

## Higher National Certificate

### Unit 19: Data Structures and Algorithms

# ASSIGNMENT 2

Assessor Name: **PHAN MINH TAM**  
Learner's Name: DAO VINH LOC  
ID: TCS21019  
Class: GCS1006

## ASSIGNMENT 2 FRONT SHEET

<b>Qualification</b>	<b>BTEC Level 5 HND Diploma in Computing</b>		
<b>Unit number and title</b>	Unit 19: Data Structures and Algorithms		
<b>Submission date</b>	20 <sup>th</sup> Oct 2023	<b>Date Received 1st submission</b>	
<b>Re-submission Date</b>		<b>Date Received 2nd submission</b>	
<b>Student Name</b>	DAO VINH LOC	<b>Student ID</b>	TCS21019
<b>Class</b>	GCS1006	<b>Assessor name</b>	PHAN MINH TAM
<b>Student declaration</b> I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.			
		<b>Student's signature</b>	

### Grading grid

P4	P5	P6	P7	M4	M5	D3	D4

☐ Summative Feedback:

☐ Resubmission Feedback:

Grade:

Assessor Signature:

Date:

Internal Verifier's Comments:

IV Signature:

## Assignment Brief 2 (RQF)

### Higher National Certificate/Diploma in Business

<b>Student Name/ID Number:</b>	DAO VINH LOC – TCS21019
<b>Unit Number and Title:</b>	Unit 19: Data Structures and Algorithms
<b>Academic Year:</b>	2023
<b>Unit Assessor:</b>	PHAN MINH TAM
<b>Assignment Title:</b>	Implement and assess specific DSA
<b>Issue Date:</b>	
<b>Submission Date:</b>	20 <sup>th</sup> Oct 2023
<b>Internal Verifier Name:</b>	
<b>Date:</b>	

#### Submission Format:

##### *Format:*

- The submission is in the form of an individual written report and a presentation. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.

##### *Submission*

- Students are compulsory to submit the assignment in due date and in a way requested by the Tutor.
- The form of submission will be a soft copy posted on <http://cms.greenwich.edu.vn/>.
- Remember to convert the word file into **PDF file** before the submission on CMS.

##### *Note:*

- The individual Assignment *must* be your own work, and not copied by or from another student.
- If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style.
- Make sure that you understand and follow the guidelines to avoid plagiarism. Failure to comply this requirement will result in a failed assignment.

#### Unit Learning Outcomes:

**L03** Implement complex data structures and algorithms

**L04** Assess the effectiveness of data structures and algorithms

**Assignment Brief and Guidance:**

**Assignment scenario**

Continued from Assignment 1.

**Tasks**

For the middleware that is currently developing, one part of the provision interface is how message can be transferred and processed through layers. For transport, normally a buffer of queue messages is implemented and for processing, the systems requires a stack of messages.

The team now has to develop these kind of collections for the system. They should design ADT / algorithms for these 2 structures and implement a demo version with message is a string of maximum 250 characters. The demo should demonstrate some important operations of these structures. Even it's a demo, errors should be handled carefully by exceptions and some tests should be executed to prove the correctness of algorithms / operations.

The team needs to write a report of the implementation of the 2 data structures and how to measure the efficiency of related algorithms. The report should also evaluate the use of ADT in design and development, including the complexity, the trade-off and the benefits.

Learning Outcomes and Assessment Criteria (Assignment 2)		
Pass	Merit	Distinction
<b>L03</b> Implement complex data structures and algorithms		<b>D3</b> Critically evaluate the complexity of an implemented ADT/algorithm
<b>P4</b> Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem.  <b>P5</b> Implement error handling and report test results.	<b>M4</b> Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem	
<b>L04</b> Assess the effectiveness of data structures and algorithms		<b>D4</b> Evaluate three benefits of using implementation independent data structures
<b>P6</b> Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm <b>P7</b> Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.	<b>M5</b> Interpret what a trade-off is when specifying an ADT using an example to support your answer	

## Contents

1	Implement complex data structures and algorithms (LO3) .....	6
1.1	Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem (P4) .....	6
1.1.1	Description of [your application] .....	6
1.1.2	Implement .....	6
1.2	Implement error handling and report test results (P5).....	13
1.2.1	Error Handling .....	13
1.2.2	Test .....	17
2	Assess the effectiveness of data structures and algorithms (LO4) .....	19
2.1	Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm (P6).....	19
2.1.1	Definition of Asymptotic analysis of an algorithm. ....	19
2.1.2	Big O notation (O) .....	20
2.1.3	Omega notation( $\Omega$ ).....	23
2.1.4	Theta notation (V). ....	23
2.2	Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example (P7) .....	24
3	References .....	29

## 1 Implement complex data structures and algorithms (L03)

### 1.1 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem (P4)

#### 1.1.1 Description of [your application]

I am going to create an application that simulates the process of transmitting messages between two people, A and B. In the application, I will use ADT as a queue and stack to store and show messages. I chose queues because they are FIFO (first in, first out), which means that the first message sent gets eliminated from the queue. In contrast to a LIFO stack, the message that comes last is processed before leaving the stack.

#### 1.1.2 Implement

Before applying stack and queue in the project I have implemented the interface which includes themain features of Stack and Queue

```
3 public interface Stack<E> extends Iterable<E> {  
4     void push(E value);  
5  
6     E pop();  
7  
8     E top();  
9  
0     boolean isEmpty();  
1  
2     int size();  
3 }  
4
```

Figure 1: Stack interface

```
public interface Queue<E> extends Iterable<E> {  
    void enqueue(E value);  
  
    E dequeue();  
  
    E peek();  
  
    boolean isEmpty();  
  
    int size();  
}
```

Figure 2: Queue interface



Then in the project I decided to use double linked list to apply in my project, I also declared the functions corresponding to the double link list used in stack and queue.

Illustration below:

```
public class StackLinkedListImpl<E> implements Stack<E> {  
  
    private List<E> linkedList;  
  
    public StackLinkedListImpl() {  
        this.linkedList = new DoublyLinkedListImpl<>();  
    }  
  
    public StackLinkedListImpl(E first) {  
        this();  
        this.linkedList.insert(first);  
    }  
  
    @Override  
    public boolean isEmpty() {  
        return this.linkedList.isEmpty();  
    }  
  
    @Override  
    public E pop() {  
        if (isEmpty())  
            throw new NoSuchElementException(s: "Stack is empty!");  
        return this.linkedList.removeFirst();  
    }  
  
    @Override  
    public void push(E value) {  
        this.linkedList.insertFirst(value);  
    }  
}
```

Figure 3: Stack Doubly Linked List

```
public class QueueLinkedListImpl<E> implements Queue<E> {
    private List<E> linkedList;

    public QueueLinkedListImpl() {
        this.linkedList = new DoublyLinkedListImpl<>();
    }

    public QueueLinkedListImpl(E first) {
        this();
        this.linkedList.insert(first);
    }

    @Override
    public E dequeue() {
        if (isEmpty())
            throw new NoSuchElementException(s: "Queue is empty!");
        return this.linkedList.removeFirst();
    }

    @Override
    public void enqueue(E value) {
        this.linkedList.insertLast(value);
    }

    @Override
    public boolean isEmpty() {
        return this.linkedList.isEmpty();
    }

    @Override
    public E peek() {
        return this.linkedList.peekFirst();
    }

    @Override
    public int size() {
        return this.linkedList.size();
    }
}
```

**Figure 4: Queue Doubly Linked List**

Since I implement the project according to the mvc model, I have created a Message model. This is the message object model, consisting of messages that users will send back and forth and receive messages and stacks and queues. I also declare message constructors for Model Message and similar get set functions to set the value entered by the user as well as get the values sent by the user. The Model Message is implement in the illustrator below:

```
public class Message {  
    private String from;  
    private String to;  
    private String data;  
  
    public Message(String from, String to) {  
        this.from = from;  
        this.to = to;  
        this.data = null;  
    }  
  
    public Message(String from, String data, String to) {  
        this.from = from;  
        this.to = to;  
        this.data = data;  
    }  
  
    public String getData() {  
        return data;  
    }  
  
    public void setData(String data) {  
        this.data = data;  
    }  
  
    public String getFrom() {  
        return from;  
    }  
  
    public void setFrom(String from) {  
        this.from = from;  
    }  
  
    public String getTo() {  
        return to;  
    }  
}
```

Figure 5: Message Model

The logic of executing the chat app project can be understood as follows: First when the user enters, I have initialized a double link list using the stack that I mentioned above. Then, when the user has entered the message and sent it, the message will be stored in the array of the stack and stored within it. If the user still wants to send messages, the stack's double linked list will continue to add messages to the array (application of the last in first out property of the stack).

After sending is done and the user selects the option to get all the messages just sent, the mechanism works as follows. I will use the double link list of the queue that I initialized above to store all the elements of the stack array stored above through the loop. I will then dequeue it to show the messages that the user has sent. That's the method I used in this project.

```

8  import helper.impl.StackLinkedListImpl;
9  import model.Message;
10
11  public class MessageStackBasedRepository implements MessageRepository {
12      private Stack<Message> stack;
13
14      public MessageStackBasedRepository() {
15          this.stack = new StackLinkedListImpl<>();
16      }
17
18      @Override
19      public void save(Message msg) {
20          stack.push(msg);
21      }
22
23      @Override
24      public Set<Message> getAllMessages() {

```

**Figure 6: Message Repository**

When the messages are sent by the user, I have created a stack implementation that stores them in a stack array, the messages are put into it one by one and wait for it to be retrieved. Corresponding to the topic, this is the way the request wants. Messages are stored on the stack to be retrieved in turn. The messages are using push method which I have initialized on the stack side to put the message and implement it as a doubly LinkedList.

An illustration of the mechanism that I have applied in the project will be below.



```
chat-app-based-data-structure > src > service > AccountService.java > AccountService > getLatestMessage()
1 package service;
2
3 import java.util.Set;
4
5 import helper.Queue;
6 import model.Message;
7 import repository.MessageRepository;
8 import repository.impl.MessageQueueBasedRepository;
9 import repository.impl.MessageStackBasedRepository;
10
11 public class AccountService {
12     MessageRepository msgQueueRepo = new MessageQueueBasedRepository();
13     MessageRepository msgStackRepo = new MessageStackBasedRepository();
14
15     public Set<Message> getAllMessages() {
16         for(Message msg : msgStackRepo.getAllMessages()) // storage to transfer
17             msgQueueRepo.save(msg); // transfer to stack
18         return msgQueueRepo.getAllMessages();
19     }
20
21     public void save(Message msg) {
22         msgStackRepo.save(msg);
23     }
24
25     public Message getLatestMessage() {
26         Message msg = msgQueueRepo.getLatestMessage();
27         msgQueueRepo.save(msg);
28         return msgQueueRepo.getLatestMessage();
29     }
30 }
31
32 }
```

Figure 7: Code flow

As stated above, after the user sends the message and sends the information to the stack. When selecting in the menu option to get all the messages that the user has sent, the system will go to the queue to output the messages that I have implemented to push to the stack before. First, I would take out all the messages from the stack and loop through it one by one. Then I would put those messages in queue and loop through it and finally, I loop through it again and get all those messages displayed as console to the user.

Application has been implemented as below.

```
import service.AccountService;

public class App {
    static final Scanner sc = new Scanner(System.in);

    Run | Debug
    public static void main(String[] args) throws Exception {
        // Set up Repository
        AccountService accountSV = new AccountService();
        // Set up Account
        Account accOne = new Account(accountSV);
        Account accTwo = new Account(accountSV);
        Account[] accounts = new Account[] { accOne, accTwo };

        // Executed Application
        while (true) {
            Account from = accounts[0];
            Account to = accounts[1];
            fillInforAccount(from, to);
            if (!isValidAccountName(from.getName(), to.getName())) {
                System.out.println(x: "The account must have name!");
                System.out.println(x: "Please register again!");
                System.out.println(x: "Do you want to continue? (Y/N)");
                if (!sc.nextLine().equalsIgnoreCase(anotherString: "Y"))
                    break;
            } else {
                int choice = 0;
                while (true) {
                    choice = createMenu(choice);
                    if (choice == 0) {
                        System.out.println(x: "Account logged out successfully!\n");
                        return;
                    }
                    process(choice, from, to);
                    // update after sending message
                    if (choice == 1) {
                        swapAccount(accounts);
                        from = accounts[0];
                    }
                }
            }
        }
    }
}
```

**Figure 8: Main application**

Since this is a simulation, I will hard code in creating 2 accounts to simulate the app chat process. I'll have the user enter the sender's name and the recipient's name. Then I also validate to check for possible user input. Then I'll let the user choose options to perform tasks. The logic I talked about above, so I won't repeat this step. This part is probably just the menu for the user to choose and perform the desired operations.

## 1.2 Implement error handling and report test results (P5)

### 1.2.1 Error Handling

First, I handled the error in the sender and receiver input of the application. I used the conditional statement to be able to catch the error that in case the user does not enter any characters in the name input field, it will display messages and ask the user to enter the name. Details in the following illustrator:

```
// Executed Application You, 1 hour ago • app-chat _
while (true) {
    Account from = accounts[0];
    Account to = accounts[1];
    fillInforAccount(from, to);
    if (!isValidAccountName(from.getName(), to.getName())) {
        System.out.println(x: "The account must have name!");
        System.out.println(x: "Please register again!");
        System.out.println(x: "Do you want to continue? (Y/N)");
        if (!sc.nextLine().equalsIgnoreCase(anotherString: "Y"))
            break;
    } else {
        int choice = 0;
    }
}
```

**Figure 9: Handle filling name errors**

After that, I also provide conditions as well as functions to handle errors when the user enters more than 255 characters in the text editor (as the requirement has been set out before). So, I also created those functions to be able to solve the above errors. Details are shown in the illustration below:

```
switch (choice) {
    case 1:
        while (true) {
            System.out.println(String.format(format: "[%s] enter the text you want to send to [%s]", from.getName(),
                to.getName()));
            String text = sc.nextLine();
            if (text.length() > 250) {
                System.out.println(x: "*****The text must be less or equal than 250 characters!*****");
                System.out.println(x: "*****");
            } else {
                from.send(new Message(from.getName(), text, to.getName()));
            }
            System.out.println(x: "Do you want to continue to type text?(Y/N)");
            String end = sc.nextLine();
            if (!end.equalsIgnoreCase(anotherString: "Y")) {
                clearConsole();
                System.out.println(String.format(format: "Switch send message to %s account.", to.getName()));
                break;
            }
        }
}
```

**Figure 10: Handle the maximum characters errors**

In addition, the helpers that I mentioned in the source code also have error checking (specifically checking the condition for null elements) to perform further tasks. So, I will give the following examples in turn:

```
23         elements,  
24     }  
25  
26     @Override  
27     public void insertFirst(E e) {  
28         final Node<E> node = new Node<E>(e, next: null);  
29         if (head == null) {  
30             this.head = node;  
31             this.tail = node;  
32         } else {  
33             node.setNext(head);  
34             this.head = node;  
35         }  
36         ++elements;  
37     }  
38  
39     @Override
```

Figure 11: Handle null n Node

```
17     this(),  
18     this.linkedList.insert(first);  
19 }  
20  
21 @Override  
22 public boolean isEmpty() {  
23     return this.linkedList.isEmpty();  
24 }  
25
```

Figure 12: Handle Empty



## Handle Errors by using Try-Catch

```
static void fillInforAccount(Account from, Account to) {
    try{
        System.out.print(s: "Enter your name: ");
        String nameFrom = sc.nextLine();
        from.setName(nameFrom);
        System.out.print(s: "Enter your name you want to send: ");
        String nameTo = sc.nextLine();
        to.setName(nameTo);
    }
    catch(IllegalArgumentException e) {
        System.out.println(x: "value must be non-negative");
        System.out.println("Errors: "+ e);
    }
}
```

**Figure 13: Handle Wrong Input using Try Catch**

In this section I have handled the error when the user enters characters that do not match the format string that I defined earlier. So, I looked into Java Exceptions and found an `IllegalArgumentException` which is the Exception thrown when a method receives an argument that is formatted differently than the method expected. I find it extremely suitable for catching errors when user input is not intended by the application.

```
public void showAllMessages() {
    Set<Message> setMsg = accountService.getAllMessages();
    if (setMsg.isEmpty()) {
        System.out.println(x: "*****");
        System.out.println(x: "****Please send more message!****");
        System.out.println(x: "*****");
    } else {
        try{
            for (Message msg : setMsg) {
                System.out.println(String.format(format: "From: %s → To: %s", msg.getFrom(), msg.getTo()));
                System.out.println(msg.getData());
            }
        }
        catch(Exception e){
            System.out.println(x: "There are errors while trying to view the message");
            return;
        }
    }
}
```

**Figure 14: Handle errors while trying view all message**

This is the mechanism for examining messages within the message stack before they are sent to the inbox. In this case, the system tries to take all elements from the stack and put them into an array, showing them each time, and finally putting them all back into the stack and returning true as in operation succeed. If it detects a problem, it sends a message to the user, followed by the error, and returns false for the program to handle.

```

        if (!sc.nextLine().equalsIgnoreCase("Y"))
            break;
    } else {
        int choice = 0;
        while (true) {
            try{
                choice = createMenu(choice);
                if (choice == 0) {
                    System.out.println(x: "Account logged out successfully!\n");
                    return;
                }
                process(choice, from, to);
                // update after sending message
                if (choice == 1) {
                    swapAccount(accounts);
                    from = accounts[0];
                    to = accounts[1];
                }
            }
            catch(Exception e)
            {
                throw new Exception("Error: " + e.getMessage());
            }
        }
    }
}

```

**Figure 15: Handle errors while user try to choose option using try - catch**

In the Menu Section, I let the user choose the options one by one according to their wishes. And depending on each option, the system will perform certain functions. To be able to control unexpected errors, I also use try catch to limit the errors so that during the running or execution of the application, it will be easier for me to handle errors.

### 1.2.2 Test

Test Case	Input Data	ExpectedOutput	Actual Output	Result
1	Try to input first account name with nothing	Show the notice error	Show the noticeerror	Passed
2	Try to input second account name with nothing	Show the notice error	Show the noticeerror	Passed
3	Send message from first account to second account with "Hi"	Send successfully	Send successfully	Passed
4	Send message from second account to first account with "Good morning"	Send successfully	Send successfully	Passed
5	Choose option get all message	Show all message of the conversation between first and second account	Show all message of the conversation between first and second account	Passed
6	Choose get message	Show the near message of the conversation	Show the near message of the conversation	Passed
7	Try to choose get message option when two account didn't send anything	Show notice nothing in conversation	Show notice nothing in conversation	Passed

#### Test Case 1 and 2

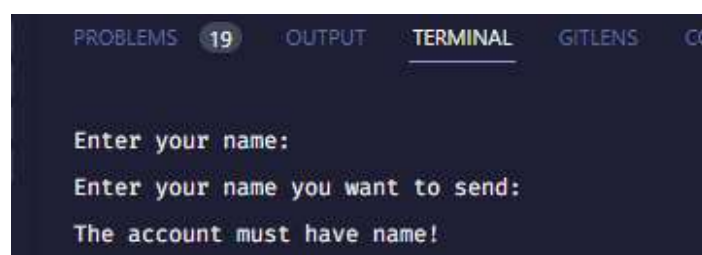
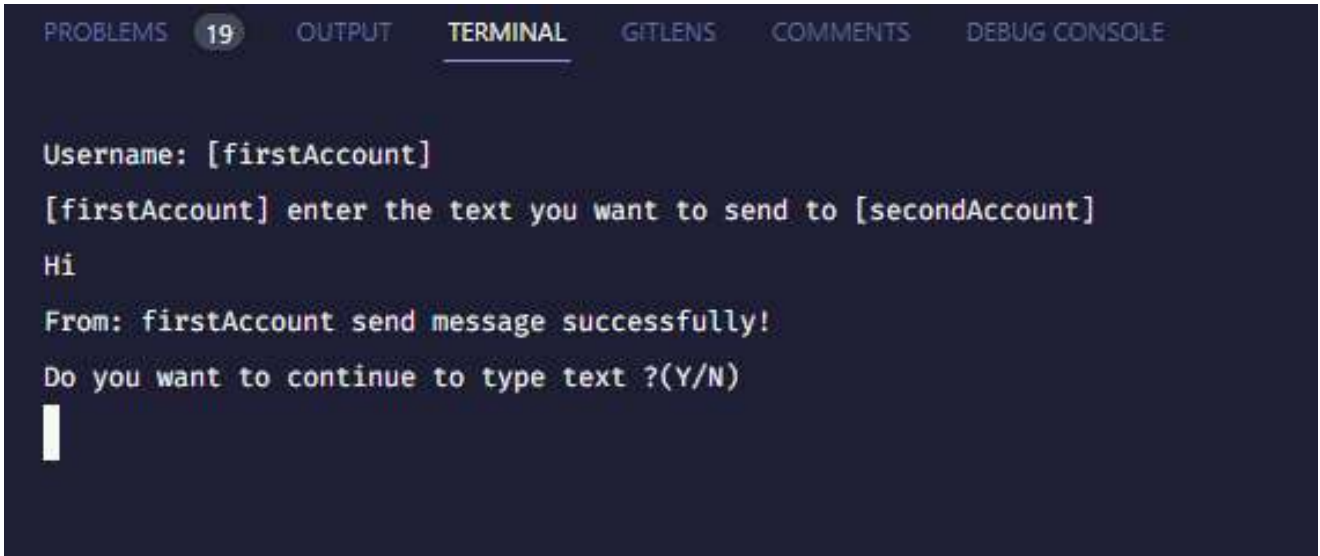


Figure 16: Test Case 1 and 2

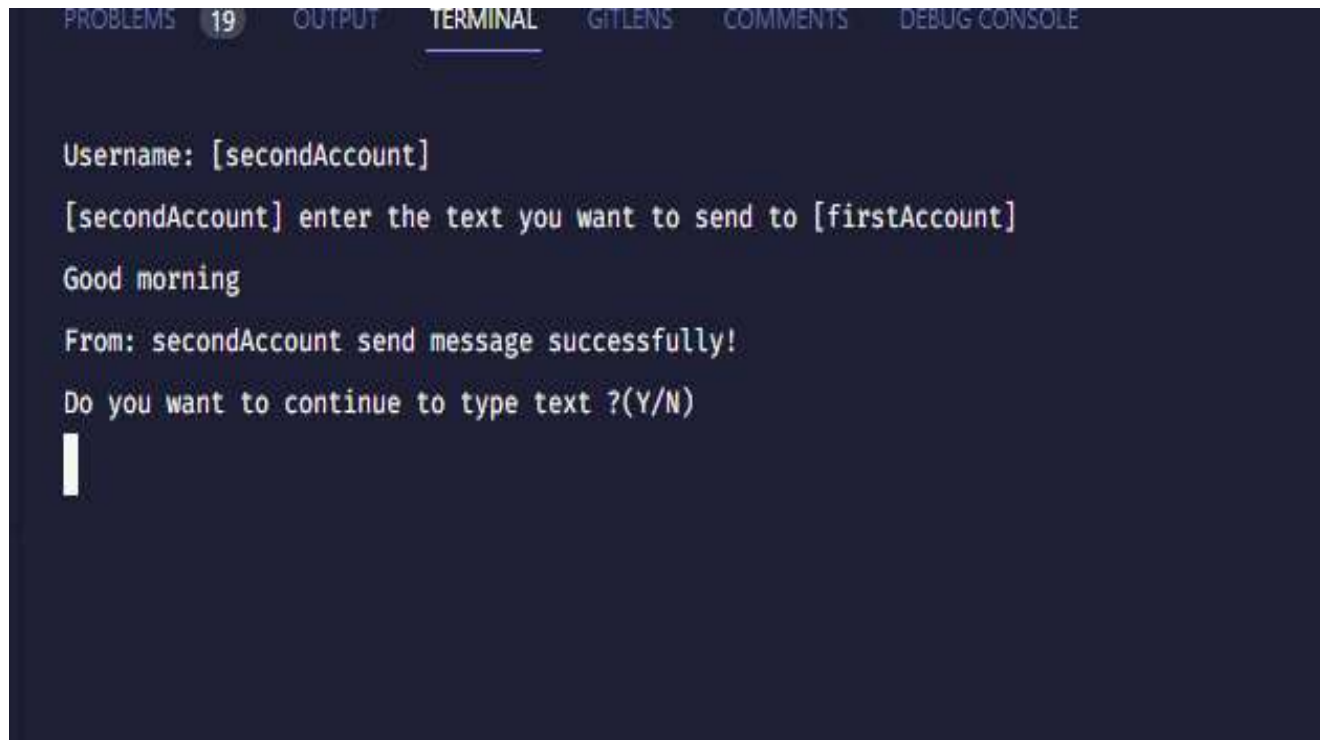
## Test Case 3 and 4



```
PROBLEMS 19 OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE

Username: [firstAccount]
[firstAccount] enter the text you want to send to [secondAccount]
Hi
From: firstAccount send message successfully!
Do you want to continue to type text?(Y/N)
█
```

Figure 17: Test Case 3 and 4

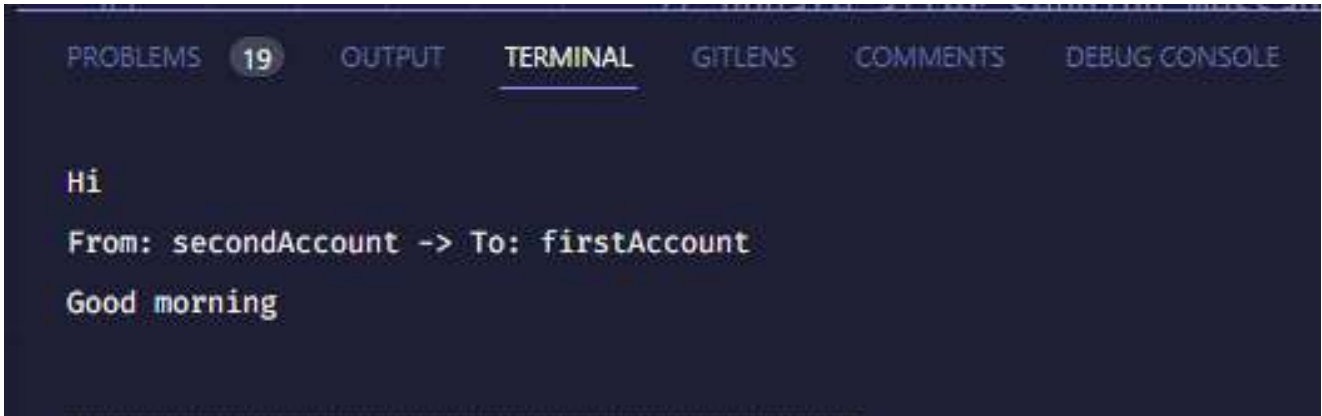


```
PROBLEMS 19 OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE

Username: [secondAccount]
[secondAccount] enter the text you want to send to [firstAccount]
Good morning
From: secondAccount send message successfully!
Do you want to continue to type text?(Y/N)
█
```

Figure 17: Test case 3 and 4 part 2

## Test Case 5 and 6



```
PROBLEMS 19 OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE

Hi
From: secondAccount -> To: firstAccount
Good morning
```

Figure 18: Test case 5 and 6

Source code is available here: <https://github.com/LokiDao/MessageQueue>

## 2 Assess the effectiveness of data structures and algorithms (L04)

### 2.1 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm (P6)

#### 2.1.1 Definition of Asymptotic analysis of an algorithm.

Asymptotes in data structures and algorithms are used to assess and estimate an algorithm's execution time. When an algorithm is developed, the asymptotic analysis stage of the algorithm allows you to analyse the method's excellent, poor, and average instances. When performing asymptotic analysis on an algorithm with no input, the running time will be a fixed time (say  $n$ ) and the running time will be a constant.

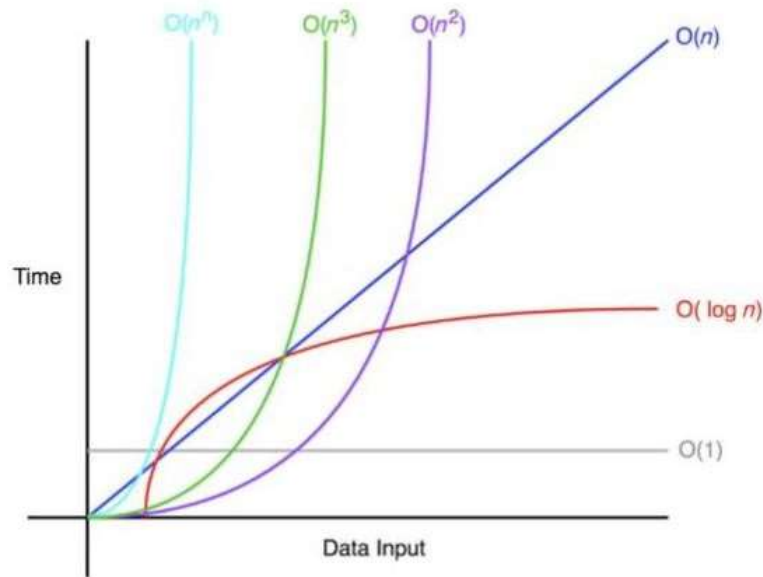
Estimating the running time of any calculation in the computational stages is sometimes referred to as asymptotic analysis. Assuming that the initial calculation's running time is expressed by the function  $f(n)$ , the other calculation's running time will be stated by the function  $f(n^2)$ .

If the value of  $n$  changes in the calculation of both operations above, the running time of the first calculation will rise linearly with the increase of  $n$ , while the running time of the second calculation would increase exponentially with the increase of  $n$ . If  $n$  is small enough, the running times of the two procedures will be almost equal.

Categorize running times according to the amount of time required into 3 main categories below:

- Best Case: Minimum time in execution
- Average Case: average time during execution
- Worst Case: Maximum time in execution

## 2.1.2 Big O notation (O)

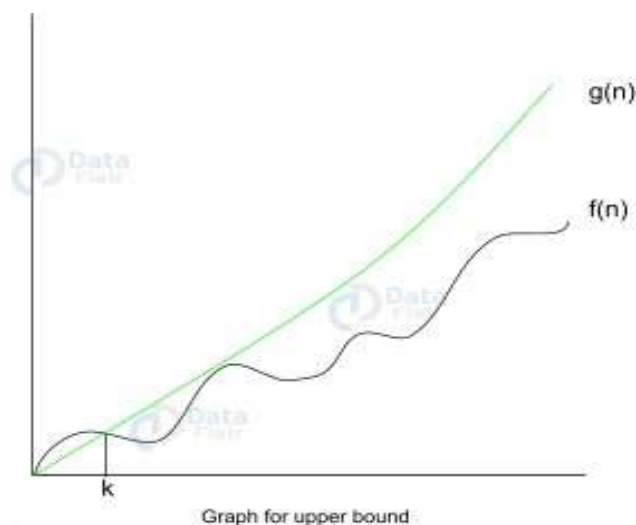


**Figure 19: Big-O notation (O)**

Big O notation is a mathematical notation that describes a function's limiting behaviour as the argument approaches a given value or infinity. Big O is a member of the Bachmann-Landau notation, often known as asymptotic notation, which was developed by Paul Bachmann, Edmund Landau, and others. Bachmann chose the letter O to symbolize Ordnung, which means "approximation order."

In computer science, Big O notation is used to describe algorithms based on how their run time or space requirements rise as the input size grows.

In this case we will analyse this Big O notation:





Let  $f(n)$ ,  $g(n)$  be two functions, where  $n \in \mathbb{Z}^+$ ,

$f(n) = O(g(n))$  [since  $f(n)$  is of the order of  $g(n)$ ].

If there exists constants 'c' and 'k' such that:  $f(n) \leq c \cdot g(n)$  for all  $n \geq k$

### $O(1) \rightarrow$ Constant Time:

```
const smallCollection = [1, 2, 3, 4];
const giganticCollection = [1, 2, 3, ..., 1000000000];

function printAllValues(n) {
  for (let i = 0; i < n.length; i++) {
    console.log(n[i]);
  }
}
```

Figure 20:  $O(1)$

In summary,  $O(1)$  indicates that it takes a consistent amount of time, such as 14 nanoseconds or threeminutes, regardless of the amount of data in the set.

### $O(n) \rightarrow$ Linear Time:

```
const smallCollection = [1, 2, 3, 4];
const giganticCollection = [1, 2, 3, ..., 1000000000];

function printAllValues(n) {
  for (let i = 0; i < n.length; i++) {
    console.log(n[i]);
  }
}
```

Figure 21:  $O(n)$

$O(n)$  indicates that it takes a time proportional to the size of the set, therefore a set twice the size will take twice as long. You probably don't want to load one of them with a million things.

### $O(n^2) \rightarrow$ Quadratic Time:

```
const smallCollection = [1, 2, 3, 4];
const giganticCollection = [1, 2, 3, ..., 1000000000];

function countOperations(n){
  let operations = 0;

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
      operations++;
    }
  }

  return operations;
}
```

Figure 22:  $O(n^2)$

$O(n^2)$  denotes that the calculation takes quadratic time, which is equal to the square of the input data size.

$O(\log n) \rightarrow$  Logarithmic Time:

```
const smallNumber = 1000;
const biggerNumber = 10000;

function countOperations(n) {
  let operations = 0;
  let i = 1;

  while (i < n) {
    i = i * 2;
    operations++;
  }

  return operations;
}
```

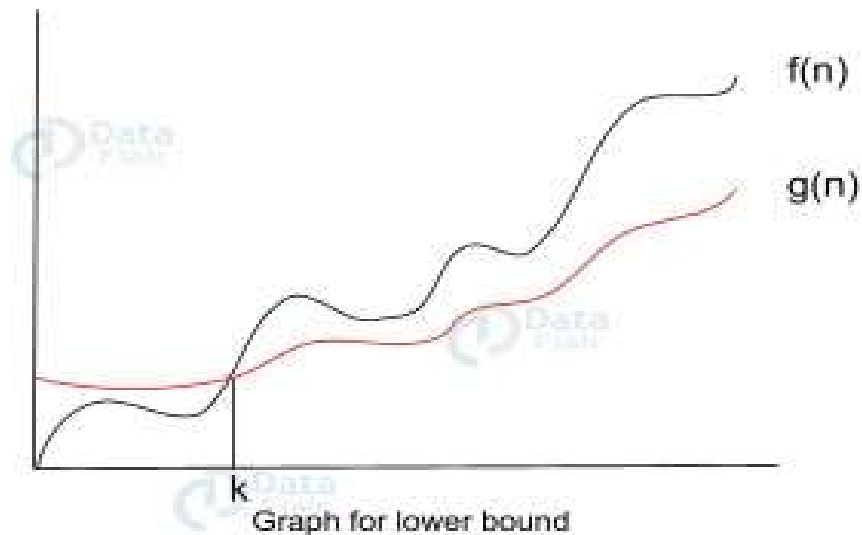
Figure 23:  $O(\log n)$

$O(\log n)$  denotes that the running time rises in proportion to the logarithm of the input size, implying that the run time increases only marginally as the input size grows exponentially.



### 2.1.3 Omega notation( $\Omega$ )

In contrast to large o notation, The formal technique to represent the bottom bound of an algorithm's execution time is with the omega notation ( $\Omega$ ). It describes the ideal situation. This signifies that this is the shortest time required to execute an algorithm. It is the quickest time an algorithm can run. It calculates the best-case time complexity, or the shortest amount of time an algorithm may take to finish.



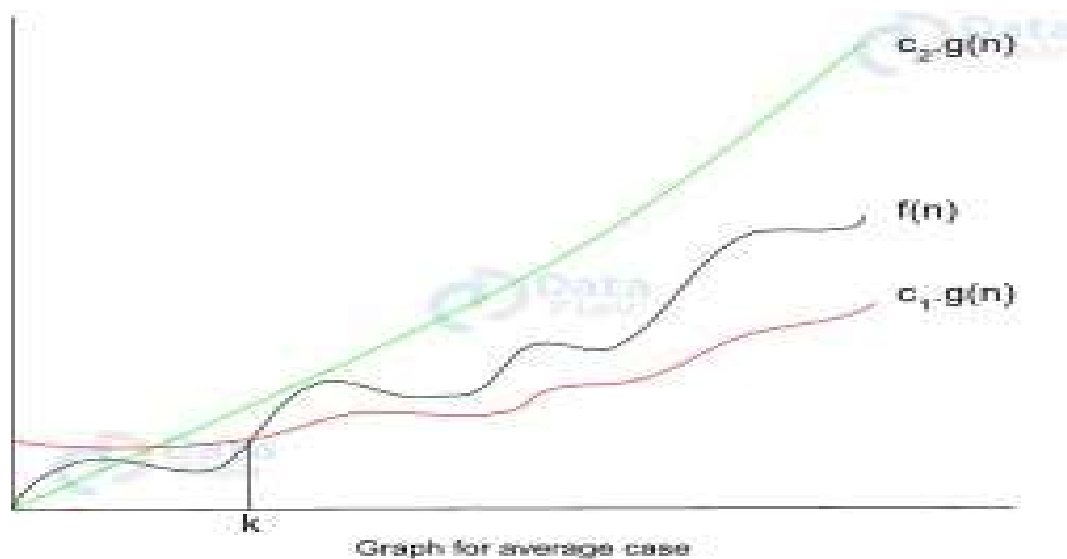
Let  $f(n)$ ,  $g(n)$  be two functions, where  $n \in \mathbb{Z}^+$ ,

$f(n) = \Omega(g(n))$  [since  $f(n)$  is of the order of  $g(n)$ ].

If there exists constants 'c' and 'k' such that:  $f(n) \geq c \cdot g(n)$  for all  $n \geq k$  and  $c > 0$

### 2.1.4 Theta notation ( $\Theta$ ).

The formal technique to represent both the lower and upper bounds of an algorithm's running time is with the theta notation  $\Theta(n)$ . The average case situation is described by this asymptotic notation. When dealing with a real-world situation, an algorithm cannot perform worst or best. The temporal complexity varies between best and worst cases, as indicated by theta notation ( $\Theta$ ), which reflects the average situation. When the worst-case and best-case values are the same, theta notation is employed. It represents both the top and lower bounds of an algorithm's execution time.



Let  $f(n)$ ,  $g(n)$  be two functions, where  $n \in \mathbb{Z}^+$ ,

$f(n) = \theta(g(n))$  [since  $f(n)$  is of the order of  $g(n)$ ].

If there exists constants 'c' and 'k' such that:  $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

## 2.2 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example (P7)

The number of resources utilized by an algorithm may be used to assess its efficiency. The speed, length, and volume of input all have an impact on an algorithm's efficiency. The key resources that an algorithm use are as follows:

**Memory complexity:** The amount of memory used by an algorithm during execution.

**Time complexity:** The amount of time the CPU takes to run the program.

To calculate the efficiency of an algorithm, the user can write a program based on the method or pseudocode, run it, and measure the time. Time constraint Time complexity is the relationship between calculation time and input quantity. This usually refers to the size of an array or object.

When people discuss algorithm efficiency, they generally refer to computational complexity, or how difficult it is to run your algorithm given the input. It is crucial to concentrate on the input component since algorithms behave differently based on the input in most cases. The best, average, and worst-case computational complexity for a

given size of input are used to illustrate this.

Benchmarking your algorithm or evaluating how long it takes to run dependent on the quantity of input, is a simple technique to evaluate its performance. As a result, you should perform rigorous studies by averaging findings from a wide sample of trials, excluding the worst and best runs, and ensuring that neither outside effects nor your own faults in selecting the benchmark may alter the run duration. Because there are overheads, this can be challenging.

### Space Complexity

Calculate space complexity algorithm.

Type	Size
bool, char, unsigned char, signed char, int8	1 byte
__int16, short, unsigned short, wchar_t, wchar_t	2 bytes
float, __int32, int, unsigned int, long, unsigned long	4 bytes
double, __int64, long double, long long	8 bytes

**Example of space complexity: Sum of all elements in an array**

```
function sum_of_numbers(arr[],N){
    sum=0
    for(i = 0 to N){
        sum=sum+arr[i]
    }
    print(sum)
}
```

**Figure 24: Space Complexity Example**

- This time, there is a method for calculating the sum of all entries in an array. We do this by supplying the array(`arr[]`) and the array size (`N`) to the generated function. So, in this case,
- The size of the array in `array(arr)` is "`N`" and each element takes "4 bytes," thus the space consumed by "`arr`" is "`N * 4 bytes`."
- The "`sum`" variable contains the sum of all items and occupies "4 bytes" of memory.
- The `I` variable is used to loop over all of the array's items, and it will also consume "4 bytes" of space.
- Now, function calls, for loop initialization, and print functions all fall under the auxiliary space, and we'll suppose they each consume "4 bytes" of space.
- As a result, total space complexity =  $(4*N + 12)$  bytes. However, because these 12 bytes are constant, we will disregard them, and after deleting all of the constants (4 from  $4*N$ ), we may conclude that the complexity of this algorithm is " $O(N)$ ."
- Where `N` depends on the size of the input array.

## Time complexity

The number of actions performed by an algorithm to achieve its goal is referred to as its temporal complexity (considering that each operation takes the same amount of time). In terms of temporal complexity, the most effective algorithm is one that completes the job with the fewest actions. The same factors that influence space difficulty also influence time complexity. To describe time constraints, utilize the mathematical function  $T(\text{num})$ , which may be measured as the number of steps if each step takes a fixed amount of time.

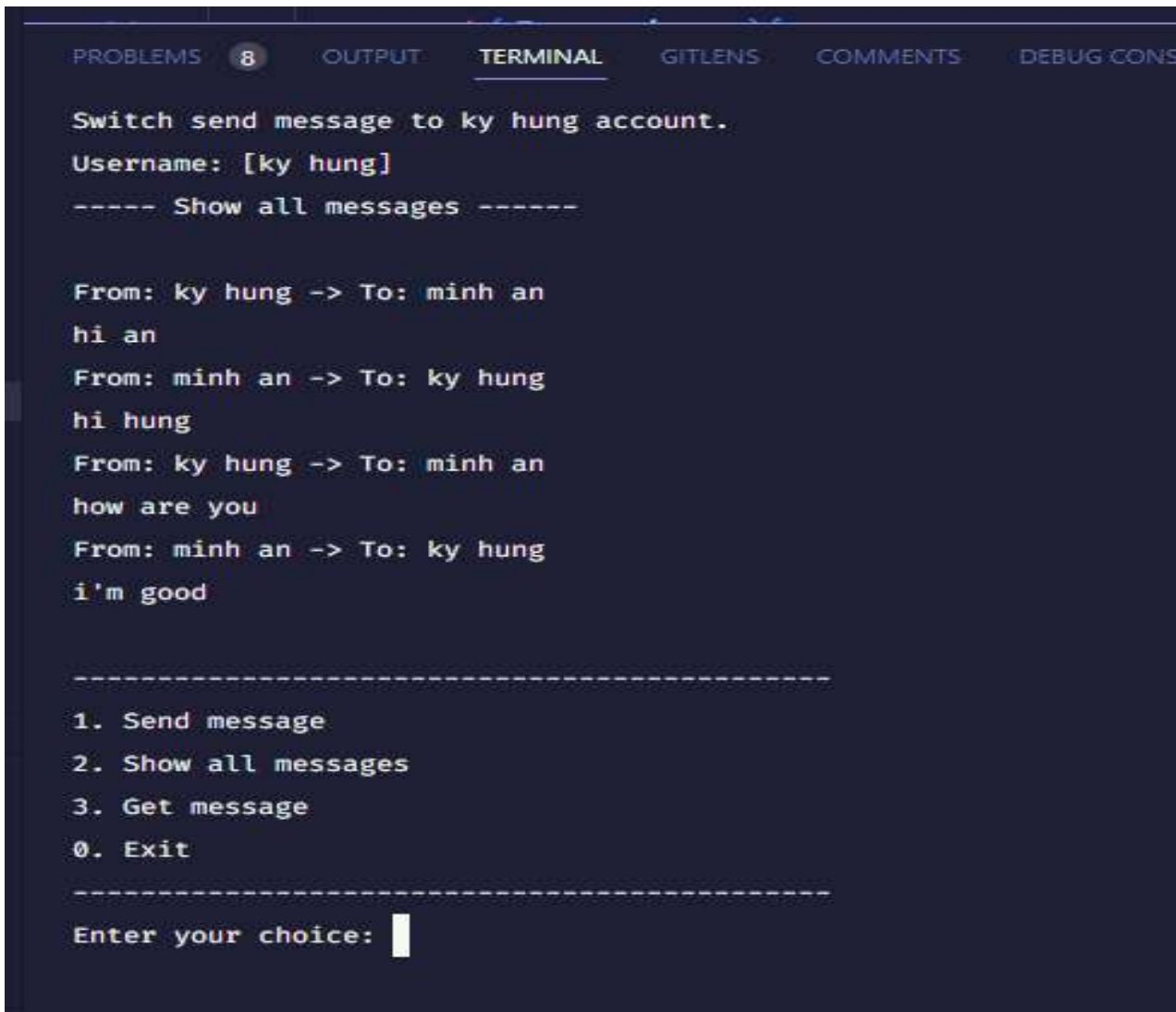
## Linear Time Algorithms – $O(n)$ :

When we state that anything develops linearly, we imply that its inputs and outputs are the same size.

## Implement code:

```
public Set<Message> getAllMessages() {
    try{
        for (Message msg : msgStackRepo.getAllMessages()) msgQueueRepo.save(msg);
        return msgQueueRepo.getAllMessages();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

## Result:



```
PROBLEMS 8 OUTPUT TERMINAL GITLENS COMMENTS DEBUG CONSOLE

Switch send message to ky hung account.
Username: [ky hung]
----- Show all messages -----

From: ky hung -> To: minh an
hi an
From: minh an -> To: ky hung
hi hung
From: ky hung -> To: minh an
how are you
From: minh an -> To: ky hung
i'm good

-----
1. Send message
2. Show all messages
3. Get message
0. Exit
-----
Enter your choice: |
```

**Figure 25: Implement and conclusion of project complexity**

## Time:

I use manual timing so I can measure the amount of time this application takes when receiving input and output levels below a fixed level of 10000000 times. We then wrote a formula that ran to nanoscale and millisecond scale to give the most accurate results about the application I designed. The illustrations will be given below:

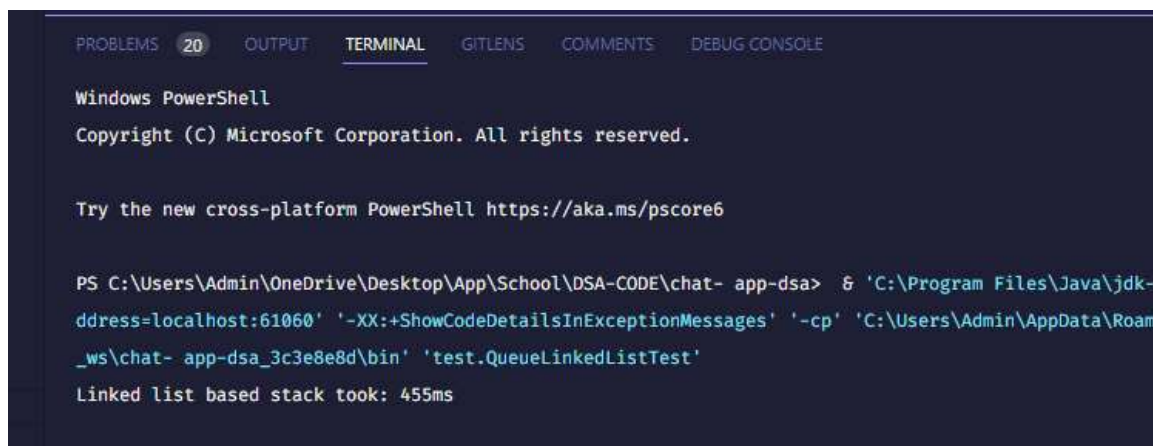
```
int numberOfOperations = 10000000;
long startTime = System.nanoTime();
// Linked list based queue
Queue<Integer> linkedListBasedQueue = new QueueLinkedListImpl<>();

startTime = System.nanoTime();
for (int i = 0; i < numberOfOperations; i++) {
    linkedListBasedQueue.enqueue(i);
}
for (int i = 0; i < numberOfOperations; i++) {
    linkedListBasedQueue.dequeue();
}

long endTime = System.nanoTime();
long linkedListTime = endTime - startTime;
System.out.println("Linked list based stack took: " + (linkedListTime / (1000*60*60)) + "ms \n");
}
```

**Figure 26: Test manual time**

This is the formula that I use to manually test the execution time of this program. With the number of operations that I declared above with the corresponding level of 10000000 times. And this is the result when I run.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\Admin\OneDrive\Desktop\App\School\DSA-CODE\chat- app-dsa> & 'C:\Program Files\Java\jdk-
ddress=localhost:61060' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roam
_ws\chat- app-dsa_3c3e8e8d\bin' 'test.QueueLinkedListTest'
Linked list based stack took: 455ms
```

**Figure 27: Time when Testing**

The result I ran this time is **455ms**. The reason for such a long time is that the operation level entered is 10000000 times. However, that is still not too good for the application, so I will try to continue to promote the application's runtime more in the future.

### **Conclusion:**

In our project, Time Complexity is calculated at  $O(n)$  (Linear Complexity) for the following reasons:

In the case of entering 5 consecutive messages, the messages will go to the stack and add those 5 messages in turn. For example, when clicking on feature 2 like show all message, it will go to the stack and take out all the elements in the stack and it will take  $n$  more times to push to Queue and it will also take  $n$  more times, for example,  $O(5N) \rightarrow O(N)$ .

### 3 References

M. Azmoodeh (1990). *Abstract Data Types and Algorithms*. doi:<https://doi.org/10.1007/978-1-349-21151-7>.

Bjørner, D. and Henson, M.C. (2007). *Logics of Specification Languages*. [online] Google Books. Springer. Available at: [https://books.google.com.vn/books?id=pGrD8NtyR\\_EC&printsec=frontcover&hl=vi#v=onepage&q&f=false](https://books.google.com.vn/books?id=pGrD8NtyR_EC&printsec=frontcover&hl=vi#v=onepage&q&f=false) [Accessed 20 Oct. 2023].

Marshall, L.S., Jones, R.B. and Bloomfield, R.E. (2008). *VDM '88. VDM - The Way Ahead: 2nd VDM-Europe Symposium, Dublin, Ireland, September 11-16, 1988. Proceedings*. 1988th edition ed. [online] Amazon. Berlin Heidelberg: Springer. Available at: <https://www.amazon.de/-/en/Lynn-S-Marshall/dp/3540502149> [Accessed 20 Oct. 2023].

computersciencewiki.org. (n.d.). *Abstract data structures - Computer Science Wiki*. [online] Available at: [https://computersciencewiki.org/index.php/Abstract\\_data\\_structures](https://computersciencewiki.org/index.php/Abstract_data_structures).

www.cs.toronto.edu. (n.d.). *Abstract Data Types*. [online] Available at: <http://www.cs.toronto.edu/~reid/tmp/adt.html>.

www.cs.yale.edu. (n.d.). *AbstractDataTypes*. [online] Available at: <https://www.cs.yale.edu/homes/aspnes/pinewiki/AbstractDataTypes.html>.

ece.uwaterloo.ca. (n.d.). *Abstract Data Types | Algorithms and Data Structures | University of Waterloo*. [online] Available at: [https://ece.uwaterloo.ca/~dwharder/aads/Abstract\\_data\\_types/](https://ece.uwaterloo.ca/~dwharder/aads/Abstract_data_types/).

pages.cs.wisc.edu. (n.d.). *Introduction: Abstract Data Types*. [online] Available at: <https://pages.cs.wisc.edu/~cs400/readings/Introduction/>.

web.cecs.pdx.edu. (n.d.). *Abstract Data Types*. [online] Available at: <https://web.cecs.pdx.edu/~sheard/course/Cs163/Doc/AbstractDataTypes.html> [Accessed 20 Oct. 2023].

www.cs.odu.edu. (n.d.). *Abstraction and Abstract Data Types*. [online] Available at: <https://www.cs.odu.edu/~zeil/cs361/latest/Public/adts/index.html> [Accessed 20 Oct. 2023].

Arya, S. (2022). *Abstract Data Type in Data Structure*. [online] Scaler Topics. Available at: <https://www.scaler.com/topics/abstract-data-type-in-data-structure/>.

web.mit.edu. (n.d.). *Reading 10: Abstract Data Types*. [online] Available at: <https://web.mit.edu/6.031/www/sp21/classes/10-abstract-data-types/> [Accessed 20 Oct. 2023].