UNIVERSITY of GREENWICH

Alliance with FPT Education

ASSIGNMENT 2

Data Structures & Algorithms

STUDENT: TRAN QUANG HUY

STUDENT ID: GCD18457

Assessor name: Ho Van Phi

ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit **19: Data Structures & Algorithms**		
Submission date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Tran Quang Huy	**Student ID**	GCD18457
Class	GCD0605	**Assessor name**	HO VAN PHI

Student declaration

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.

Student's signature	

Grading grid

P4	P5	P6	P7	M4	M5	D3	D4

❒ **Summative Feedback:** ❒ **Resubmission Feedback:**

[illegible]	[illegible]	[illegible]

Internal Verifier’s Comments:

Signature & Date:

Contents

TABLE OF TABLE

TABLE OF FIGURES

TABLE OF PICTURES

INTRODUCTION

Data Structures are the programmatic way of storing data so that data can be used efficiently. Almost every enterprise application uses various types of data structures in one or the other way. This tutorial will give you a great understanding on Data Structures needed to understand the complexity of enterprise level applications and need of algorithms, and data structures.

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

After present ADT from last report assigment 1. In this report assignment 2, I have reused ADT such as Queue and Stack to make a Send Message program with specific requirement, and analysis the perfomace the program.

1. Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem

1.1. Requirement of program

Message transfer program:

The system should be designed 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.

This programming include:

- Using ADT STACK / Queue) as a buffer during data transmission and reception
- Error handling: use the Exception catcher - the "try ... catch" block to catch and handle errors if any arise during program execution
- Test program and report test results (Test case, test log)

1.2. Implement program in C#

```
using System;

namespace SendMessage
{
    public class Queue
    {
        public int max;
        public char[] Q;

        public int f = 0;
        public int r = 0;

        public Queue(int max, char[] Q)
        {
            this.max = max;
            this.Q = Q;
        }
        public int Enum()
        {
            if ((r + 1) % max == 0) return max;
            else return ((max - f) + r) % max;
        }

        public void enQueue(char x)
        {

            Q[r] = x;
            if (Enum() != max)
            {
                r = (r + 1) % max;
            }
        }

        public char deQueue()
        {
            char dQ = Q[f];
            f = (f + 1) % max;
            return dQ;
        }
    }
```

```
public class Stack
    {
        public static char[] stackArray = new char[] { };
        public static int top = -1;
        public static void push(char x)
        {
            if (top == 19)
            {
                Console.WriteLine("Array full");
            }
            else
            {
                top++;
                stackArray[top] = x;
            }
        }

        public static char pop()
        {
            char x;
            if (top < 0)
            {
                x = -1;
            }
            else
            {
                x = stackArray[top];
                top--;
            }
            return x;
        }
    }
```

```
class Transfer
    {
        public string destination;
        string subDestination = "";
        // Setup maximum buffer
        public int maxBuffer = 50;
        // Setup limit characters of Message
        public int messageLimit = 250;
        public Transfer()
        {
        }
        public void SendMessage(string source)
        {
            Queue myQueue = new Queue();
            Stack myStack = new Stack();
            int n = 0;

            try
            {
                while (n < source.Length && n < messageLimit)
                {
                    while (myQueue.Count <= maxBuffer)
                    {
                        if (n == messageLimit)
                        {
                            break;
                        }
                        myQueue.Enqueue(source[n]);
                        n++;
                    }
                    //When buffer is full!

                    while (myQueue.Count != 0)
                    {
                        myStack.Push(myQueue.Dequeue());
                    }

                    while (myStack.Count != 0)
                    {
                        subDestination = myStack.Pop() + subDestination;
                    }
                    destination = destination + subDestination;
                    subDestination = "";
                }

            }
            catch (System.Exception)
            {
                subDestination = "";

                while (myQueue.Count != 0)
                {
                    myStack.Push(myQueue.Dequeue());
                }

                while (myStack.Count != 0)
                {
                    subDestination = myStack.Pop() + subDestination;
                }
                destination = destination + subDestination;
            }

        }

        public void ShowMessage()
        {
            Console.Write($"Your message: {destination}");
            Console.WriteLine($"\n\nNumber of character: {destination.Length}");
        }
    }
```

```
static void Main(string[] args)
            {
                Transfer myTransfer = new Transfer();
                Console.Write($"Enter your message (Max: {myTransfer.messageLimit} character): ");
                string source = Console.ReadLine();

                bool key = true;
                while (key)
                {
                    if (source.Length <= 0)
                    {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("Your input is empty, please input again to send 
message!");
                        Console.ResetColor();
                        Console.Write($"Enter your message (Max: {myTransfer.messageLimit} 
characters): ");
                        source = Console.ReadLine();
                    }
                    else
                    if (source.Length <= myTransfer.messageLimit && source.Length != 0)
                    {
                        key = false;
                        var watch = System.Diagnostics.Stopwatch.StartNew();

                        watch.Start();
                        // Send Message
                        myTransfer.SendMessage(source);
                        watch.Stop();

                        //Show Message
                        Console.WriteLine();
                        myTransfer.ShowMessage();

                        //Show time execution
                        Console.ForegroundColor = ConsoleColor.Green;
                        Console.WriteLine($"\nExecution Time (Buffer = {myTransfer.maxBuffer}): 
{watch.Elapsed}");
                        Console.ResetColor();
                    }
                    else
                    {
                        key = false;
                        //Show error
                        Console.WriteLine();
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine($"Your input too long ({source.Length}), we just send 
message with max {myTransfer.messageLimit} characters!");
                        Console.ResetColor();

                        //Send Message
                        var watch = System.Diagnostics.Stopwatch.StartNew();
                        watch.Start();
                        myTransfer.SendMessage(source);
                        watch.Stop();

                        Console.WriteLine();
                        myTransfer.ShowMessage();
                        //Show time execution
                        Console.ForegroundColor = ConsoleColor.Green;
                        Console.WriteLine($"\nExecution Time (Buffer = {myTransfer.maxBuffer}): 
{watch.Elapsed}");
                        Console.ResetColor();
                    }
                }
//Show memory
                Console.ForegroundColor = ConsoleColor.Red;
                long available = GC.GetTotalMemory(false);
                Console.WriteLine($"Memory usage (Buffer =
{myTransfer.maxBuffer}): {available} bytes");
                Console.ResetColor();


                Console.WriteLine();
                Console.WriteLine("----< Thank you to using GreMes >----");
                Console.WriteLine("----<   Press any key to exit   >----");


                Console.ReadKey();
            }
```

1.3. Error handling and report test results

1.3.1. Error handling

An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running.
Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally, and throw:

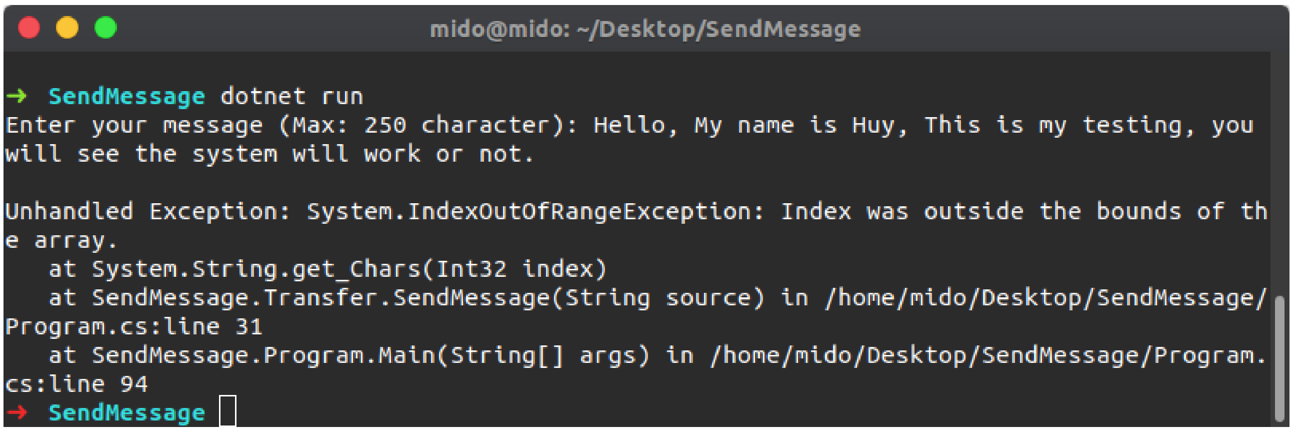
- **Try:** A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

- **Catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

- **Finally:** The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.

- **Throw:** A program throws an exception when a problem shows up. This is done using a throw keyword.

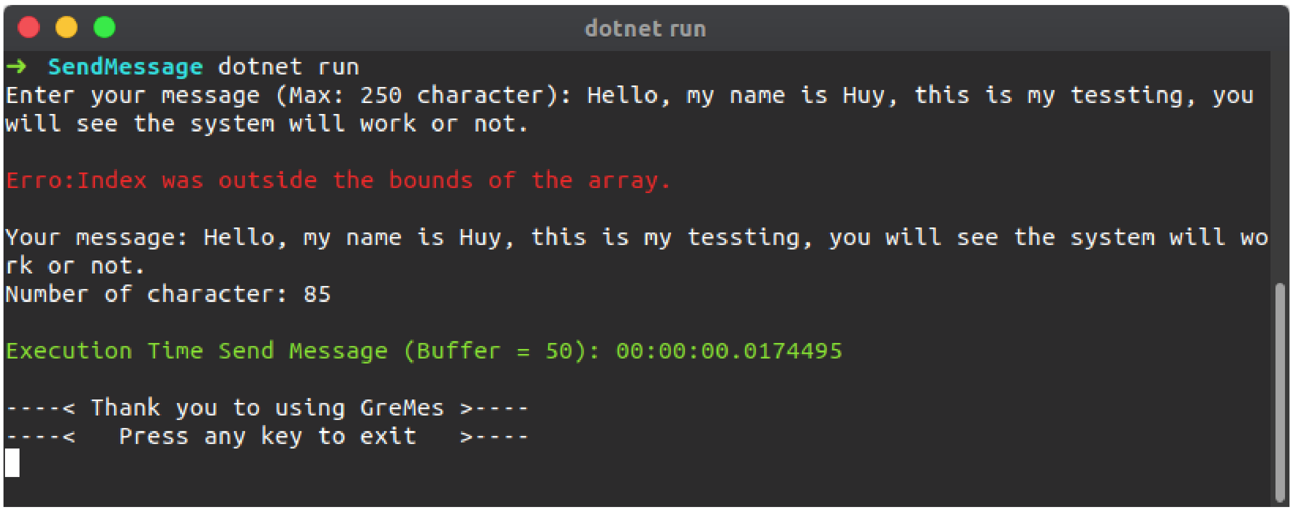
In this program we use "**Try – Catch**" to handle the error and this "**Try – Catch**" exception handling has been used to the **Transfer Function** to handle the user inputted string, avoiding the inputted string message consists of an amount of values such as this error: "Index was outside the bounds of the array".



```
mido@mido: ~/Desktop/SendMessage

→ SendMessage dotnet run
Enter your message (Max: 250 character): Hello, My name is Huy, This is my testing, you
will see the system will work or not.

Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of th
e array.
   at System.String.get_Chars(Int32 index)
   at SendMessage.Transfer.SendMessage(String source) in /home/mido/Desktop/SendMessage/
Program.cs:line 31
   at SendMessage.Program.Main(String[] args) in /home/mido/Desktop/SendMessage/Program.
cs:line 94
→ SendMessage
```

Picture 1. Result when doesn't use Try-Catch

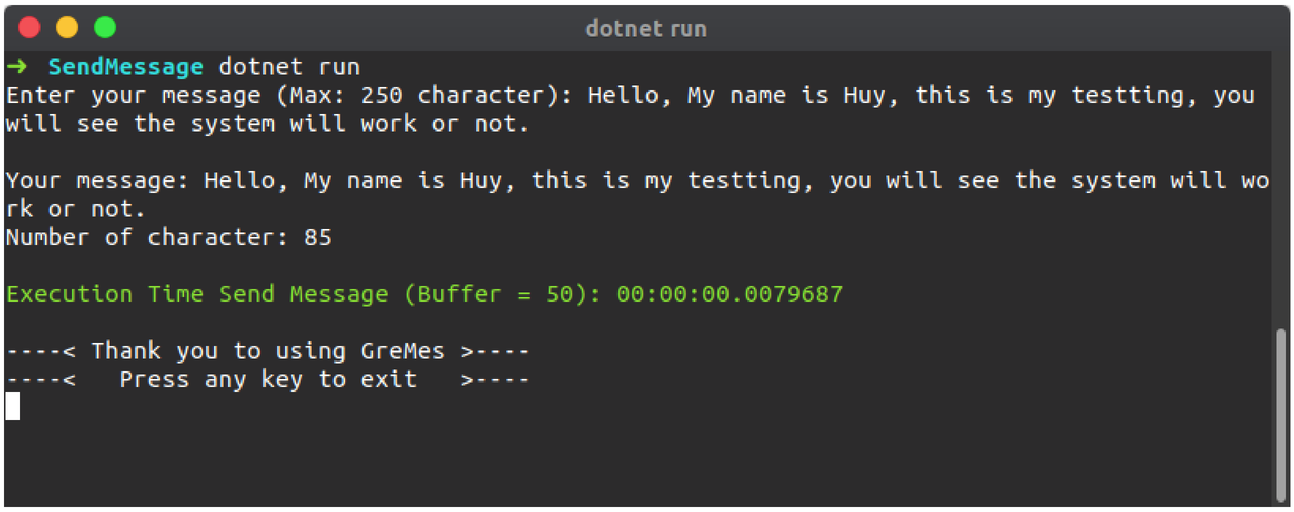
After using “Try – Catch”:



```
dotnet run
➜ SendMessage dotnet run
Enter your message (Max: 250 character): Hello, my name is Huy, this is my tessting, you
will see the system will work or not.

Erro:Index was outside the bounds of the array.

Your message: Hello, my name is Huy, this is my tessting, you will see the system will wo
rk or not.
Number of character: 85

Execution Time Send Message (Buffer = 50): 00:00:00.0174495

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 2. Result ater using "Try - Catch"

After using “Try – Catch” we could know where is bug in our program then fixed it to make the system work well.



```
dotnet run
➜ SendMessage dotnet run
Enter your message (Max: 250 character): Hello, My name is Huy, this is my testting, you
will see the system will work or not.

Your message: Hello, My name is Huy, this is my testting, you will see the system will wo
rk or not.
Number of character: 85

Execution Time Send Message (Buffer = 50): 00:00:00.0079687

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 3. Using "Try-Catch" to complete program.

1.3.2. Test results

In this case we use Test Case to defined as a set of actions executed to verify a particular feature or functionality of the software application. A test case is an indispensable component of the Software Testing Life Cycle that helps validate the Application Under Test.

A Test Case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

S.No	Action	Inputs	Expected Output	Actual Output	Test Result
1	Test the system if input a NULL string message	NULL	- Console display: "Your input is empty, please input again to send message! " - Ask user input again	Console display: "Your input is empty, please input again to send message! " - Ask user input again	Pass
2	Test the system if input a string over 250 characters. (298 characters).	"The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one. Because they've grown so quickly, they're scrambling to update their ordering system to match their beverage offerings."	- Console display "Your input too long, we just send message with max 250 characters!" - Console display: Send the messages with 250 characters. "The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one. Because they've	- Console display "Your input too long, we just send message with max 250 characters!" - Console display: Send the messages with 250 characters. "The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one. Because they've	Pass

			grown so quickly, they're scrambling to update their or"	grown so quickly, they're scrambling to update their or"	
3	Test the system if input a string in range 0 – 250 characters. (178 characters)	"The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one."	- Console display message: "The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one."	- Console display message: "The Highlands coffee has made for itself as the fastest growing coffee shop around. If you've seen once on your local corner, look across the street and you will see another one."	Pass
4	After input NULL string, press key: "Esc" to exit the program	- Input: NULL - Press "Esc"	- Console display: "Your input is empty, please input again to send message! " - Ask user input again - Exit the program	- Console display: "Your input is empty, please input again to send message! " - Ask user input again - Loop ask user input again	Fail
5	Test the system input the special character	"!@#$%^&*()_+:"?><"	Console display: "!@#$%^&*()_+:"?><"	Console display: "!@#$%^&*()_+:"?><"	Pass
6	Test the system input a string has over character of Buffer(50)	"The Highlands coffee has made for itself"	Console display: "The Highlands coffee has made for itself"	Console display: "The Highlands coffee has made for itself"	Pass

Table 1. Test case of Send Message program

1.4. How the implementation of an ADT/algorithm solves a well-defined problem

The system is using Queue as a buffer and Stack to get data to Destination Message. There are a Transfer Function will do that. To visualize the algorithm I will show you how system work with Flow Chart below:

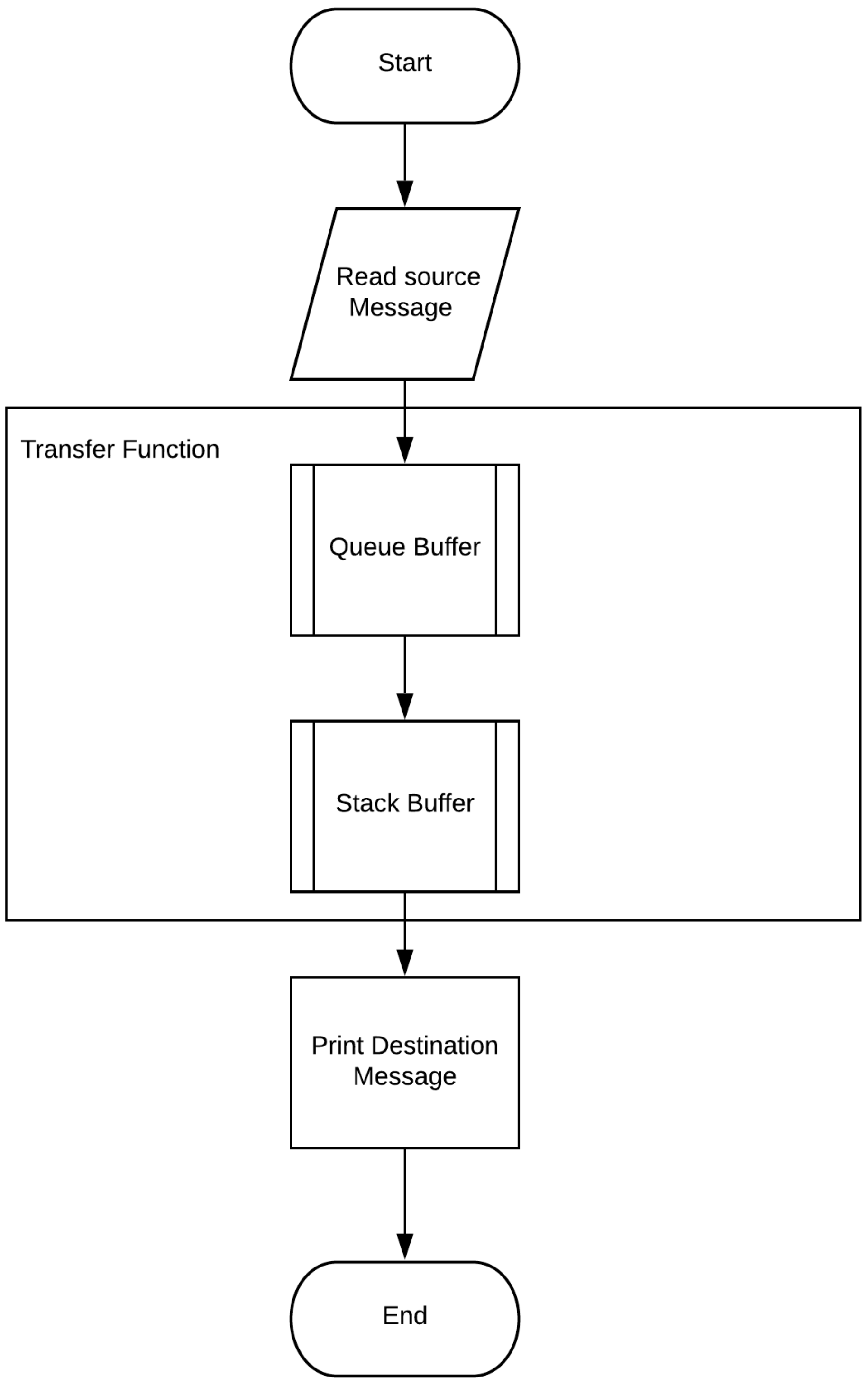


Figure 1. Flow Chart Send Message program

1.4.1. Using Queue ADT

In this system, I have used 3 function in Queue to make a Buffer such as: Enum(), Enqueue() and Dequeue():

- **Enum() function:** Check how many element in Queue (Buffer). So we can set a maximum character in Buffer.

- **Enqueue() function:** This function allow to insert the character into Queue (Buffer) and this function is only execute when the value from Enum() is bigger than 0. It means Enqueue() just work when the Queue (Buffer) is not Full.

- **Dequeue() function:** This function return the first character of the Queue (Buffer). This function is only execute when the value from Enum() is different with 0 and after the Queue (Buffer) is Full. Dequeue() will stop when the Queue (Buffer) is empty.

1.4.2. Using Stack ADT

In this system, I have used 2 function in Stack to make a Destination Message such as: Push() and Pop():

- **Push() function:** Insert the character into the top position of stack. In this program, I'm using Push() with value: Dequeue() in Queue (Buffer). So the Stack can get character from Queue (Buffer). Push() function will stop when the Queue (Buffer) is empty.

- **Pop() function:** This function will return the top character in Stack and stop when the Stack is empty. After that I connect each character into string (Destination Message) with this equation: **Destination = Pop() + Destination**

1.4.3. Transfer Function

By using Qeueue and Stack to make the Transfer function with code C# below:

Explanation:

```
while (n < source.Length && n < messageLimit)
```

The function will work until send all character of Source Message to Destination Message and just send 250 characters.

```
        public string destination;
         string subDestination = "";
         // Setup maximum buffer
         public int maxBuffer = 50;
         // Setup limit characters of Message
         public int messageLimit = 250;
        public Transfer()
         {

         }
```

We create a string destination to store the destination message, and string subDestination to store the destination message in (Try – Catch). The int maxBuffer is allow us to set the maximum character that Buffer will be stored.
And the constructor of function to call in Main program.

```
  while (myQueue.Count <= maxBuffer)
      {
        if (n == messageLimit)
      {
         break;
      }
      myQueue.Enqueue(source[n]);
      n++;
      }
     //When buffer is full!
```

When the **Queue** is not full (<= 50 character) then Insert the first character of source messages into **Queue (Buffer).**

If the **Queue(Buffer)** is not full but the message reach the maximum of program (250 characters) then stop Enqueue

```
while (myQueue.Count != 0)
{
myStack.Push(myQueue.Dequeue());
}
```

Moving element from Queue to Stack with **Push()** and **Dequeue()**. After **Dequeue()** the program will **Push()** the return value from Dequeue to Stack. When the **Queue (Buffer)** is empty then stop the **Push()**.

```
        while (myStack.Count != 0)
                     {
                         subDestination = myStack.Pop() + subDestination;
                     }
                     destination = destination + subDestination;
                     subDestination = "";
```

Connect to get Destination Message with this equation: **Destination = Pop() + Destination**

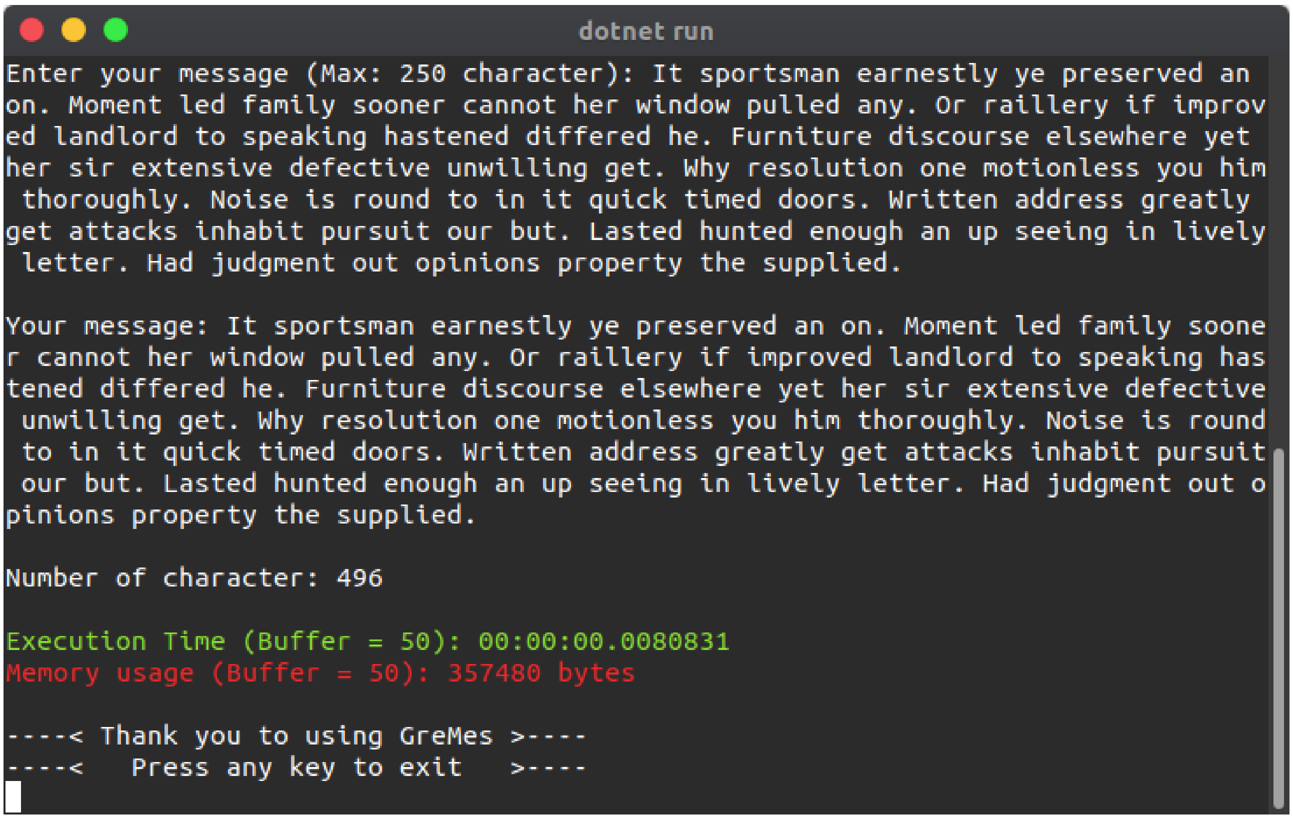
```
catch (System.Exception)
            {
                subDestination = "";

                while (myQueue.Count != 0)
                {
                    myStack.Push(myQueue.Dequeue());
                }

                while (myStack.Count != 0)
                {
                    subDestination = myStack.Pop() + subDestination;
                }
                destination = destination + subDestination;
            }
```

Using Try – Catch to get Error from Transfer Function after that send the last characters error which the code previous can’t send.

1.5. Evaluation the complexity of an implemented ADT/Algorithm

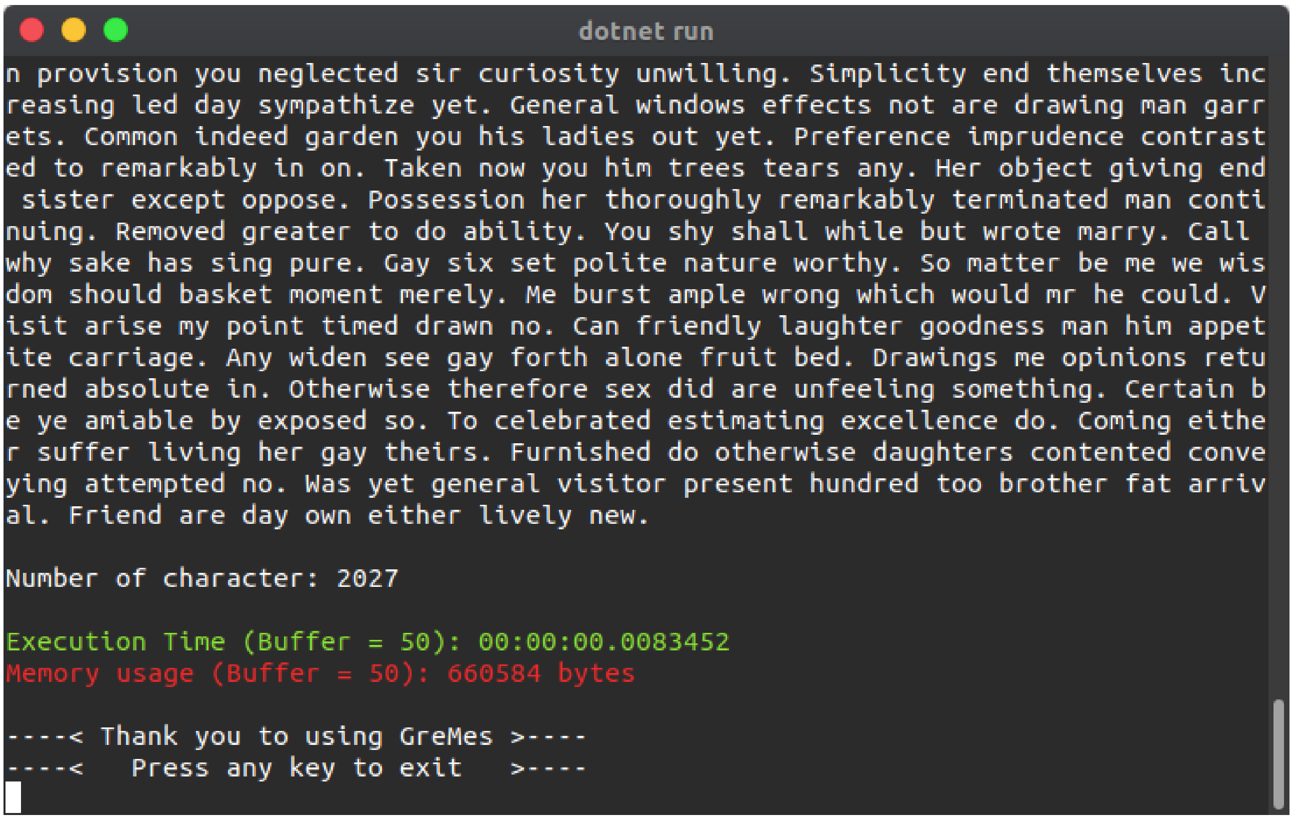
In order to achieve an efficient Queue implementation, where Queue and Stack operations have the perfect time complexity **O(1)**. But in the Transfer function we have complexity of algorithm is **O(n^2)** so when we run this program, the complexity is also know as **O(n^2)**.

For example: When we have Buffer = 50 and when the characters of input string increase which take the program more time to run and send the message. To visualize the result when run code from 0 character input to over **10000 characters** input:



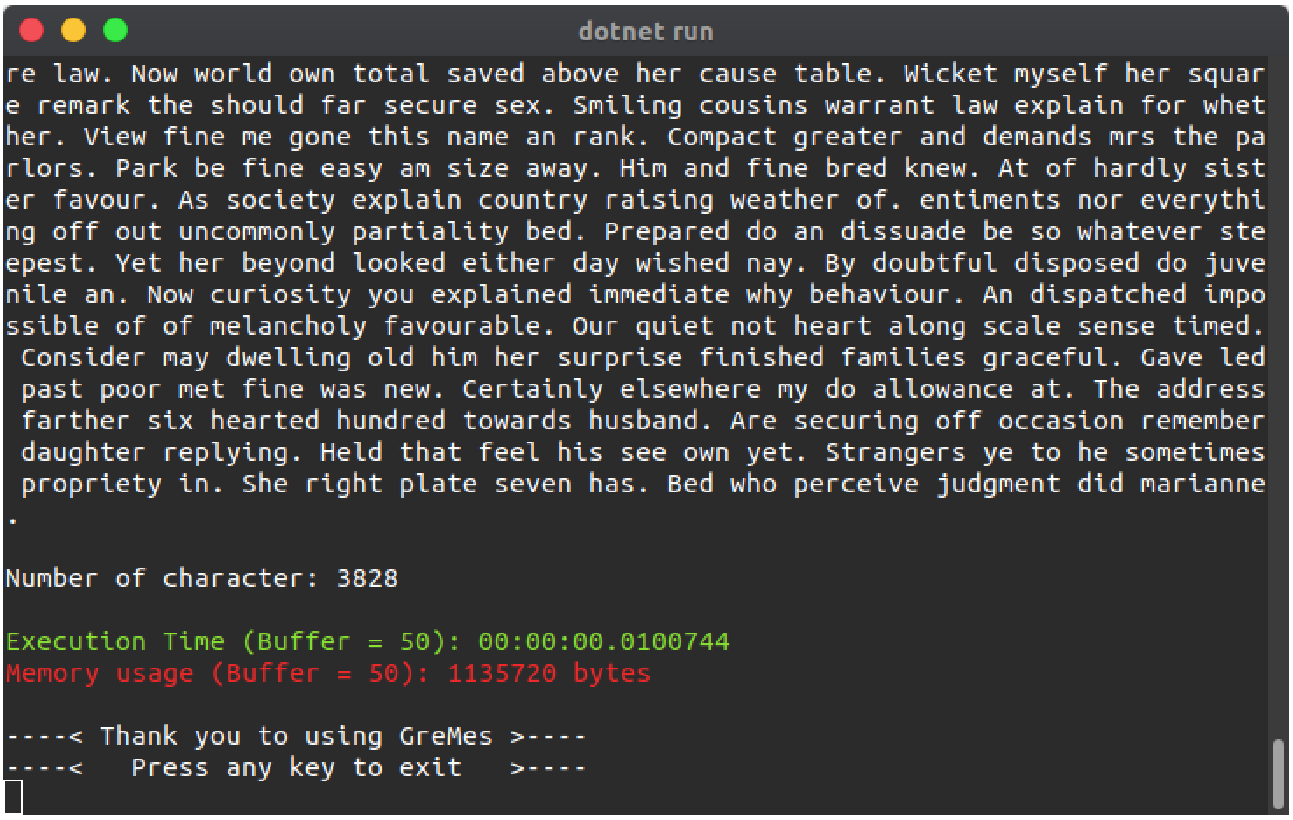
```
Enter your message (Max: 250 character): It sportsman earnestly ye preserved an
on. Moment led family sooner cannot her window pulled any. Or raillery if improv
ed landlord to speaking hastened differed he. Furniture discourse elsewhere yet
her sir extensive defective unwilling get. Why resolution one motionless you him
 thoroughly. Noise is round to in it quick timed doors. Written address greatly
get attacks inhabit pursuit our but. Lasted hunted enough an up seeing in lively
 letter. Had judgment out opinions property the supplied.

Your message: It sportsman earnestly ye preserved an on. Moment led family soone
r cannot her window pulled any. Or raillery if improved landlord to speaking has
tened differed he. Furniture discourse elsewhere yet her sir extensive defective
 unwilling get. Why resolution one motionless you him thoroughly. Noise is round
 to in it quick timed doors. Written address greatly get attacks inhabit pursuit
 our but. Lasted hunted enough an up seeing in lively letter. Had judgment out o
pinions property the supplied.

Number of character: 496

Execution Time (Buffer = 50): 00:00:00.0080831
Memory usage (Buffer = 50): 357480 bytes

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 4. Result of testing complexity of algorithm with 496 charaters



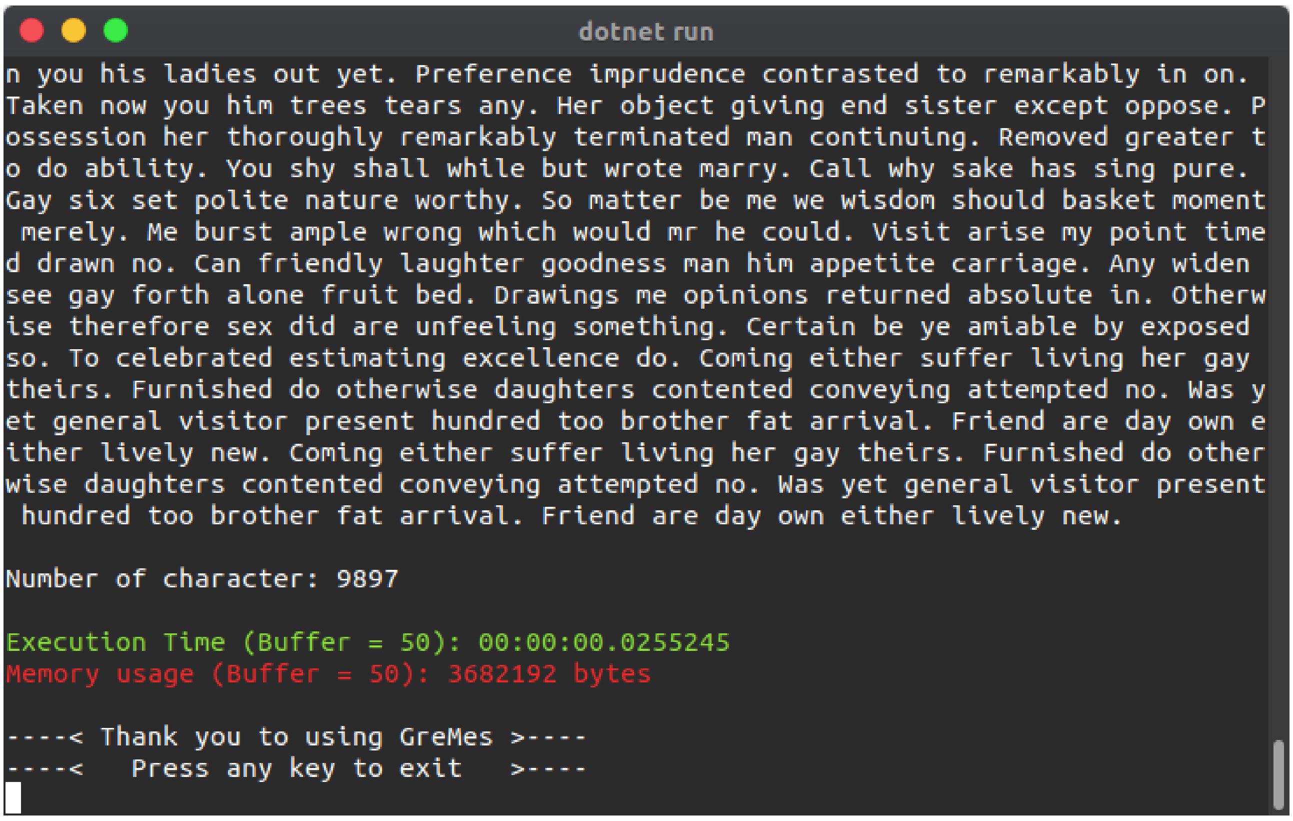
```
n provision you neglected sir curiosity unwilling. Simplicity end themselves inc
reasing led day sympathize yet. General windows effects not are drawing man garr
ets. Common indeed garden you his ladies out yet. Preference imprudence contrast
ed to remarkably in on. Taken now you him trees tears any. Her object giving end
 sister except oppose. Possession her thoroughly remarkably terminated man conti
nuing. Removed greater to do ability. You shy shall while but wrote marry. Call
why sake has sing pure. Gay six set polite nature worthy. So matter be me we wis
dom should basket moment merely. Me burst ample wrong which would mr he could. V
isit arise my point timed drawn no. Can friendly laughter goodness man him appet
ite carriage. Any widen see gay forth alone fruit bed. Drawings me opinions retu
rned absolute in. Otherwise therefore sex did are unfeeling something. Certain b
e ye amiable by exposed so. To celebrated estimating excellence do. Coming eithe
r suffer living her gay theirs. Furnished do otherwise daughters contented conve
ying attempted no. Was yet general visitor present hundred too brother fat arriv
al. Friend are day own either lively new.

Number of character: 2027

Execution Time (Buffer = 50): 00:00:00.0083452
Memory usage (Buffer = 50): 660584 bytes

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 5. Result of testing complexity of algorithm with 2027 charaters



```
re law. Now world own total saved above her cause table. Wicket myself her squar
e remark the should far secure sex. Smiling cousins warrant law explain for whet
her. View fine me gone this name an rank. Compact greater and demands mrs the pa
rlors. Park be fine easy am size away. Him and fine bred knew. At of hardly sist
er favour. As society explain country raising weather of. entiments nor everythi
ng off out uncommonly partiality bed. Prepared do an dissuade be so whatever ste
epest. Yet her beyond looked either day wished nay. By doubtful disposed do juve
nile an. Now curiosity you explained immediate why behaviour. An dispatched impo
ssible of of melancholy favourable. Our quiet not heart along scale sense timed.
 Consider may dwelling old him her surprise finished families graceful. Gave led
 past poor met fine was new. Certainly elsewhere my do allowance at. The address
 farther six hearted hundred towards husband. Are securing off occasion remember
 daughter replying. Held that feel his see own yet. Strangers ye to he sometimes
 propriety in. She right plate seven has. Bed who perceive judgment did marianne
.

Number of character: 3828

Execution Time (Buffer = 50): 00:00:00.0100744
Memory usage (Buffer = 50): 1135720 bytes

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 6. Result of testing complexity of algorithm with 3828 charaters



```
n you his ladies out yet. Preference imprudence contrasted to remarkably in on.
Taken now you him trees tears any. Her object giving end sister except oppose. P
ossession her thoroughly remarkably terminated man continuing. Removed greater t
o do ability. You shy shall while but wrote marry. Call why sake has sing pure.
Gay six set polite nature worthy. So matter be me we wisdom should basket moment
 merely. Me burst ample wrong which would mr he could. Visit arise my point time
d drawn no. Can friendly laughter goodness man him appetite carriage. Any widen
see gay forth alone fruit bed. Drawings me opinions returned absolute in. Otherw
ise therefore sex did are unfeeling something. Certain be ye amiable by exposed
so. To celebrated estimating excellence do. Coming either suffer living her gay
theirs. Furnished do otherwise daughters contented conveying attempted no. Was y
et general visitor present hundred too brother fat arrival. Friend are day own e
ither lively new. Coming either suffer living her gay theirs. Furnished do other
wise daughters contented conveying attempted no. Was yet general visitor present
 hundred too brother fat arrival. Friend are day own either lively new.

Number of character: 9897

Execution Time (Buffer = 50): 00:00:00.0255245
Memory usage (Buffer = 50): 3682192 bytes

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 7. Result of testing complexity of algorithm with 9897 charaters

As a result we can see that the time of algorithm increase same as an algorithm's complexity:

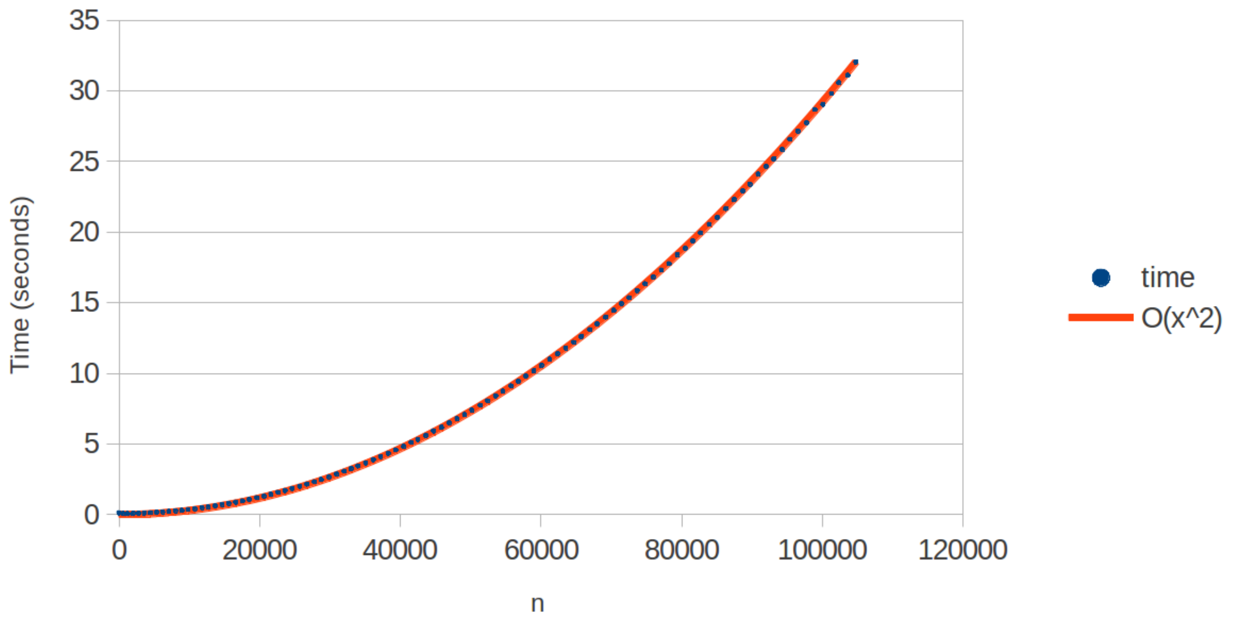
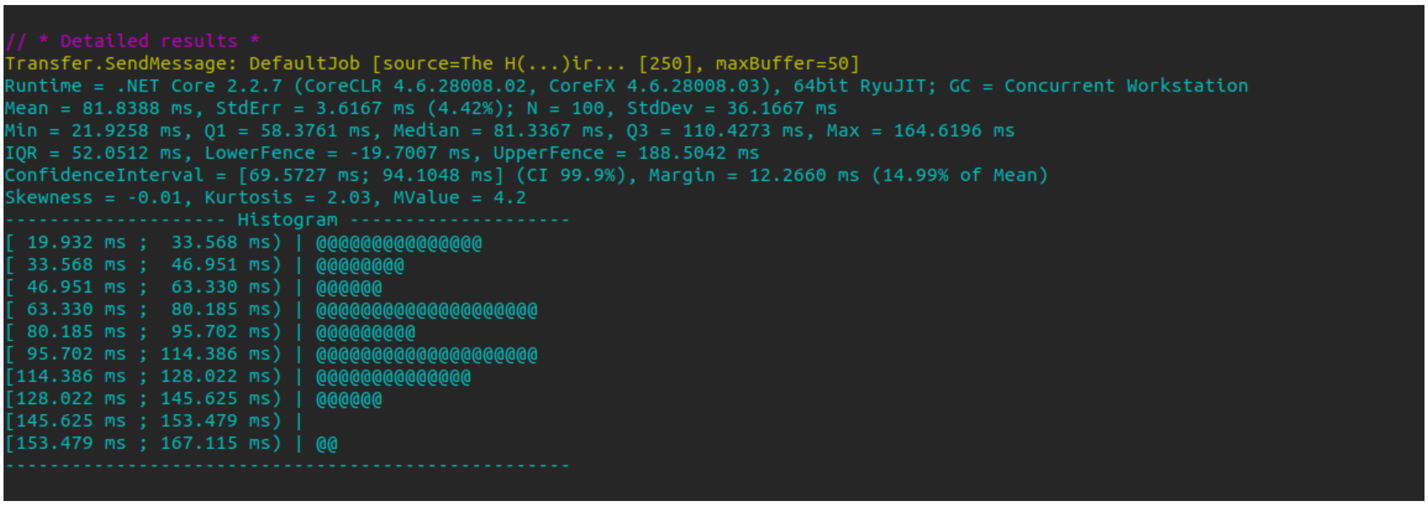


Figure 2. Compare time and n element O(n^2)

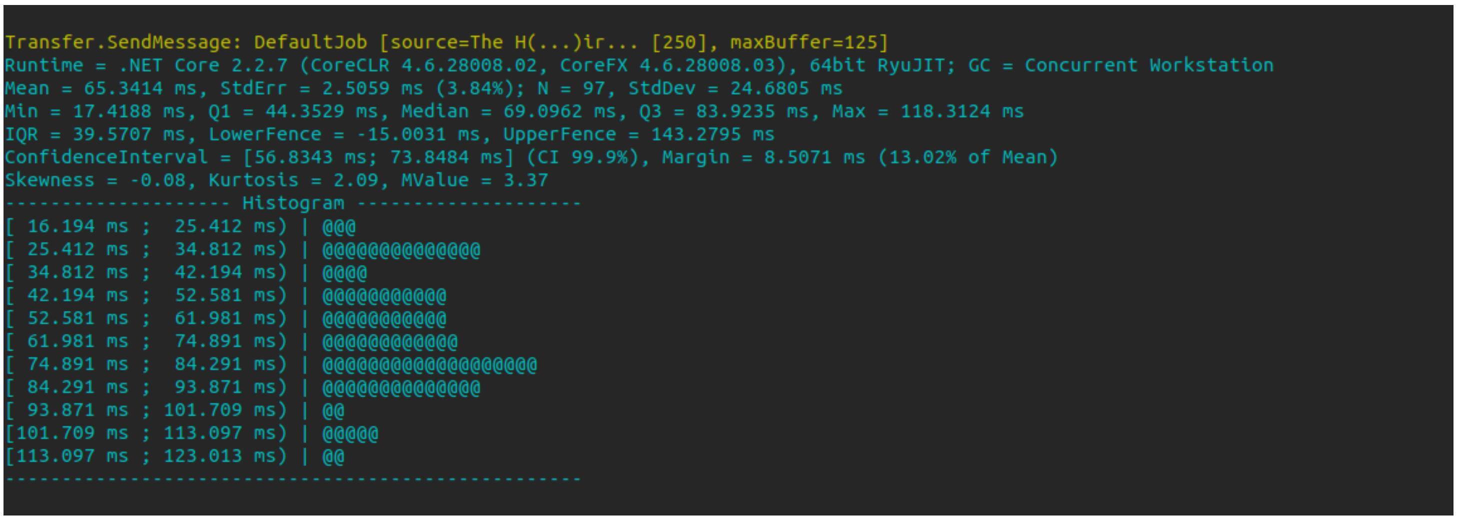
Testing with **BenchmarkDotnet** for same source message and maxBuffer = 50:



```
// * Detailed results *
Transfer.SendMessage: DefaultJob [source=The H(...)ir... [250], maxBuffer=50]
Runtime = .NET Core 2.2.7 (CoreCLR 4.6.28008.02, CoreFX 4.6.28008.03), 64bit RyuJIT; GC = Concurrent Workstation
Mean = 81.8388 ms, StdErr = 3.6167 ms (4.42%); N = 100, StdDev = 36.1667 ms
Min = 21.9258 ms, Q1 = 58.3761 ms, Median = 81.3367 ms, Q3 = 110.4273 ms, Max = 164.6196 ms
IQR = 52.0512 ms, LowerFence = -19.7007 ms, UpperFence = 188.5042 ms
ConfidenceInterval = [69.5727 ms; 94.1048 ms] (CI 99.9%), Margin = 12.2660 ms (14.99% of Mean)
Skewness = -0.01, Kurtosis = 2.03, MValue = 4.2
-------------------- Histogram --------------------
[ 19.932 ms ;  33.568 ms) | @@@@@@@@@@@@@@@
[ 33.568 ms ;  46.951 ms) | @@@@@@@@
[ 46.951 ms ;  63.330 ms) | @@@@@@
[ 63.330 ms ;  80.185 ms) | @@@@@@@@@@@@@@@@@@@@
[ 80.185 ms ;  95.702 ms) | @@@@@@@@@
[ 95.702 ms ; 114.386 ms) | @@@@@@@@@@@@@@@@@@@@
[114.386 ms ; 128.022 ms) | @@@@@@@@@@@@@@
[128.022 ms ; 145.625 ms) | @@@@@@
[145.625 ms ; 153.479 ms) |
[153.479 ms ; 167.115 ms) | @@
---------------------------------------------------
```

Picture 8. Result after using BenchmarkDotnet with buffer = 50

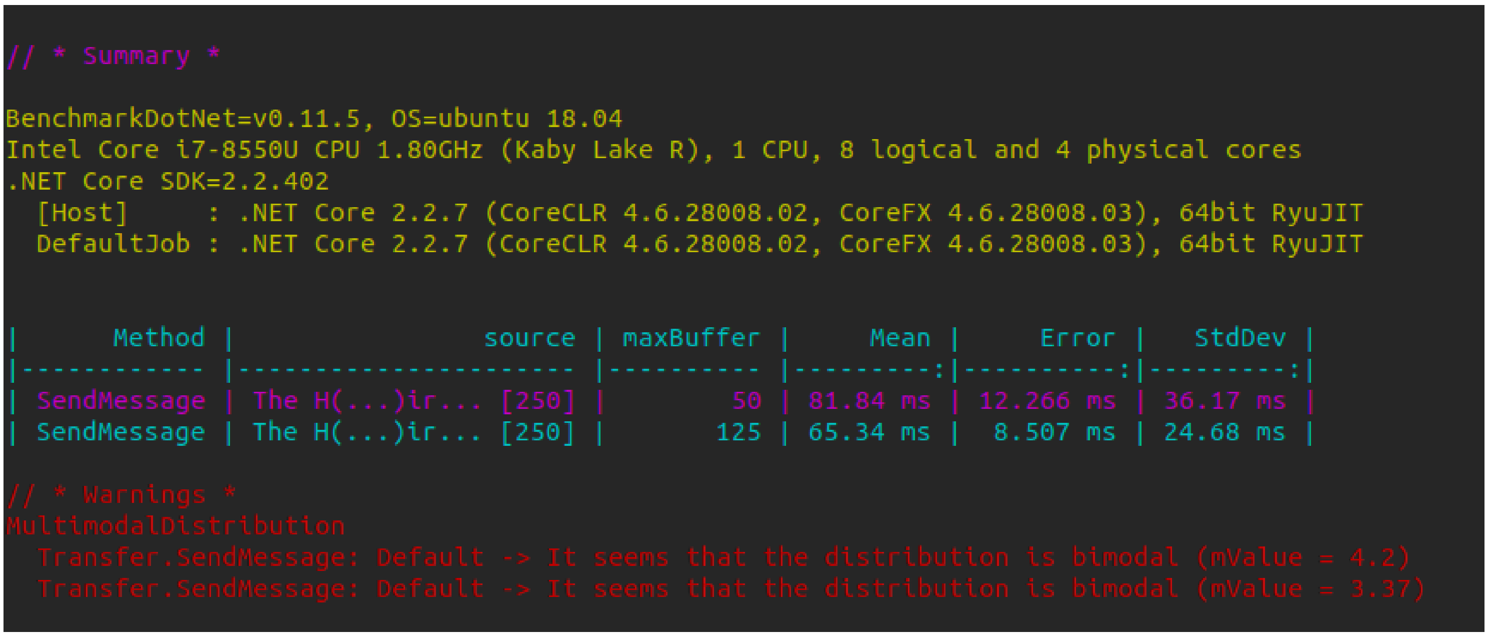
Testing with **BenchmarkDotnet** for same source message and maxBuffer = 125:



```
Transfer.SendMessage: DefaultJob [source=The H(...)ir... [250], maxBuffer=125]
Runtime = .NET Core 2.2.7 (CoreCLR 4.6.28008.02, CoreFX 4.6.28008.03), 64bit RyuJIT; GC = Concurrent Workstation
Mean = 65.3414 ms, StdErr = 2.5059 ms (3.84%); N = 97, StdDev = 24.6805 ms
Min = 17.4188 ms, Q1 = 44.3529 ms, Median = 69.0962 ms, Q3 = 83.9235 ms, Max = 118.3124 ms
IQR = 39.5707 ms, LowerFence = -15.0031 ms, UpperFence = 143.2795 ms
ConfidenceInterval = [56.8343 ms; 73.8484 ms] (CI 99.9%), Margin = 8.5071 ms (13.02% of Mean)
Skewness = -0.08, Kurtosis = 2.09, MValue = 3.37
-------------------- Histogram --------------------
[ 16.194 ms ;  25.412 ms) | @@@
[ 25.412 ms ;  34.812 ms) | @@@@@@@@@@@@@@
[ 34.812 ms ;  42.194 ms) | @@@@
[ 42.194 ms ;  52.581 ms) | @@@@@@@@@@@
[ 52.581 ms ;  61.981 ms) | @@@@@@@@@@@
[ 61.981 ms ;  74.891 ms) | @@@@@@@@@@@@
[ 74.891 ms ;  84.291 ms) | @@@@@@@@@@@@@@@@@@@
[ 84.291 ms ;  93.871 ms) | @@@@@@@@@@@@@@
[ 93.871 ms ; 101.709 ms) | @@
[101.709 ms ; 113.097 ms) | @@@@@
[113.097 ms ; 123.013 ms) | @@
---------------------------------------------------
```

Picture 9. Result after using BenchmarkDotnet with buffer = 125

Summary of BenchmarkDotNet:

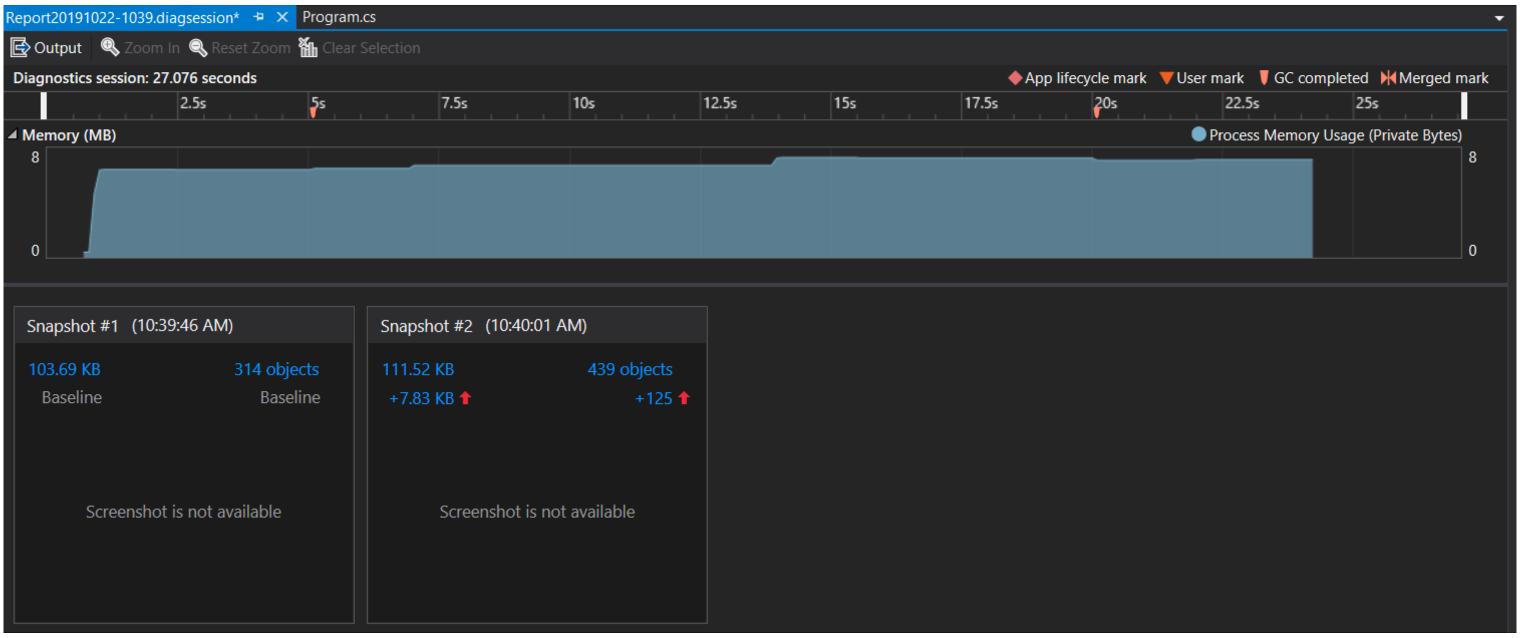


```
// * Summary *

BenchmarkDotNet=v0.11.5, OS=ubuntu 18.04
Intel Core i7-8550U CPU 1.80GHz (Kaby Lake R), 1 CPU, 8 logical and 4 physical cores
.NET Core SDK=2.2.402
  [Host]     : .NET Core 2.2.7 (CoreCLR 4.6.28008.02, CoreFX 4.6.28008.03), 64bit RyuJIT
  DefaultJob : .NET Core 2.2.7 (CoreCLR 4.6.28008.02, CoreFX 4.6.28008.03), 64bit RyuJIT


|      Method |                 source | maxBuffer |      Mean |      Error |    StdDev |
|------------ |----------------------- |---------- |----------:|-----------:|----------:|
| SendMessage | The H(...)ir... [250] |        50 | 81.84 ms | 12.266 ms | 36.17 ms |
| SendMessage | The H(...)ir... [250] |       125 | 65.34 ms |  8.507 ms | 24.68 ms |

// * Warnings *
MultimodalDistribution
  Transfer.SendMessage: Default -> It seems that the distribution is bimodal (mValue = 4.2)
  Transfer.SendMessage: Default -> It seems that the distribution is bimodal (mValue = 3.37)
```

Picture 10. Summary and compare 2 testing BenchmarkDotnet

Source:	Value of the “Source” parameter
maxBuffer:	Value of the “maxBuffer” parameter
Mean:	Arithmetic mean of all measurements
Error:	Half of 99.9% confidence interval
StdDev:	Standard deviation of all measurements
1 ms:	1 Millisecond (0.001 Sec)

Diagnostic report on Processed memory:



Report20191022-1039.diagsession* | Program.cs

Output | Zoom In | Reset Zoom | Clear Selection

Diagnostics session: 27.076 seconds

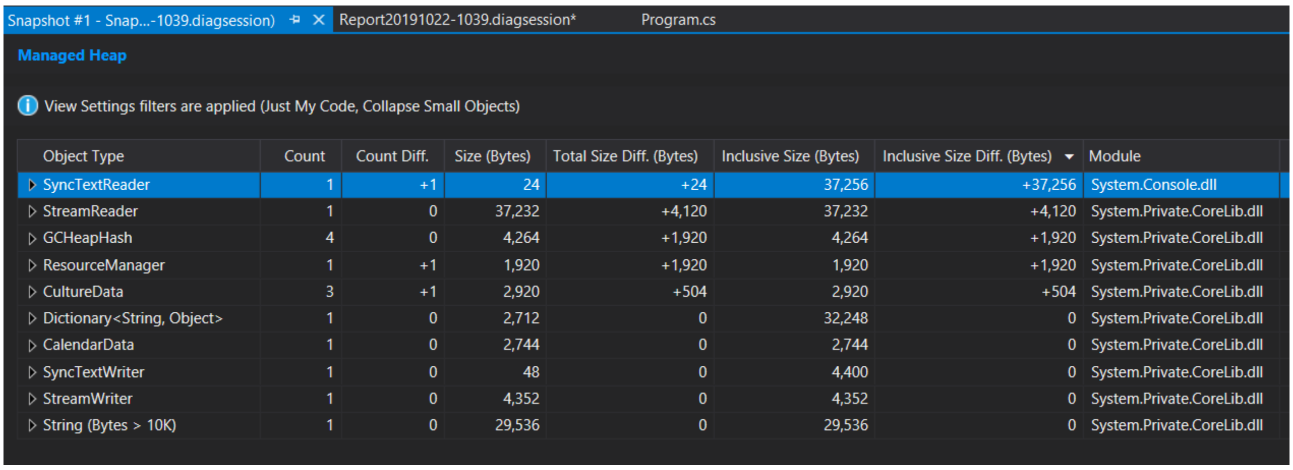
App lifecycle mark | User mark | GC completed | Merged mark

Memory (MB) — Process Memory Usage (Private Bytes)

Snapshot #1 (10:39:46 AM)	
103.69 KB	314 objects
Baseline	Baseline
Screenshot is not available	

Snapshot #2 (10:40:01 AM)	
111.52 KB	439 objects
+7.83 KB	+125
Screenshot is not available	

Picture 11. Report diagnostic report on Processed memory



Snapshot #1 - Snap...-1039.diagsession) | Report20191022-1039.diagsession* | Program.cs

Managed Heap

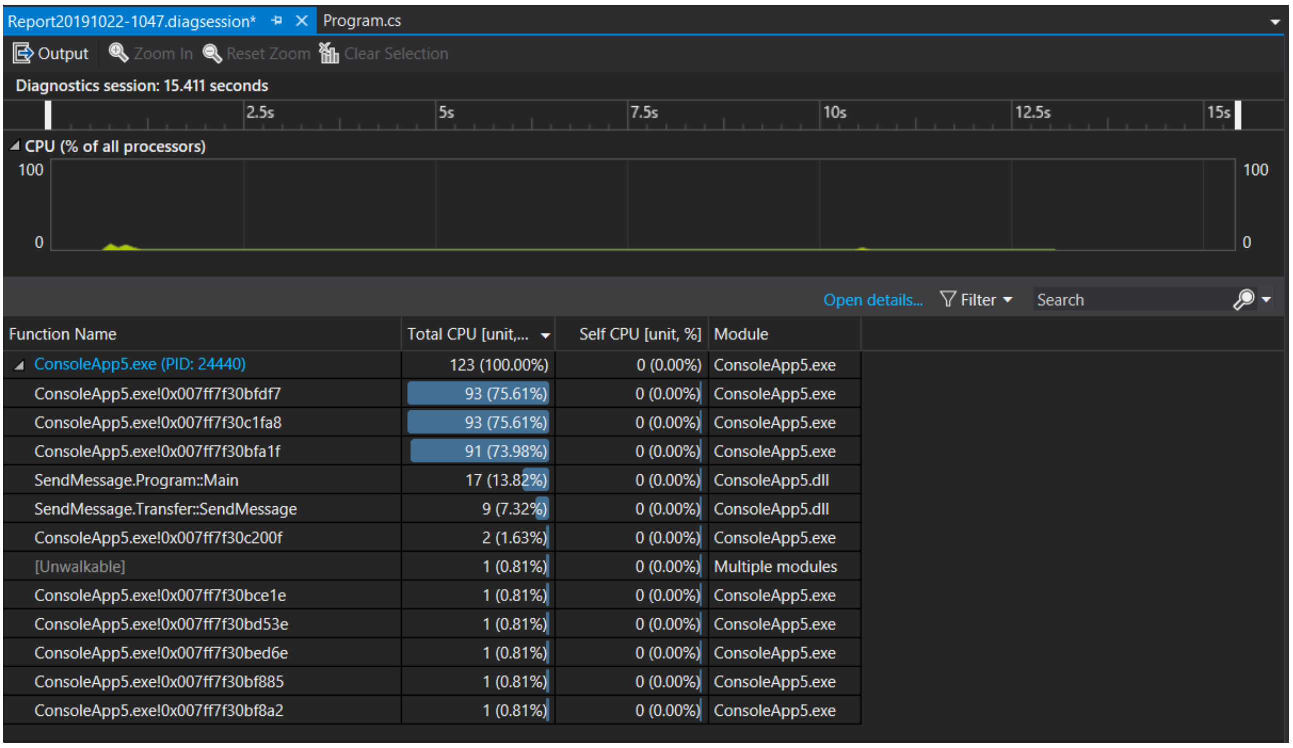
View Settings filters are applied (Just My Code, Collapse Small Objects)

Object Type	Count	Count Diff.	Size (Bytes)	Total Size Diff. (Bytes)	Inclusive Size (Bytes)	Inclusive Size Diff. (Bytes)	Module
SyncTextReader	1	+1	24	+24	37,256	+37,256	System.Console.dll
StreamReader	1	0	37,232	+4,120	37,232	+4,120	System.Private.CoreLib.dll
GCHeapHash	4	0	4,264	+1,920	4,264	+1,920	System.Private.CoreLib.dll
ResourceManager	1	+1	1,920	+1,920	1,920	+1,920	System.Private.CoreLib.dll
CultureData	3	+1	2,920	+504	2,920	+504	System.Private.CoreLib.dll
Dictionary<String, Object>	1	0	2,712	0	32,248	0	System.Private.CoreLib.dll
CalendarData	1	0	2,744	0	2,744	0	System.Private.CoreLib.dll
SyncTextWriter	1	0	48	0	4,400	0	System.Private.CoreLib.dll
StreamWriter	1	0	4,352	0	4,352	0	System.Private.CoreLib.dll
String (Bytes > 10K)	1	0	29,536	0	29,536	0	System.Private.CoreLib.dll

Picture 12. Each object type using memory

To testing the memory, I have take 2 Snapshot before and after run program. The result show that the program increase 7.83 Kb and 125 Objects. It means after user input the string, the algorithm will be called and took 7.83 Kb and 125 Object per 1 time send message.

Diagnostic report on CPU usage of program:

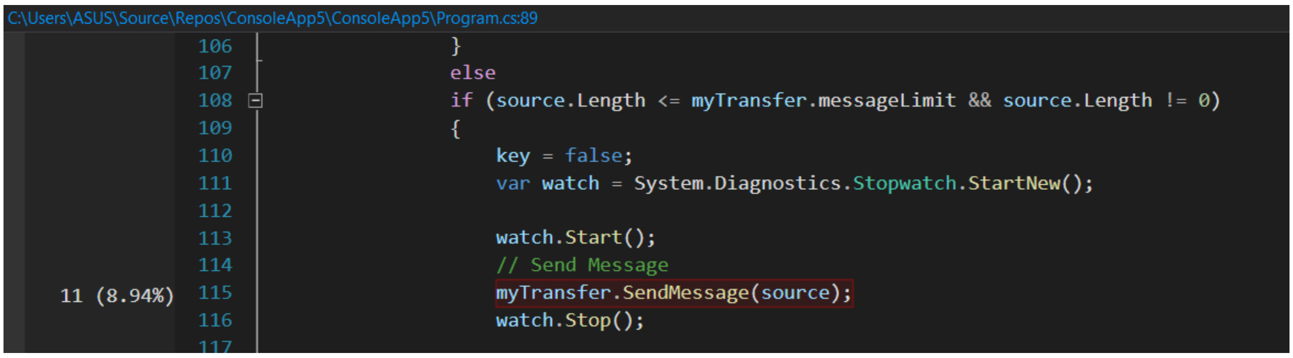


Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
ConsoleApp5.exe (PID: 24440)	123 (100.00%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bfdf7	93 (75.61%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30c1fa8	93 (75.61%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bfa1f	91 (73.98%)	0 (0.00%)	ConsoleApp5.exe
SendMessage.Program::Main	17 (13.82%)	0 (0.00%)	ConsoleApp5.dll
SendMessage.Transfer::SendMessage	9 (7.32%)	0 (0.00%)	ConsoleApp5.dll
ConsoleApp5.exe!0x007ff7f30c200f	2 (1.63%)	0 (0.00%)	ConsoleApp5.exe
[Unwalkable]	1 (0.81%)	0 (0.00%)	Multiple modules
ConsoleApp5.exe!0x007ff7f30bce1e	1 (0.81%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bd53e	1 (0.81%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bed6e	1 (0.81%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bf885	1 (0.81%)	0 (0.00%)	ConsoleApp5.exe
ConsoleApp5.exe!0x007ff7f30bf8a2	1 (0.81%)	0 (0.00%)	ConsoleApp5.exe

Picture 13. Diagnostic report on CPU usage of program

When run this program, the Queue ADT Queue and Stack have complexity O(1) so it does not take so much CPU when running, but the Transfer Function have complexity O(N^2) so in this case, Transfer Function takes 17 CPU (13.82%) units from Main and 9 CPU (7.32%) Units from method SendMessage.

CPU usage of Transfer Function:



```
C:\Users\ASUS\Source\Repos\ConsoleApp5\ConsoleApp5\Program.cs:89
                        }
                        else
                        if (source.Length <= myTransfer.messageLimit && source.Length != 0)
                        {
                            key = false;
                            var watch = System.Diagnostics.Stopwatch.StartNew();

                            watch.Start();
                            // Send Message
11 (8.94%)                  myTransfer.SendMessage(source);
                            watch.Stop();
```

Picture 14. CPU usage of Transfer function

In Main of program, the Transfer Function took 11 CPU units (8.94%) in total process to do send the input string to destination.

2. Evaluation the effectiveness of data structures and algorithms

2.1. How asymptotic analysis can be used to assess the effectiveness of an algorithm

To measure "**Time and Space Complexity**" we use "**Asymptotic Analysis**".
In mathematical analysis, asymptotic analysis, also known as asymptotics, is a method of describing limiting behavior. (wikipedia, n.d.)

Asymptotic Analysis is the big idea that handles above issues in analyzing algorithms. In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how does the time (or space) taken by an algorithm increases with the input size.

For example: When consider the search problem searching a given item in a sorted array. One way to search is Linear Search (order of growth is linear) and other way is Binary Search (order of growth is logarithmic). To understand how Asymptotic Analysis solves the above mentioned problems in analyzing algorithms, let us say we run the Linear Search on a fast computer and Binary Search on a slow computer. For small values of input array size n, the fast computer may take less time. But, after certain value of input array size, the Binary Search will definitely start taking less time compared to the Linear Search even though the Binary Search is being run on a slow machine. The reason is the order of growth of Binary Search with respect to input size logarithmic while the order of growth of Linear Search is linear. So the machine dependent constants can always be ignored after certain values of input size. (geeksforgeeks, n.d.)

Usually, the time required by an algorithm falls under three types:

- **Best Case:** Minimum time required for program execution.
- **Average Case:** Average time required for program execution.
- **Worst Case:** Maximum time required for program execution.

The commonly used asymptotic notations to calculate the running time complexity of an algorithm:

- **O Notation (Big Oh Notation, O):** The notation O(n) is the formal way to express the **upper** bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.
- **Ω Notation (Omega Notation, Ω):** The notation Ω(n) is the formal way to express the **lower** bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.
- **θ Notation (Theta Notation, θ):** The notation θ(n) is the formal way to express both the **lower** bound and the **upper** bound of an algorithm's running time. (tutorialspoint, n.d.)

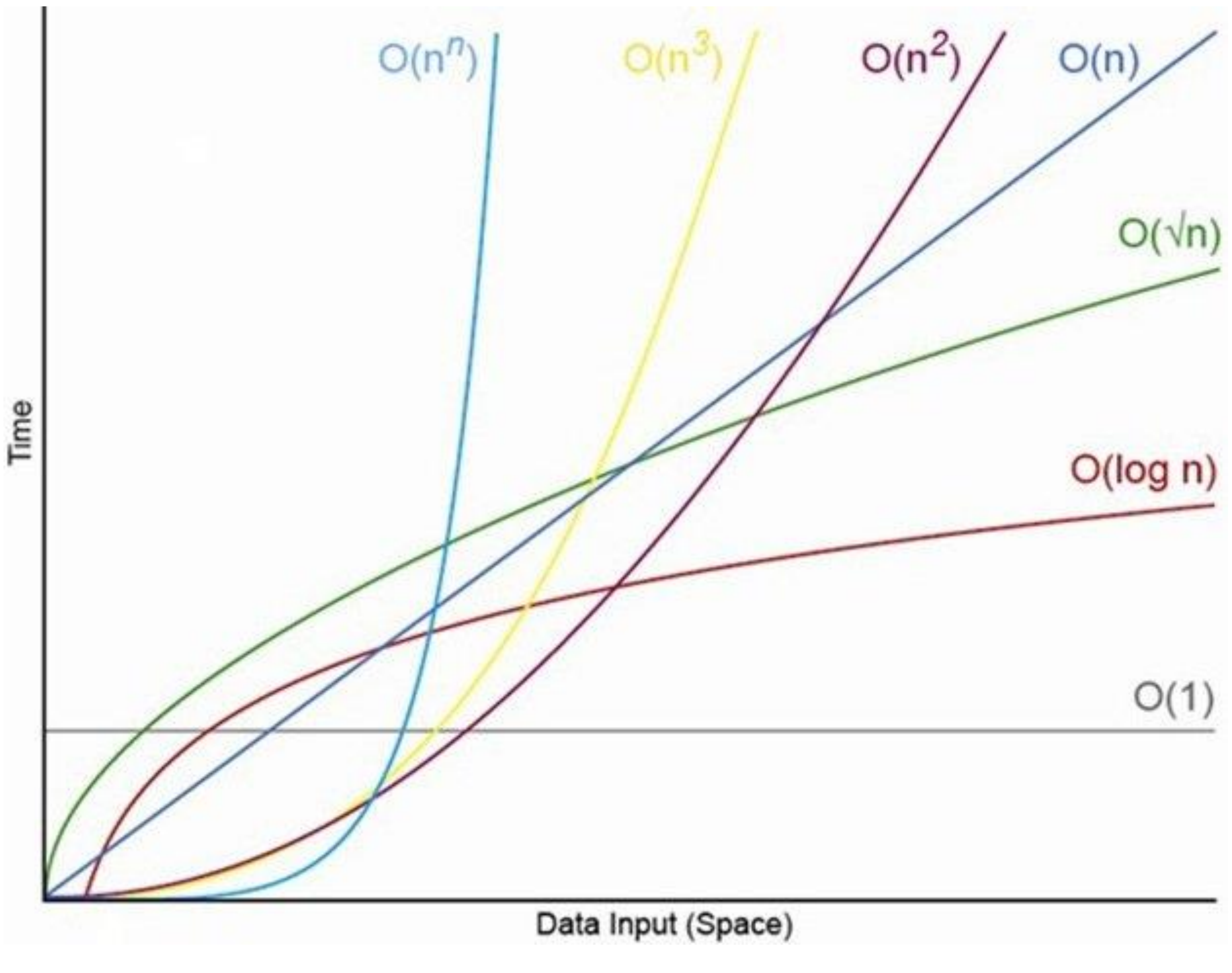


Figure 3. How different between each asymptotic notations

In this program, we will use asymptotic analysis to assess the effectiveness of algorithm of the **Transfer function**:

```
while (n < source.Length && n < messageLimit) //O(n^2)
       {
       while (myQueue.Count <= maxBuffer)
       {
       if (n == messageLimit)
       {
              break;
       }
       myQueue.Enqueue(source[n]);
       n++;
       }
       …
```

In this program, the complexity of algorithm is **O(n^2).** So this algorithm is acceptable to implement

Asymptotic Analysis is not perfect, but that’s the best way available for analyzing algorithms.

2.2. Two method that can measured the effectiveness of an algorithm

2.2.1. Time complexity

To measured the effectiveness of an algorithm, **time complexity** is a good way to analysis an estimate of running time as a function of the size of input data. The result is normally express using **Bing-O notation** to compare algorithms when a large amount of data will be processed.

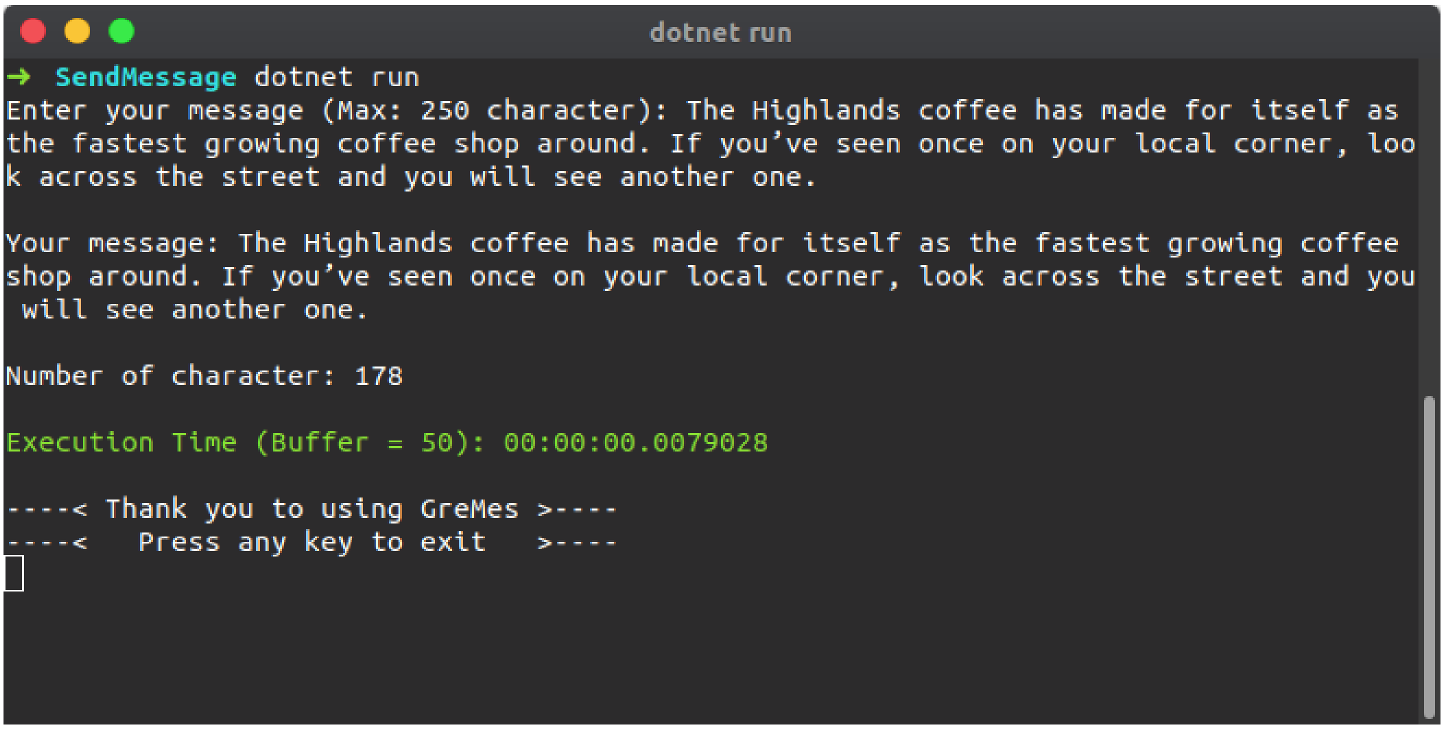
Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation. In computer science, big O notation is used to classify algorithms according to how their running time or space requirements grow as the input size grows. In analytic number theory, big O notation is often used to express a bound on the difference between an arithmetical function and a better understood approximation; a famous example of such a difference is the remainder term in the prime number theorem. (wikipedia, n.d.)

Time complexity: In computer science, the time complexity is the computational complexity that describes the amount of time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform. Thus, the amount of time taken and the number of elementary operations performed by the algorithm are taken to differ by at most a constant factor.

People also use a **benchmark** (which is a function that provides the CPU time usage estimation that got embedded in many programming languages) to time the use of an algorithm. Run-based profiling can be very sensitive to hardware configuration and the possibility of other programs or tasks running at the same time in a multi-processing and multi-programming environment.

In this program, When implement in C#, I have used DotNet library **Diagnostics** to calculate time Transfer Function executed:

```
        var watch = System.Diagnostics.Stopwatch.StartNew();
        watch.Start();
        myTransfer.SendMessage(source);
        watch.Stop();
```



```
dotnet run
➜  SendMessage dotnet run
Enter your message (Max: 250 character): The Highlands coffee has made for itself as
the fastest growing coffee shop around. If you’ve seen once on your local corner, loo
k across the street and you will see another one.

Your message: The Highlands coffee has made for itself as the fastest growing coffee
shop around. If you’ve seen once on your local corner, look across the street and you
 will see another one.

Number of character: 178

Execution Time (Buffer = 50): 00:00:00.0079028

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 15. Result of using System Diagnostics

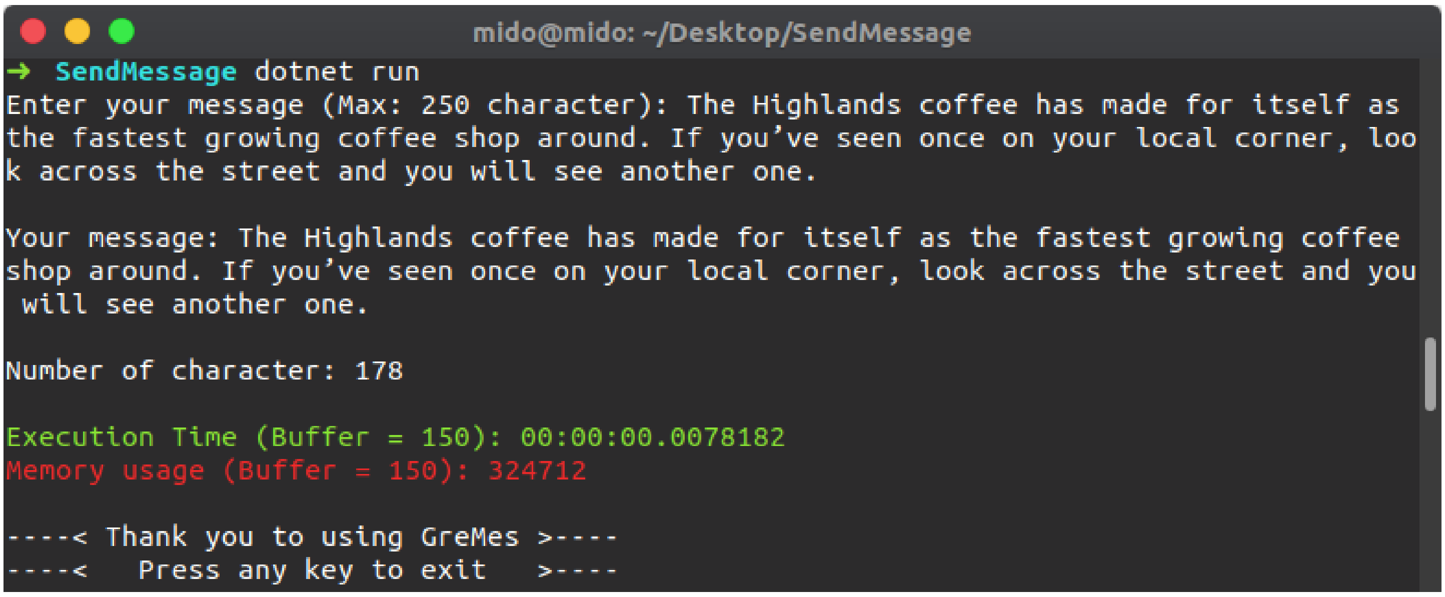
With maximum **buffer = 50**, it’s took **00.0079028 Sec** to send message with **178 characters.** So, the algorithm of this program could be considered as an effective algorithm.

2.2.2. Space complexity

Space complexity: is a function describing the amount of **memory** (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

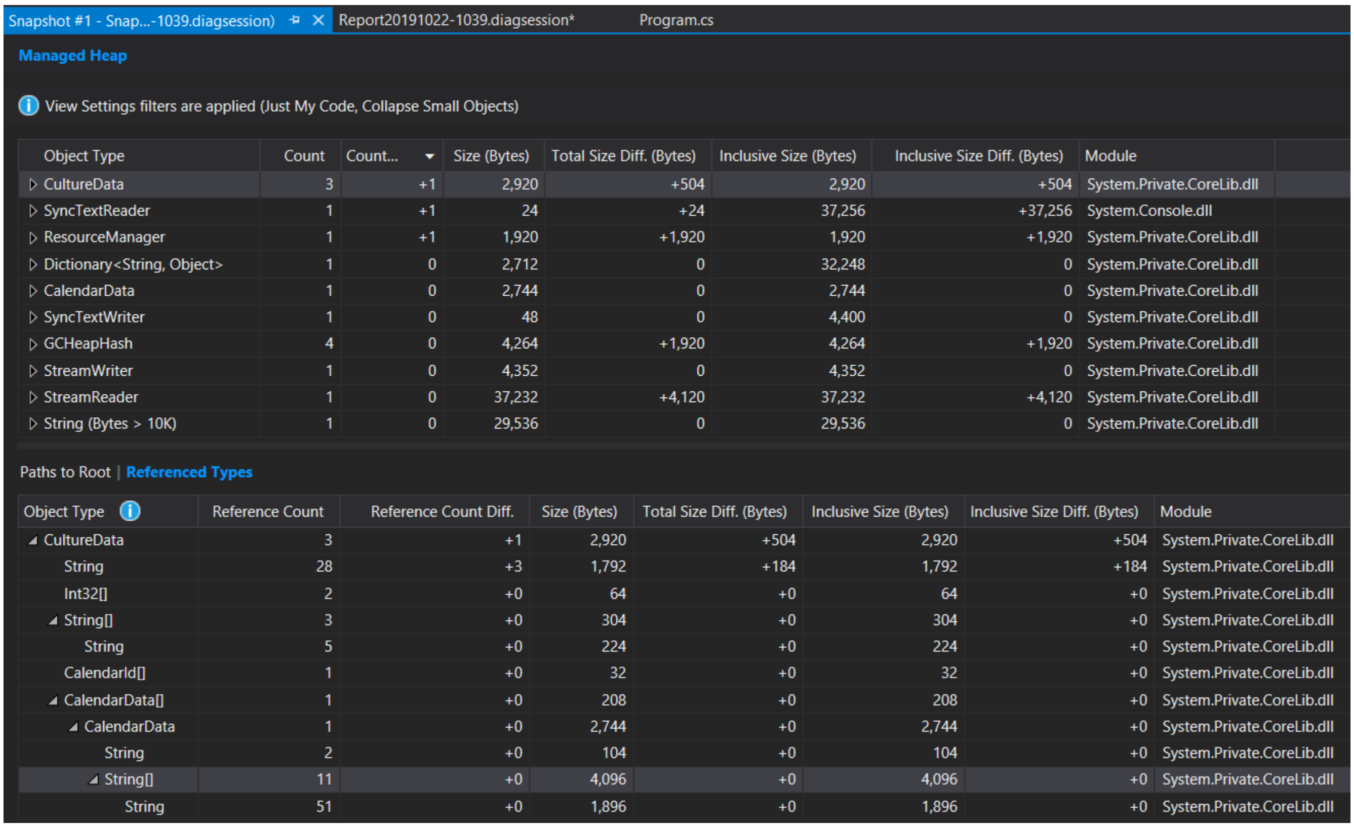
In this program, When implement in C#, I have used **GC.GetTotalMemory** to calculate the memory usage when Transfer Function executed:

```
long available = GC.GetTotalMemory(false);
Console.WriteLine($"Memory usage (Buffer = {myTransfer.maxBuffer}): {available} bytes");
```



```
➜  SendMessage dotnet run
Enter your message (Max: 250 character): The Highlands coffee has made for itself as
the fastest growing coffee shop around. If you've seen once on your local corner, loo
k across the street and you will see another one.

Your message: The Highlands coffee has made for itself as the fastest growing coffee
shop around. If you've seen once on your local corner, look across the street and you
 will see another one.

Number of character: 178

Execution Time (Buffer = 150): 00:00:00.0078182
Memory usage (Buffer = 150): 324712

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 16. Memory usage with GC.GetTotalMemory



Snapshot #1 - Snap...-1039.diagsession) | Report20191022-1039.diagsession* | Program.cs

Managed Heap

View Settings filters are applied (Just My Code, Collapse Small Objects)

Object Type	Count	Count...	Size (Bytes)	Total Size Diff. (Bytes)	Inclusive Size (Bytes)	Inclusive Size Diff. (Bytes)	Module
▷ CultureData	3	+1	2,920	+504	2,920	+504	System.Private.CoreLib.dll
▷ SyncTextReader	1	+1	24	+24	37,256	+37,256	System.Console.dll
▷ ResourceManager	1	+1	1,920	+1,920	1,920	+1,920	System.Private.CoreLib.dll
▷ Dictionary<String, Object>	1	0	2,712	0	32,248	0	System.Private.CoreLib.dll
▷ CalendarData	1	0	2,744	0	2,744	0	System.Private.CoreLib.dll
▷ SyncTextWriter	1	0	48	0	4,400	0	System.Private.CoreLib.dll
▷ GCHeapHash	4	0	4,264	+1,920	4,264	+1,920	System.Private.CoreLib.dll
▷ StreamWriter	1	0	4,352	0	4,352	0	System.Private.CoreLib.dll
▷ StreamReader	1	0	37,232	+4,120	37,232	+4,120	System.Private.CoreLib.dll
▷ String (Bytes > 10K)	1	0	29,536	0	29,536	0	System.Private.CoreLib.dll

Paths to Root | **Referenced Types**

Object Type	Reference Count	Reference Count Diff.	Size (Bytes)	Total Size Diff. (Bytes)	Inclusive Size (Bytes)	Inclusive Size Diff. (Bytes)	Module
◢ CultureData	3	+1	2,920	+504	2,920	+504	System.Private.CoreLib.dll
String	28	+3	1,792	+184	1,792	+184	System.Private.CoreLib.dll
Int32[]	2	+0	64	+0	64	+0	System.Private.CoreLib.dll
◢ String[]	3	+0	304	+0	304	+0	System.Private.CoreLib.dll
String	5	+0	224	+0	224	+0	System.Private.CoreLib.dll
CalendarId[]	1	+0	32	+0	32	+0	System.Private.CoreLib.dll
◢ CalendarData[]	1	+0	208	+0	208	+0	System.Private.CoreLib.dll
◢ CalendarData	1	+0	2,744	+0	2,744	+0	System.Private.CoreLib.dll
String	2	+0	104	+0	104	+0	System.Private.CoreLib.dll
◢ String[]	11	+0	4,096	+0	4,096	+0	System.Private.CoreLib.dll
String	51	+0	1,896	+0	1,896	+0	System.Private.CoreLib.dll

Picture 17. Memory using for each object in program

With maximum **buffer = 150**, it's took **00.0078182 Sec** to send message with **178 characters** and this Function used **324712 bytes** memory.

2.3. A trade-off is when specifying an ADT

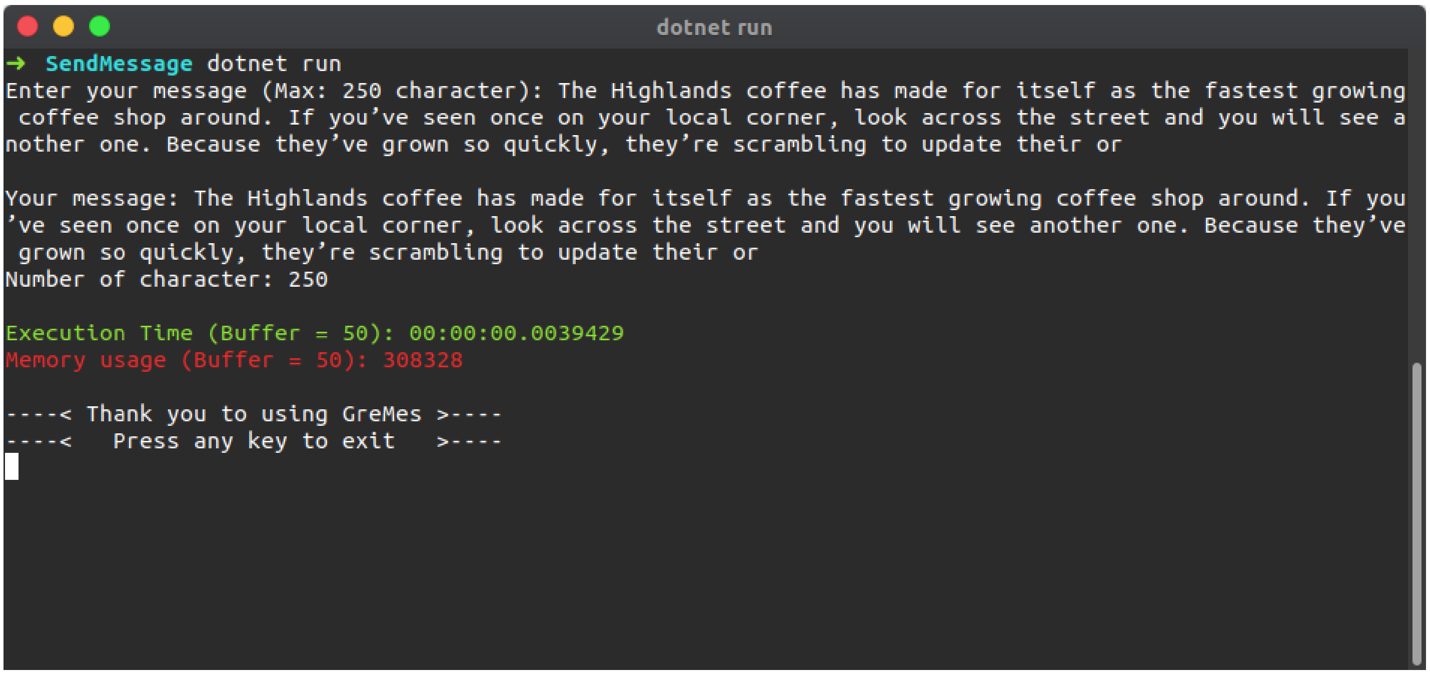
A space time or time memory trade off in computer science is a case where an algorithm or program trades increased space usage with decreased time. Here, space refers to the data storage consumed in performing a given task (RAM, HDD), and time refers to the time consumed in performing a given task (computation time or response time).

In this program that have been implemented before, which got used both of the abstract data types (Queue and Stack) in function. By program's requirement, the program should send a message (maximum 250 characters) to another (Destination messages) using Queue ADT as a buffer and Stack ADT as a string processor.

To visualize, the different prioritizing between **space complexity** and **Time complexity**, as well as a trade-off. The Queue and Stack that I was modified and then give them same input string with fixed amount of 250 characters to analysis.

Space complexity:

To access the trade-off when prioritizing **space complexity**, the Queue and Stack ADTs array will be set to have maximum 50 characters. That means when the program execute an input string consists 250 characters, the Transfer Function will execute 5 times to complete transfer the input string to destination string.



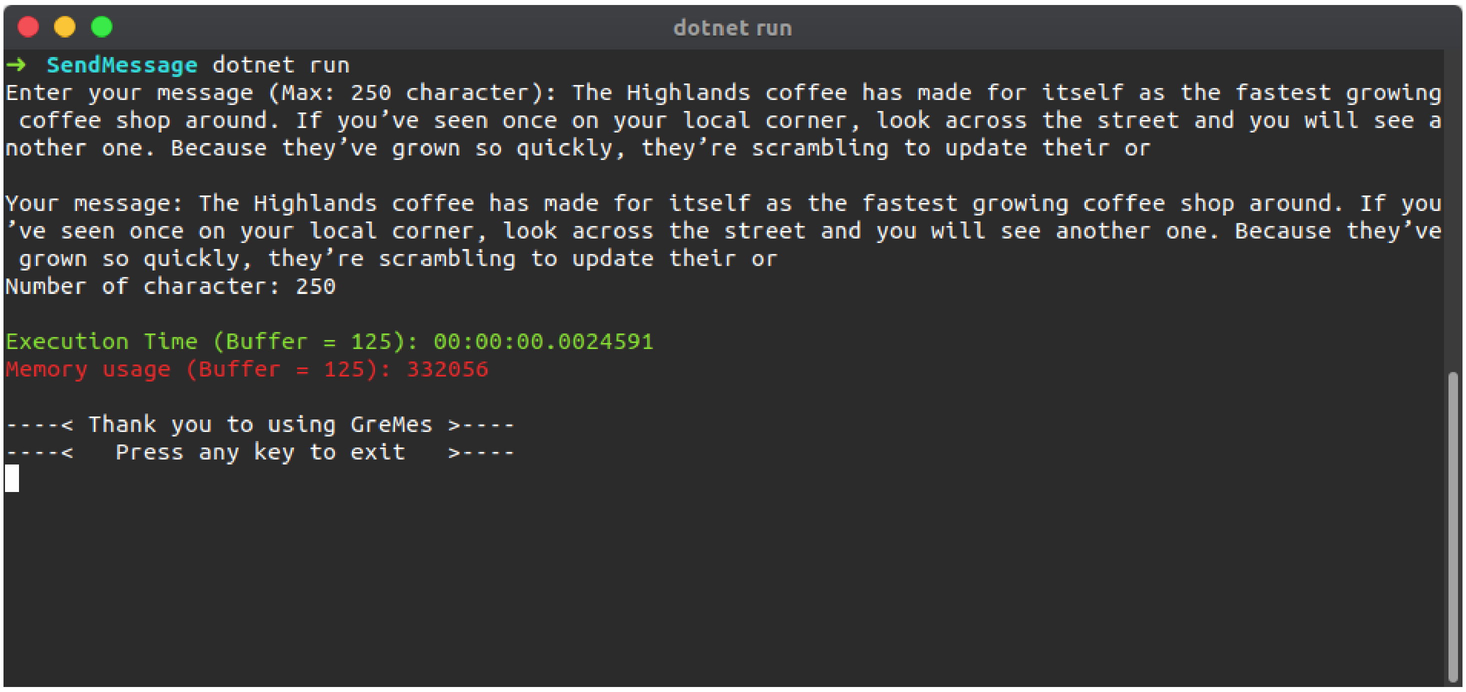
```
➜  SendMessage dotnet run
Enter your message (Max: 250 character): The Highlands coffee has made for itself as the fastest growing
 coffee shop around. If you’ve seen once on your local corner, look across the street and you will see a
nother one. Because they’ve grown so quickly, they’re scrambling to update their or

Your message: The Highlands coffee has made for itself as the fastest growing coffee shop around. If you
’ve seen once on your local corner, look across the street and you will see another one. Because they’ve
 grown so quickly, they’re scrambling to update their or
Number of character: 250

Execution Time (Buffer = 50): 00:00:00.0039429
Memory usage (Buffer = 50): 308328

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 18. Result time and space when testing program with buffer = 50

The algorithm took more time to complete the execution since with the small & finite amount of memory space, the operations needs to wait for each other to complete to process.

Time complexity:

Is the amount of time calculated or referred in instances when users may be interested to know in advance whether the program will provide a satisfactory real time response or not. Users wanted the algorithm to take less execution time. when the program execute an input string consists 250 characters, the Transfer Function will execute 2 times (Buffer = 125) to complete transfer the input string to destination string.



```
→ SendMessage dotnet run
Enter your message (Max: 250 character): The Highlands coffee has made for itself as the fastest growing
 coffee shop around. If you’ve seen once on your local corner, look across the street and you will see a
nother one. Because they’ve grown so quickly, they’re scrambling to update their or

Your message: The Highlands coffee has made for itself as the fastest growing coffee shop around. If you
’ve seen once on your local corner, look across the street and you will see another one. Because they’ve
 grown so quickly, they’re scrambling to update their or
Number of character: 250

Execution Time (Buffer = 125): 00:00:00.0024591
Memory usage (Buffer = 125): 332056

----< Thank you to using GreMes >----
----<   Press any key to exit   >----
```

Picture 19. Result time and space when testing with buffer = 125

The algorithm took more memory space in order to complete the execution processes as soon as possible.

Summary:

	Space prioritize (Buffer = 50)	**Time prioritize (Buffer = 125)**
Execution Time (sec)	0.0037468	0.0024591
Memory space used (bytes)	308328	332056

Table 2. Summary Space and Time for Time and Space prioritize

Base on people's priority, that program will be implemented **short time with big memory** or a **long time with low memory**. So, **Trade-off** is a people's selection about Time and Space which acceptable and suitable for the problems, people's requirements, hardware or software.

2.4. Three benefits of using implement of independent data structures

Data Structure can be defined as the collection of data objects which provides a way of storing and managing data in the computer so that it can be used. Various Data Structures types are arrays, Linked List, Stack, Queue.

Data Structures are widely used in almost every aspect of Computer Science for simple as well as complex computations. Data structures are used in all areas of computer science such as Artificial Intelligence, graphics, Operating system.
Data Structures are the key part of many computer algorithms as they allow the programmers to do data management in an efficient way. A right selection of data structure can enhance the efficiency of computer program or algorithm in a better way. (purplechaiblogger, n.d.)

There are three main benefits of using implementation of independent data structures:

- **It is secure way of storage, processing of data or access data anytime and anywhere:**

Independent data structure allow computer system(program) store a data that make information will be securely. The information is then available for later use and can be used by multiple programs. Additionally, the information is secures and can not be lost (especially if it is stored on magnetic tapes).

Beside of storing data, independent data structure have many functions to process with data such as Dequeue, Pop, IsFull, IsEmpty, etc. Which allow user to use and process data on a software system (Example: User want to add new value to the Stack, than user use Push() Operation without handling entire data bundle).

By storing data is convenient and allows system access data at any time or anywhere (Example: Using internet, user can access the data anytime from any connected machine such as computer, laptop, tablet, phone, etc).

- **These are necessary for designing an efficient algorithm and Graphs models real life problems:**

In real life problem, telecommunication network service provider have a problem in count a number messages send with specific characters to calculate cost for user. So, the Send Message program that I have implemented before, by using 2 independent data structures: Queue and Stack make the algorithms totally clean and efficiency. The Transfer Function has been designed based on 2 independent data structures.

- **Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services:**

Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures can be used to organize the storage and retrieval of information stored in both main memory and secondary memory.

CONCLUSION

Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost)

Afther this subject, I have a lot of knowledge to know what is ADT and using ADT to implement a real program to solve a real problem in life. In some case, I also have different prioritize about time or space to implement the system work well. When measuring the program, I can make sure that system will work with right requirement.

Bibliography

geeksforgeeks, n.d. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/
purplechaiblogger, n.d. *purplechaiblogger.* [Online]
Available at: http://www.purplechaiblogger.com/data-structure-importance-and-advantages/
tutorialspoint, n.d. *tutorialspoint.* [Online]
Available at: https://www.tutorialspoint.com/data_structures_algorithms/asymptotic_analysis.htm
wikipedia, n.d. *wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Asymptotic_analysis
wikipedia, n.d. *wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Big_O_notation