
Stack	139.7	143.5	139.3	652.7	201.4	255.32
Tree Map	33.1	20.3	16.7	244.5	68.9	76.7
Linked List	164.6	49.2	38.7	753.4	159.4	233.06
Queue	143.7	138.6	138.2	166.1	189.9	155.3

Update Flower (x = 500)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	161.7	144.1	33.1	51.2	33.1	84.64
Stack	268.1	229	230.5	254.4	231.4	242.68
Tree Map	103.9	62.1	29.2	28.7	27.1	50.2
Linked List	126.6	163	30.1	29.6	30.4	75.94
Queue	230.5	229.4	229	221.5	231.4	228.36

- Remove Flower



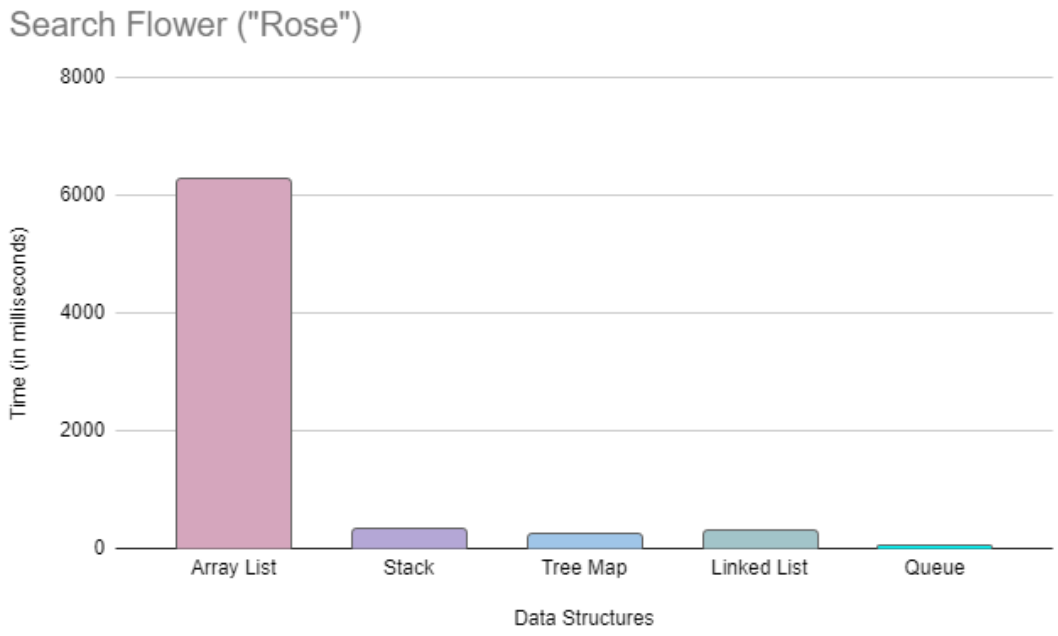
Remove Flower (x = 0)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	0.2	0.2	0.2	0.2	0.1	0.18
Stack	0.1	0.1	0.1	0.1	0.1	0.1
Tree Map	0.1	0.1	0.1	0.1	0.1	0.1
Linked List	0.1	0.1	0.1	0.1	0.1	0.1
Queue	0.1	0.1	0.2	0	0.1	0.1

Remove Flower (x = 150)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	520.6	511.4	538.5	529.5	533.9	526.78
Stack	4469.8	4648.8	4538.9	3416.7	3703.8	4155.6
Tree Map	657.1	751.9	599.2	701.8	611.7	664.34
Linked List	392.3	298.9	274.3	254.9	344.8	313.04
Queue	2700	2614.5	2566.3	2682.9	2718.1	2656.36

RemoveFlower (x = 300)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	647.9	704.2	845.8	605.5	644.8	689.64
Stack	4868.3	4455.6	5136.7	4249.2	4271.2	4596.2
Tree Map	1461.6	1521.8	1298.6	1252	1360.5	1378.9
Linked List	332.1	347.8	329.4	331.9	344.4	337.12
Queue	2639	2829.9	2698.8	2739.3	2763.3	2734.06

Remove Flower (x = 500)	Time (In milliseconds)					
Data Structure	1	2	3	4	5	Average
Array List	644.8	728.2	727.7	760.5	717.6	715.76
Stack	4271.2	4698	4805.9	4490	4354.8	4523.98
Tree Map	2420.7	2101.6	2073.1	2421	2258.9	2255.06
Linked List	399.3	434.7	471.8	450	414.2	434

- Search Flower



Search Flower ("Rose")						
Data Structure	1	2	3	4	5	Average
Array List	6567.7	6862.1	7032.9	7187	3717.2	6273.38
Stack	317.2	345.4	313.5	318.2	337.3	326.32
Tree Map	282	235.9	218.4	228	226.7	238.2
Linked LIst	305.3	300.4	291.6	306.4	304	301.54
Queue	51.8	50.8	49.3	50.6	51.1	50.72

From the data we obtained above, we can conclude that:

Method	Data Structure	0	150	300	500
Input Flower	Array List	0.24	23.3	20.16	30.96
	Stack	0.1	8.36	23.84	21.86
	Tree Map	0.14	40.84	46.64	64.64
	Linked List	0.12	10.38	26.68	25.36
	Queue	0.1	8.8	22.02	27.14
Update Flower	Array List	0.26	215.88	332.74	84.64
	Stack	0.18	557.36	255.32	242.68
	Tree Map	0.12	47.22	76.7	50.2
	Linked List	0.12	182.98	233.06	75.94
	Queue	0.1	90.02	155.3	228.36
Remove Flower	Array List	0.18	526.78	689.64	715.76
	Stack	0.1	4155.6	4596.2	4523.98
	Tree Map	0.1	664.34	1378.9	2255.06
	Linked List	0.1	313.04	337.12	434
	Queue	0.1	2656.36	2734.06	2054.68
Search Flower ("Rose")	Array List	-	-	-	6273.38
	Stack	-	-	-	326.32
	Tree Map	-	-	-	238.2
	Linked List	-	-	-	301.54
	Queue	-	-	-	50.72

Method	Fastest Data Structure	Second Fastest Data Structure	Third Fastest Data Structure	Fourth Fastest Data Structure	Slowest Data Structure	Explanation
Input Flower	Stack	Linked List	Queue	Array List	Tree Map	Stack is the fastest here, with linked list coming as second, queue as third, arraylist as fourth, and lastly tree map as fifth, being the slowest in the five data structures. Tree maps have a time complexity of O(log n) because the tree must be traversed from the root to the insertion point.
Update Flower	Tree Map	Linked List	Array List	Queue	Stack	Tree Map is the fastest here, with linked list coming as second, array list as third, queue as fourth, and lastly stack as fifth, being the slowest in the five data structures. Stack can be relatively slower because it typically requires traversing through the elements in the stack until the target element is reached.
Remove Flower	Linked List	Array List	Queue	Tree Map	Stack	Linked List is the fastest here, with array list coming as second, queue as third, Tree Map as fourth, and lastly stack as fifth, being the slowest in the five data structures.This is because stacks do not provide direct access to elements other than the topmost one.
Search Flower	Queue	Tree Map	Linked List	Stack	Array List	Queue is the fastest here, with tree map coming as second, linked list as third, stack as fourth, and lastly array list as fifth, being the slowest in the five data structures. Array List is an ordered collection that internally uses an array to store elements. It offers random access to elements based on their indices, which allows for fast search operations using index-based access. Searching for an element by index in an Array List has a time complexity of O(1), which is considered constant time.

V. Conclusion

Data Structure	Description
Array list	According to the analysis, the array list is the slowest data structure for our flower shop management system.
Stack	According to the analysis, while stack is the fastest for the input flower method, stack has the slowest time complexity for the remove flower method and update flower method, making it unsuitable for use in the flower shop management system.
Tree Map	According to the analysis, while Tree Map is the quickest in update flowers, it is also the slowest in input flowers, and for the other method, it remains stable, not the slowest nor the fastest. With that in mind, it's a good data structure but still slower than linked list and queue.
Linked List	According to the analysis, the linked list is the best performing data structure in the list, providing a rapid yet steady time complexity for all methods in the flower shop management system.
Queue	According to the analysis, the queue is the second most stable and quick data structure to be utilized in the flower shop management system. It also has the shortest time complexity in the search flower method, however it is still slower than the Linked List as a data structure.

Based on our analysis, we have evaluated the performance of five different data structures in our flower shop management system to determine each time complexity. Among them, the Array List has shown to be the slowest option, with its time complexity increasing as the number of flowers grows. This inefficiency can be attributed to its fixed size and the frequent resizing operations it requires, which significantly impact the overall time speed needed for our flower shop management system.

Next, we examined the Stack data structure and found it to be the fastest when it comes to adding new flowers. However, it falls short in the removal and update methods, where its LIFO (Last-In-First-Out) principle is not well-suited for our system. Therefore, the time complexity for these operations increases, making the Stack unsuitable for our flower shop management system.

In contrast, the Tree Map data structure performs well in the update method but falls behind in the input method. For the remaining methods, the Tree Map shows stability, neither being the slowest nor the fastest option. While it can still be considered a usable data structure for our system, there are still better options like linked list and data structure.

Additionally, the linked list appears as the top-performing data structure, offering fast and consistent time complexity across all methods in our flower shop management system. With a linked list, adding, removing, searching, and updating flowers can be efficiently accomplished.

The queue, on the other hand, stands as the second most stable data structure, demonstrating the shortest time complexity in the search flower method. However, it's important to note that the queue's performance is slightly slower compared to the linked list.

In conclusion, the linked list proves to be the recommended choice for our flower shop management system. Its efficient performance across all methods ensures that our operations can be carried out swiftly and reliably. By implementing the linked list data structure, we can significantly enhance the efficiency and effectiveness of our flower shop management system.

The performance results mentioned earlier might differ from theoretical expectations for certain data structures due to various factors, including differences in implementation approaches, the distribution of real-world data, and the impact of hardware and software environments. It's important to note that theoretical analysis often makes assumptions about ideal conditions and may not fully fulfill the specific use that arises in real-world applications.

Performance results for specific data structures may differ from theoretical predictions due to a variety of issues, including variances in implementation methodologies, the distribution of real-world data, and the impact of hardware and software environments. It should be noted that theoretical analysis frequently includes assumptions about ideal conditions and may not fully satisfy the specific use that arises in real-world applications.

VI. Appendix

A. Program Manual (How to execute)

i. Go to the GitHub repository https://github.com/VaniaAgnes/FlowerShop-Management-System.git

ii. You can clone the repository to your device by clicking on the "Code" button and copying the repository URL. Then, open a terminal or command prompt, navigate to the desired directory, and use the git clone command followed by the repository URL. (e.g: git clone https://github.com/VaniaAgnes/FlowerShop-Management-System.git) You can also download the entire repository as a zip file by pressing on the “Download ZIP” option and then extract the file in your computer.

iii. After extracting the file into your computer, you can locate the file and open the file in your IDE (Integrated Development Environment) like Intellij, Netbeans, and more.

iv. To run a java file, you will need to set up a JDK file. (You will have to download JDK if you don’t have one installed or if you don’t have the appropriate version)

v. You will need to compile it using the javac command,iff the Java file is not already compiled. For example, to compile a file named Main.java, run the following command: "javac Main.java" in your terminal. This will generate a file named Main.class if there are no compilation errors.

vi. Lastly,you can run the java program in the Main.java file. You can also type "java Main" in your terminal. The Java program should now execute, and you will see the output in the terminal or command prompt.

B. Link to the GIT website

https://github.com/VaniaAgnes/FlowerShop-Management-System.git

C. Link to the presentation file

https://www.canva.com/design/DAFloEGyFjs/7-pyG-LbH7UQS0oEWLhPyg/edit?utm_content=DAFloEGyFjs&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton