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	Nguyen Cong Nhat	**Student ID**	GCD17202
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	Nhat

Grading grid

P4	P5	P6	P7	M4	M5	D3	D4

❒ Summative Feedback:	❒ Resubmission Feedback:

Grade:	Assessor Signature:	Date:

Internal Verifier's Comments:
Signature & Date:

Table of Contents

I. INTRODUCTION:

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.

I 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.

I need 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.

II. LO3 - IMPLEMENT COMPLEX DATA STRUCTURES AND ALGORITHMS:

1. P4 Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem:

1.1. Situation: Write a program that can send a text message (maximum 250 characters) from source to destination.

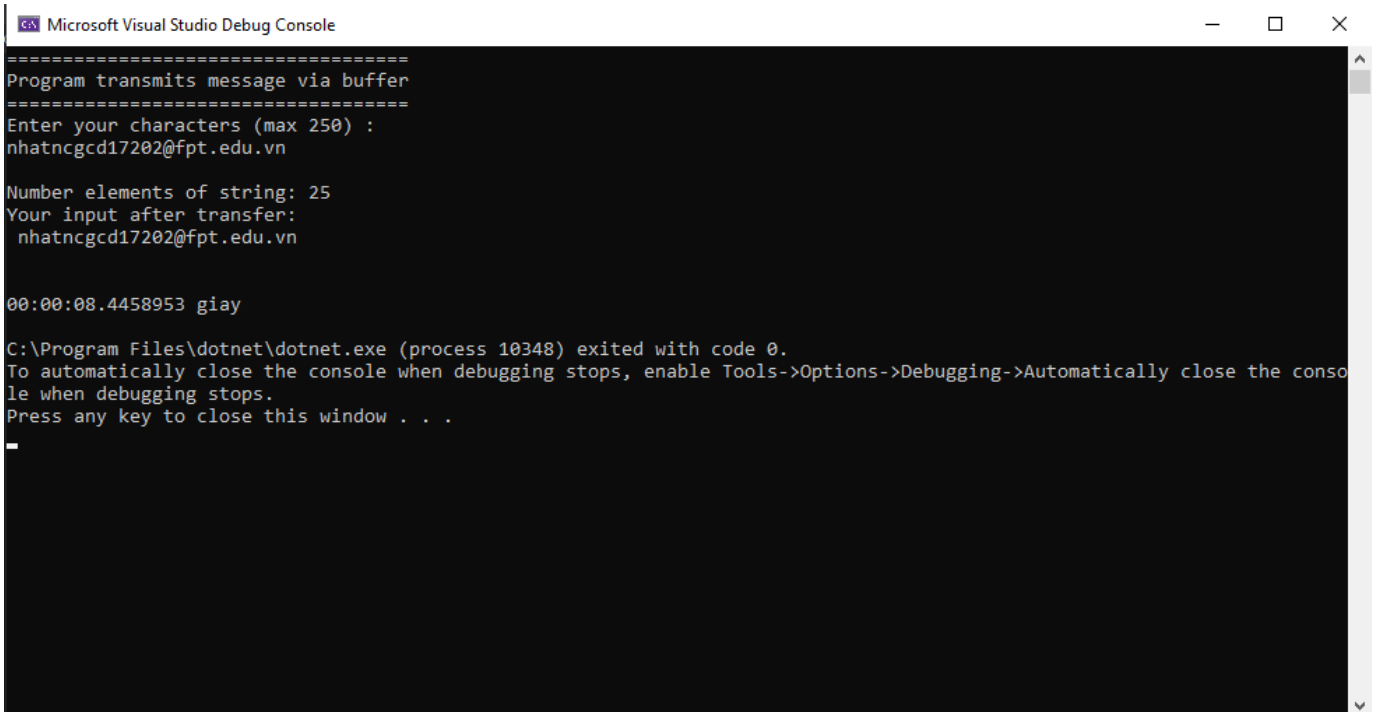
❖ I have to build a program that allows transmitting a message (up to 250 characters) from the first 2 points and the destination. I have to use buffers (including stack & queue) as intermediaries to move that message. My idea is:

- The program consists of 2 series s1 and s2.
- 2 buffers include: Stack & Queue.
- Initially the string s1 will transmit the message to the Stack, then the Stack will transmit the message to the Queue and finally the Queue will transmit the message to the string s2 and print out s2.

1.2. My program:

```
using System;
using System.Diagnostics;
namespace ASM2_DataStructure
{
    class Queue
    {
        public int max;
        public char[] Q;
        public int r;
        public int f;
        public Queue(int max, char[] Q)
        {
            r = 0;
            f = 0;
            this.max = max;
            this.Q = Q;
        }
        public int Enum()
        {
            if (r == f)
            {
                return 0;
            }
            else
                return (((max - f) + r) % max);
        }
        public bool Isempty()
        {
            if (f == r)
            {
                Console.WriteLine("Queue is empty");
                return true;
            }
            else
                return false;
        }
        public void Enqueue(char x)
        {
            Q[r] = x;
            r = (r + 1) % max;
        }
        public char Dequeue()
        {
            char De =' ';
            De = Q[f];
                f = (f + 1) % max;
                return De;
        }
    }
    class Stack
    {
        public int MAX ;
        public int top;
        public char[] stack;
        public Stack()
        {
            top = -1;
            MAX = 250;
            stack = new char[MAX];
        }
        public void Push(char data)
        {
            if (top >= MAX)
            {
                Console.WriteLine("Stack Overflow");
            }
            else
            {
                top = top + 1;
                stack[top] = data;
            }
        }

        public char Pop()
        {
            char value = ' ';
            if (top < 0)
            {
                Console.WriteLine("Stack Underflow");
                return value;
            }
            else
            {
                value = stack[top--];
                return value;
            }
        }
    }
    class Send
    {
        public void SendMess(char[] input)
        {
            var queue = new char[250];
            var s2 = new char[250];
            Stack mystack = new Stack();
            Queue myqueue = new Queue(250, queue);

            Console.WriteLine("\nNumber elements of string: {0}", input.Length);

                try
                {
                    for (int i = 0; i < input.Length; i++)
                    {
                        mystack.Push(input[i]);
                    }
                    for (int i = 0; i < input.Length; i++)
                    {
                        myqueue.Enqueue(mystack.Pop());
                    }
                    for (int i = input.Length; i > 0; i--)
                    {
                        s2[i] = myqueue.Dequeue();
                    }

                    Console.WriteLine("Your input after transfer: ");
                    Console.Write(s2);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }


        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Send mysend = new Send();
            Stopwatch st = new Stopwatch();
            Console.WriteLine("====================================");
            Console.WriteLine("Program transmits message via buffer");
            Console.WriteLine("====================================");
            Console.WriteLine("Enter your characters (max 250) : ");
            st.Start();
            string input = Console.ReadLine();
            if (input.Length >= 250)
            {
                string new_input = input.Substring(0, 250);
                char[] s1 = new_input.ToCharArray();
                mysend.SendMess(s1);
                Console.WriteLine("\nWe just send message with 250 characters.");

            }
            else if (input.Length < 1)
            {
                Console.WriteLine("Error! Array null...");
            }
            else
            {
                char[] s1 = input.ToCharArray();
                mysend.SendMess(s1);
            }
            st.Stop();
            Console.WriteLine("                                ");
            Console.WriteLine("\n{0} giay", st.Elapsed.ToString());
            Console.ReadKey();
        }
    }
}
```

1.3. Result:

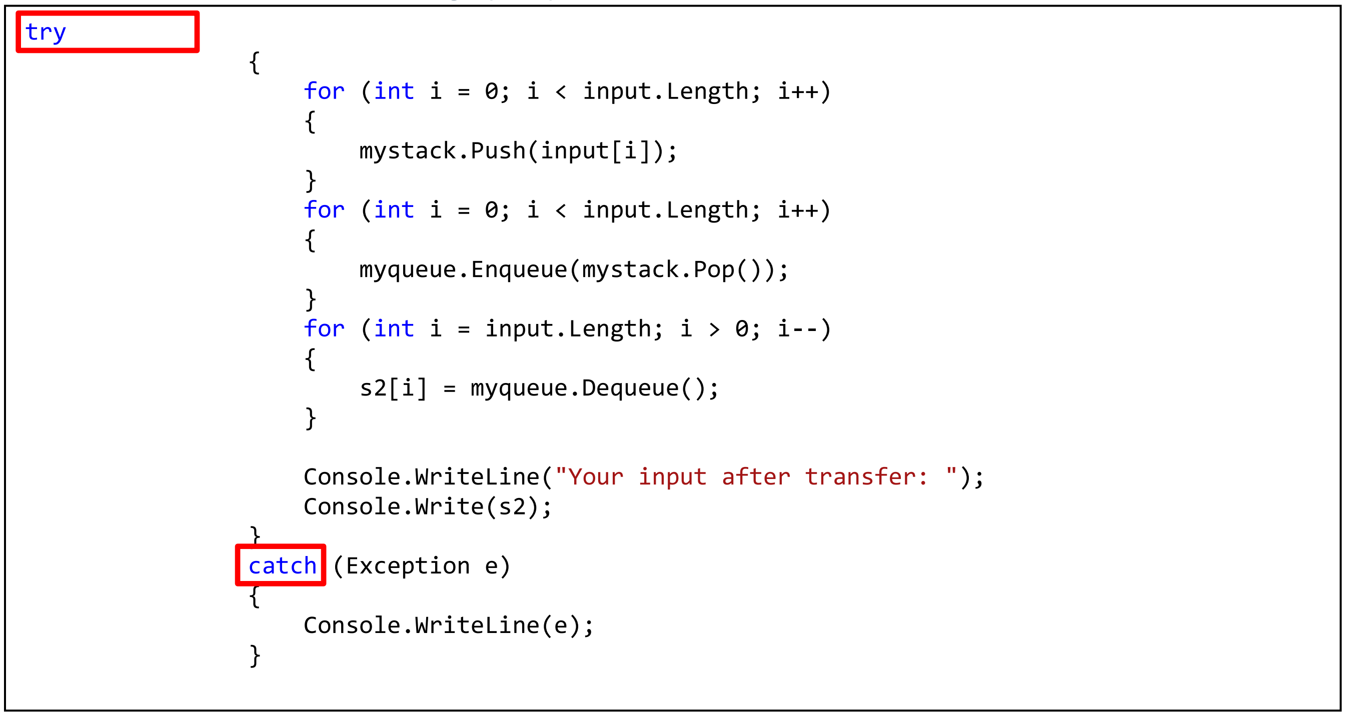


```
===================================
Program transmits message via buffer
===================================
Enter your characters (max 250) :
nhatncgcd17202@fpt.edu.vn

Number elements of string: 25
Your input after transfer:
 nhatncgcd17202@fpt.edu.vn


00:00:08.4458953 giay

C:\Program Files\dotnet\dotnet.exe (process 10348) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

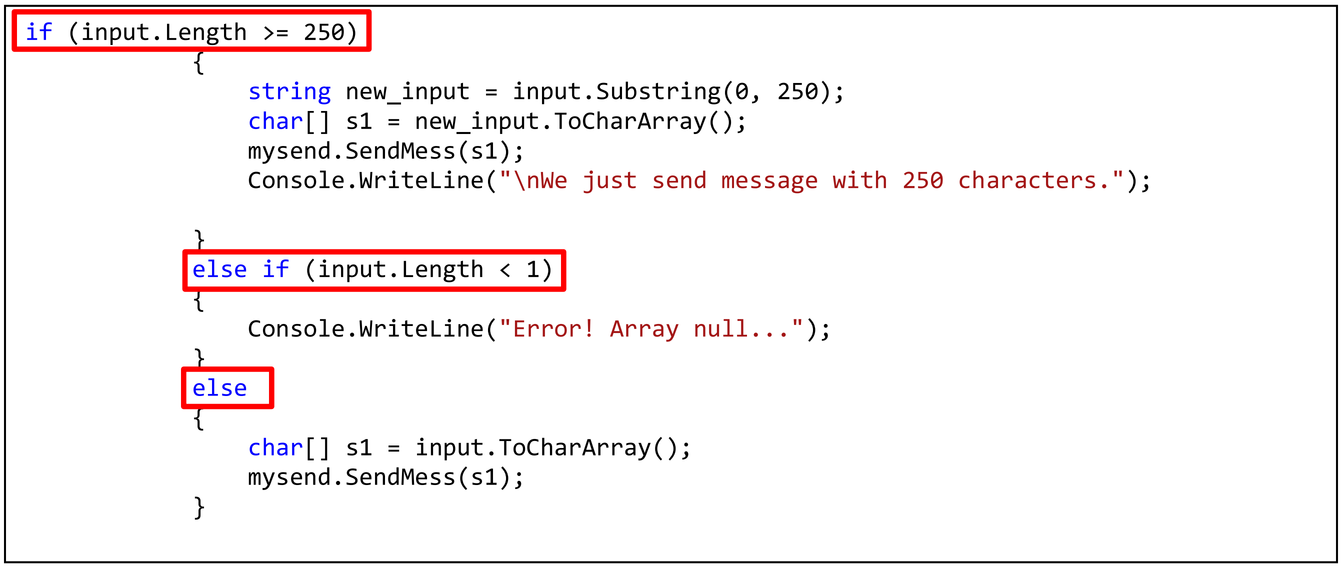
- I sent the string ***nhatncgcd17202@fpt.edu.vn*** and the medium it remains unchanged and outputs the results as expected.

2. P5 Implement error handling and report test results:

2.1. Error handling by Try Catch:



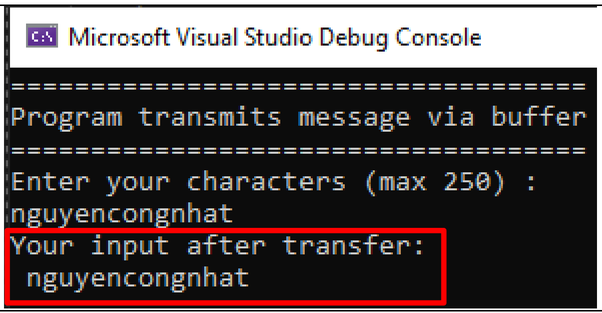
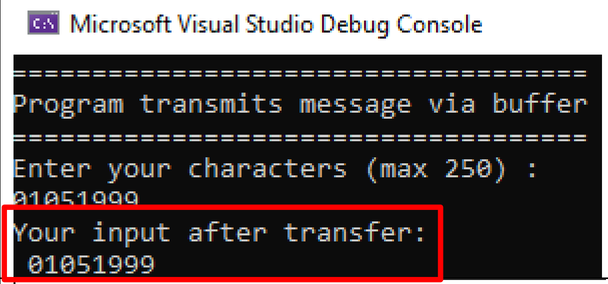
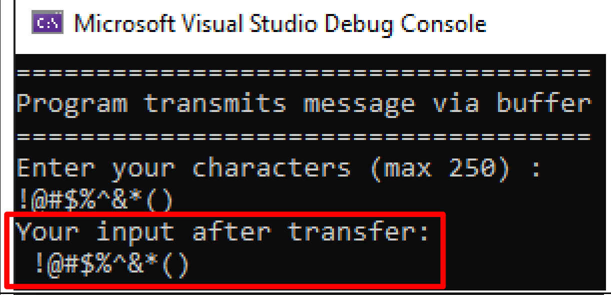
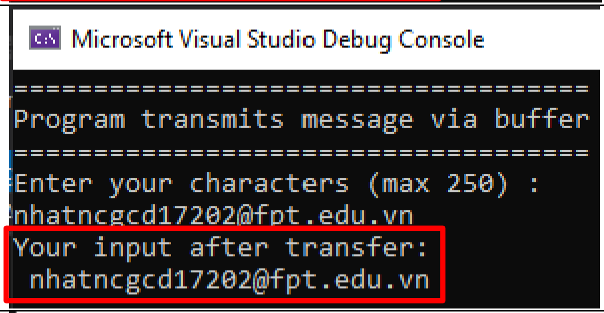
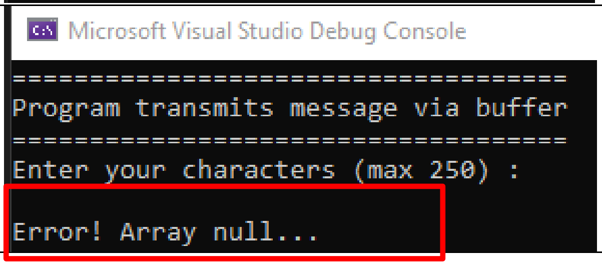
```
try
            {
                for (int i = 0; i < input.Length; i++)
                {
                    mystack.Push(input[i]);
                }
                for (int i = 0; i < input.Length; i++)
                {
                    myqueue.Enqueue(mystack.Pop());
                }
                for (int i = input.Length; i > 0; i--)
                {
                    s2[i] = myqueue.Dequeue();
                }

                Console.WriteLine("Your input after transfer: ");
                Console.Write(s2);
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
```

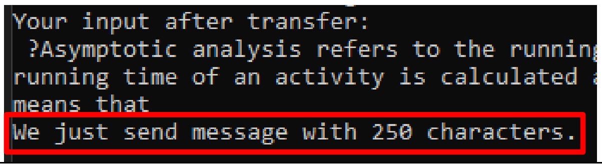


```
if (input.Length >= 250)
            {
                string new_input = input.Substring(0, 250);
                char[] s1 = new_input.ToCharArray();
                mysend.SendMess(s1);
                Console.WriteLine("\nWe just send message with 250 characters.");

            }
            else if (input.Length < 1)
            {
                Console.WriteLine("Error! Array null...");
            }
            else
            {
                char[] s1 = input.ToCharArray();
                mysend.SendMess(s1);
            }
```

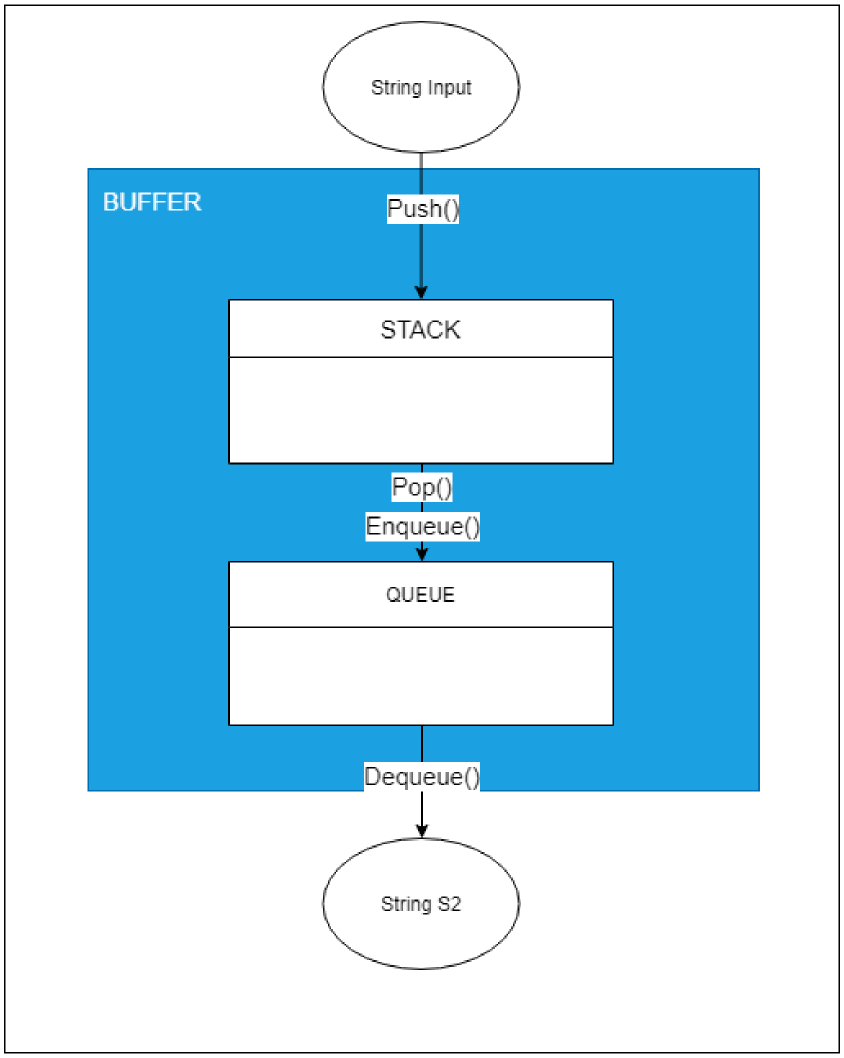
❖ Based on my ideas outlined in section 1.1, I use the If-else & Try-Catch command to catch errors for the program. The catch of this error to handle the input is wrong, oversizie, empty or check for algorithms of ADT Stack & Queue.

2.2. Test Case:

Serial	DESC	Input	Expected	Result	Status
1	Enter string only text type	*"nguyencongnhat"*	Sucessfully	 `Microsoft Visual Studio Debug Console` `====================================` `Program transmits message via buffer` `====================================` `Enter your characters (max 250) :` `nguyencongnhat` `Your input after transfer:` `nguyencongnhat`	PASS
2	Enter string only int type	*01051999*	Sucessfully	 `Microsoft Visual Studio Debug Console` `====================================` `Program transmits message via buffer` `====================================` `Enter your characters (max 250) :` `01051999` `Your input after transfer:` `01051999`	PASS
3	Enter string only special characters	*!@#$%^&*()*	Sucessfully	 `Microsoft Visual Studio Debug Console` `====================================` `Program transmits message via buffer` `====================================` `Enter your characters (max 250) :` `!@#$%^&*()` `Your input after transfer:` `!@#$%^&*()`	PASS
4	Enter the string with int, text type and special characters	*Nhatncgcd17202@fpt.edu.vn*	Sucessfully	 `Microsoft Visual Studio Debug Console` `====================================` `Program transmits message via buffer` `====================================` `Enter your characters (max 250) :` `nhatncgcd17202@fpt.edu.vn` `Your input after transfer:` `nhatncgcd17202@fpt.edu.vn`	PASS
5	String Null	""	Can't enter null	 `Microsoft Visual Studio Debug Console` `====================================` `Program transmits message via buffer` `====================================` `Enter your characters (max 250) :` `Error! Array null...`	PASS

6	Enter a string of more than 250 characters.	*String more than 250 characters*	The program just send 250 characters	 ```Your input after transfer: ?Asymptotic analysis refers to the runnin running time of an activity is calculated means that We just send message with 250 characters.```	PASS

3. **M4 Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem:**



❖ As shown in the image above, we can see how the data is transferred through the buffer. My software uses 4 functions **Pop(), Push(), Enqueue()** and finally **Dequeue()** to transfer data to **S2** and print it to the screen.

3.1. Stack:

```
class Stack
    {
        public int MAX ;
        public int top;
        public char[] stack;
        public Stack()
        {
            top = -1;
            MAX = 250;
            stack = new char[MAX];
        }
        public void Push(char data)
        {
            if (top >= MAX)
            {
                Console.WriteLine("Stack Overflow");
            }
            else
            {
                top = top + 1;
                stack[top] = data;
            }
        }

        public char Pop()
        {
            char value = ' ';
            if (top < 0)
            {
                Console.WriteLine("Stack Underflow");
                return value;
            }
            else
            {
                value = stack[top--];
                return value;
            }
        }
    }
```

❖ In ADT **Stack**, when using it as a buffer, I created in it 2 functions **Push**() and **Pop**(). Function **push**() to get the word of the Input string into the Stack. Then save and use the function **pop()** to pass into the Queue:
  - **Push**(): when you want to transfer data to a Stack, we need to increase the top variable to 1 unit to transfer data to the top value.
  - **Pop():** when you want to take a part from the Stack, we need to reduce the top by one unit and return the value of the section from top - **stack(top--).**

3.2. Queue:

```
class Queue
    {
        public int max;
        public char[] Q;
        public int r;
        public int f;
        public Queue(int max, char[] Q)
        {
            r = 0;
            f = 0;
            this.max = max;
            this.Q = Q;
        }
        public int Enum()
        {
            if (r == f)
            {
                return 0;
            }
            else
                return (((max - f) + r) % max);
        }
        public bool Isempty()
        {
            if (f == r)
            {
                Console.WriteLine("Queue is empty");
                return true;
            }
            else
                return false;
        }
        public void Enqueue(char x)
        {
            Q[r] = x;
            r = (r + 1) % max;
        }
        public char Dequeue()
        {
            char De =' ';
            De = Q[f];
                f = (f + 1) % max;
                return De;
        }
    }
```

- In ADT **Queue**, when using it as a buffer I created in it 2 functions **Enqueue() & Dequeue().**
    - **Enqueue():** I use it to enter values from the **pop()** function of the Stack. **Pop()** takes data from Stack and enqueue () takes that value and passes it into Queue.
    - **Dequeue():** I use it to export values from **Queue** to string **S2** to print to the screen.

3.3. How buffer works SendMess Function:

```
public void SendMess(char[] input)
   {
       var queue = new char[250];
       var s2 = new char[250];
       Stack mystack = new Stack();
       Queue myqueue = new Queue(250, queue);
       Console.WriteLine("\nNumber elements of string: {0}", input.Length);
           try
           {
               for (int i = 0; i < input.Length; i++)
               {
                   mystack.Push(input[i]);
               }
               for (int i = 0; i < input.Length; i++)
               {
                   myqueue.Enqueue(mystack.Pop());
               }
               for (int i = input.Length; i > 0; i--)
               {
                   s2[i] = myqueue.Dequeue();
               }
               Console.WriteLine("Your input after transfer: ");
               Console.Write(s2);
           }
           catch (Exception e)
           {
               Console.WriteLine(e);
           }
          }
```

❖ As I explained ADTs in section 3.2, my buffer consists of **ADT Stack** and **ADT Queue** which will be the intermediary to transport characters from string **Input to S2**. The process of conversion in the order of **Input → Stack → Queue → S2.**

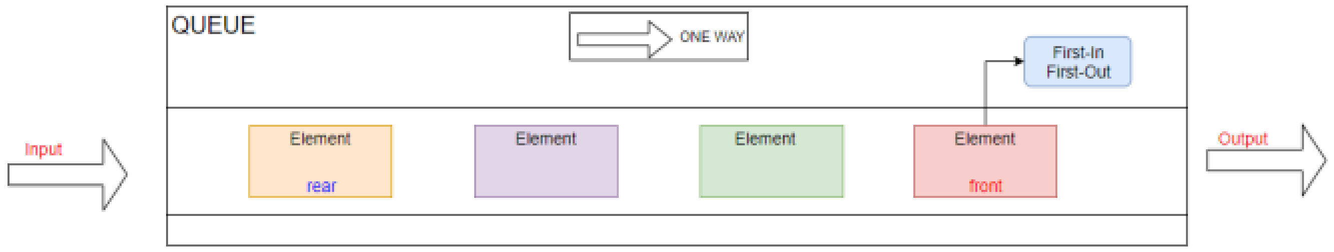
4. D3 Critically evaluate the complexity of an implemented ADT/algorithm:

Normally, when we select a preferred algorithm to use for a program, we need to compare the complexity of those algorithms to find the most suitable algorithm. In this report, I have used ADT Queue many times, so we will go into more depth about the complexity of an ADT Queue algorithm. (Anon., n.d.)

4.1. The structure of a Queue:

In a Queue, there is a string that stores both the first and last elements and the front and rear respectively. Elements are stored, imported / exported into the queue in a certain direction from rear-front. This means that elements that have been imported into the Queue first will be output from the Queue earlier.

4.2. How the Queue works:



In the queue structure, we can only add elements to one end of the queue (rear), and can only delete elements at the other end of the queue (front). Thus, at one end it is not possible to have two additional actions and simultaneous deletion. So, with the Queue, we have the following functions:

- ❖ EnQueue: Add element at the end of the Queue.
- ❖ DeQueue: Removes the element from the beginning (front) of Queue. If Queue is empty, the error message.
- ❖ IsEmpty: Check for an empty Queue.
- ❖ Front: Gets the value of the element at the top (front) of the Queue. Get the value without changing Queue.

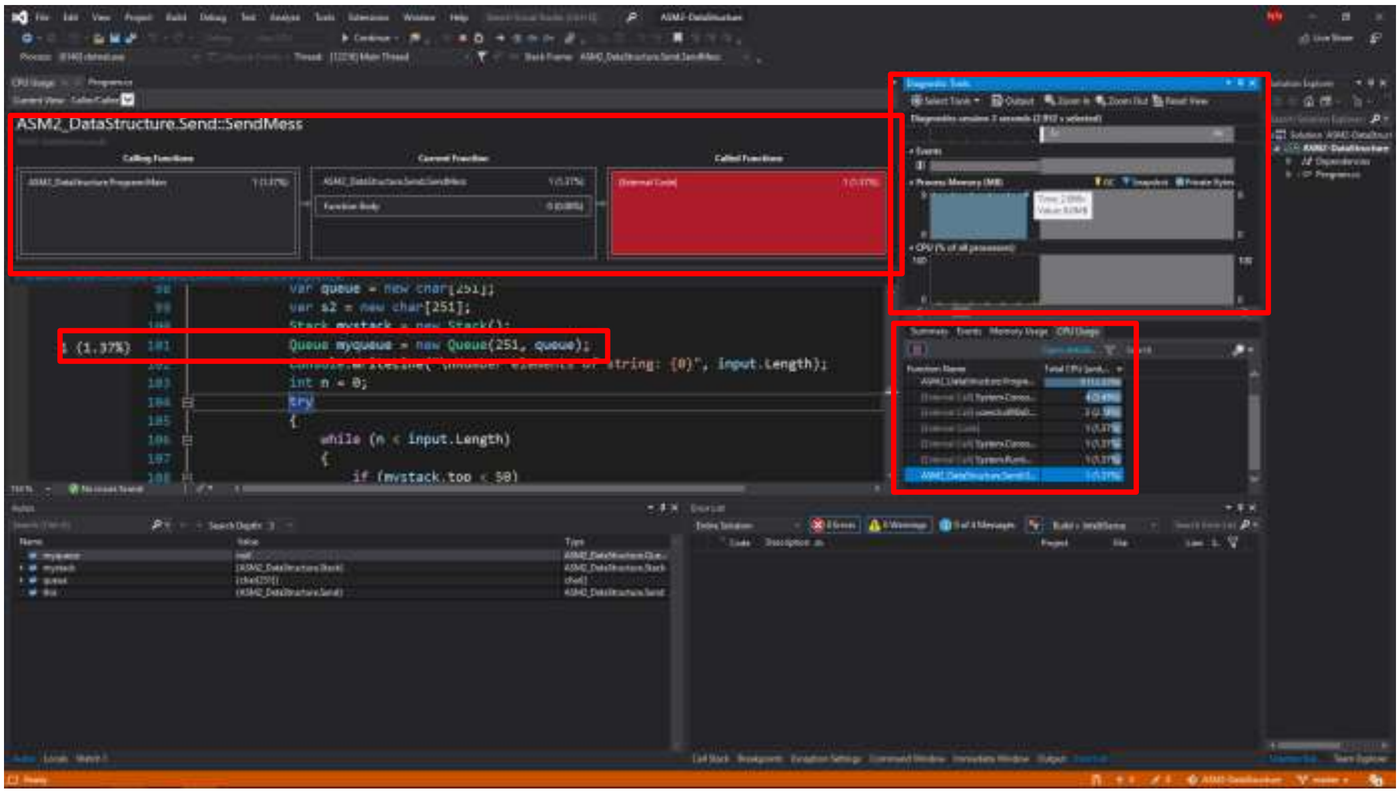
4.3. The complexity of ADT Queue:

- ❖ Below is code and I calculated the complexity according to Big O notation:

```
public class Queue
    {
        // O(1)
        public int max;
        public string[] Q;
        public int r = 0;
        public int f = 0;

        public Queue(int max, string[] Q)
        {
            this.max = max;
            this.Q = Q;
        }

        public int Enum()
        {
            // O(1)
            if (r == f)
            {
                return 0;
            }
            else
                return (((max - f) + r) % max);
        }
        public bool Isempty()
        {
            // O(1)
            if (f == r)
            {
                Console.WriteLine("Queue is empty");
                return true;
            }
            else
                return false;
        }
        public bool Isfull()
        {
            // O(1)
            if ((((max - f) + r) % max) == (max - 1))
            {

                Console.WriteLine("Queue is full");
                return true;
            }
            else
                return false;
        }
        public string Peek()
        {
            return Q[f];
        }
        public void Enqueue(string x)
        {
            // O(1)
            if (Enum() == max)
            {
                Console.WriteLine("Queue is full");
            }
            else
            {
                Q[r] = x;
                r = (r + 1) % max;
            }

        }
        public string Dequeue()
        {
            // De - O(1)
            string De;
            De = Q[f];
            f = (f + 1) % max;
            return De;
        }
    }
```

- With ADT Queue, we have the complexity calculated by the Big O notation: O(1) + O(1) + O(1) + O(1) = O(1). **Suppose, Queue initializes the sequence Char[50] with O(1) and the time value is 5s, then it is constant and it cannot be changed during the implementation of ADT.**

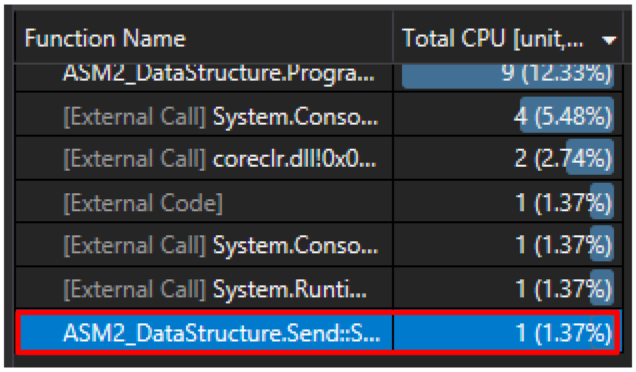
❖ Calculating complexity in Visual Studio 2019:



ASM2_DataStructure.Send::SendMess

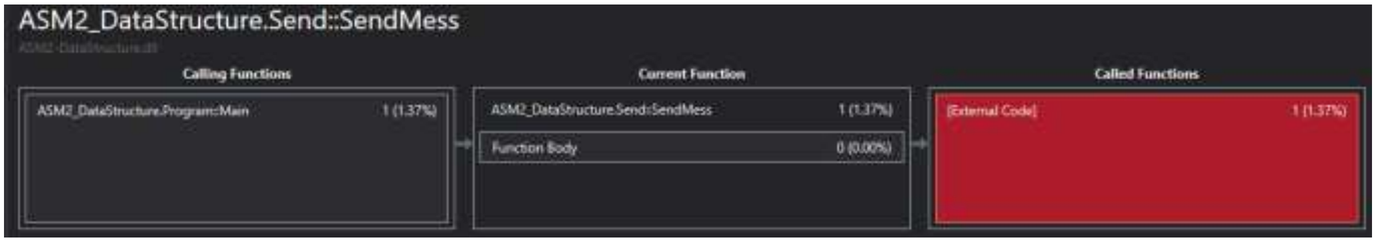
```
var queue = new char[251];
var s2 = new char[251];
Stack mystack = new Stack();
Queue myqueue = new Queue(251, queue);
Console.WriteLine("Number elements of string: {0}", input.Length);
int n = 0;
try
{
    while (n < input.Length)
    {
        if (mystack.top < 50)
```

- Overview of measuring time complexity & Space Complexity on visual studio 2019. We have parts: CPU usage, RAM usage. We will then go into details on when the Queue function is called.



Function Name	Total CPU [unit,...
ASM2_DataStructure.Progra...	9 (12.33%)
[External Call] System.Conso...	4 (5.48%)
[External Call] coreclr.dll!0x0...	2 (2.74%)
[External Code]	1 (1.37%)
[External Call] System.Conso...	1 (1.37%)
[External Call] System.Runti...	1 (1.37%)
ASM2_DataStructure.Send::S...	1 (1.37%)

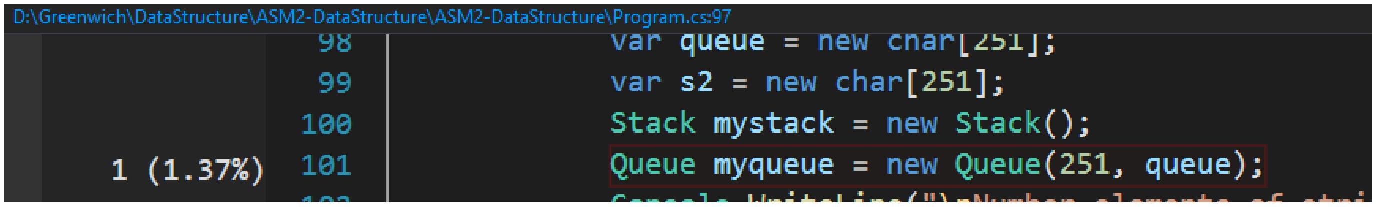
- When Queue is called, it only takes **1CPU** of program.


ASM2_DataStructure.Send::SendMess

ASM2-DataStructure.dll

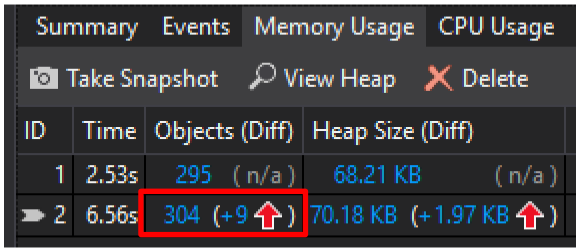
Calling Functions		Current Function		Called Functions	
ASM2_DataStructure.Program::Main	1 (1.37%)	ASM2_DataStructure.Send::SendMess	1 (1.37%)	[External Code]	1 (1.37%)
		Function Body	0 (0.00%)		

- The order of call functions: **Main → SendMess → Queue.**


D:\Greenwich\DataStructure\ASM2-DataStructure\ASM2-DataStructure\Program.cs:97

```
                    var queue = new char[251];
                    var s2 = new char[251];
                    Stack mystack = new Stack();
1 (1.37%)           Queue myqueue = new Queue(251, queue);
```

- The place where Queue is called and it accounts for **1.37% of the CPU**.



ID	Time	Objects (Diff)	Heap Size (Diff)
1	2.53s	295 (n/a)	68.21 KB (n/a)
2	6.56s	304 (+9)	70.18 KB (+1.97 KB)

- I debug the program and break at the place where Queue is called, when the program takes 8.0MB RAM usage. In particular, the RAM used to call Queue out only accounts for **9kb.**
- The program stops at the breakpoint called Queue, the time lost for this process is 6.56 and the number is 304kb. These numbers will be constant during ADT Queue usage because of their complexity O(1).

❖ **Conclusion:** In essence, **ADT Queue** is just an abstract storage problem, so Queue is not as complicated as other algorithms. It is very simple, only takes **1 CPU, time complexity & space complexity (O(1)) is very low**.

III. LO4 – ASSESS THE EFFECTIVENESS OF DATA STRUCTURES AND ALGORITHMS:

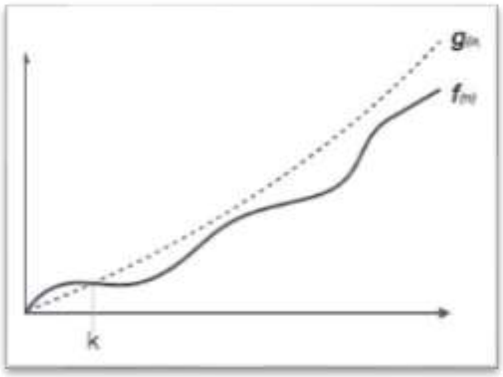
1. P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm: (Anon., 2016)

1.1. Definition of Asymptotic analysis:

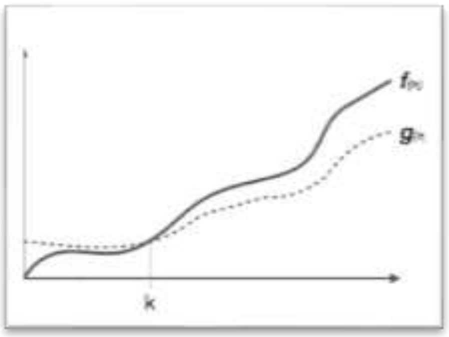
- ❖ Asymptotic analysis of an algorithm helps us estimate the running time of an algorithm (algorithm). We can rely on it to draw the best conclusions about possible situations for an algorithm. In other words, asymptotic analysis refers to the runtime estimation of calculations in steps performed by an algorithm.
- ❖ The time to perform an algorithm is divided into 3 cases:
  - o The best case: is the minimum time needed to execute the program.
  - o Average case: the average time needed to execute the program.
  - o Worst case: the maximum amount of time needed to execute a program.

1.2. Asymptotic Notation & how it can be use to assess the effectiveness of an algorithm:

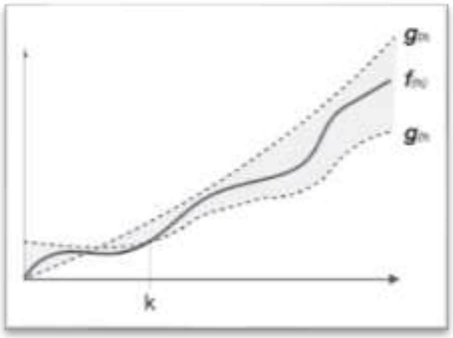
- ❖ Asymptotic analysis is bound to input, that is, if there is no input to the algorithm, it is concluded to be active for a constant time. Other than "input", all other factors are considered constants.
- ❖ Asymptotic analysis refers to the running time calculation of any activity in the calculation units. For example, the running time of an activity is calculated as **f(n)** and it is possible for another activity, it is counted as **g(n^2)**. This means that the run time of the first activity will increase linearly with the increase in **n**, and the running time of the second activity will increase exponentially as **n** increases. Similarly, the running time of both operations will be almost the same if **n** is significantly small.
- ❖ Asymptotic Notations in estimating run-time complexity of an algorithm:
  - o **Big Oh Notation: O(n)** is a way to express the upper asymptote of the algorithm's runtime. It estimates the worst case complexity or the longest period the algorithm needs.



  - o **Omega Notation: Ω(n)** is a way to represent a lower asymptotic representation of the running time of the algorithm. It estimates the best case time complexity or the shortest amount of time required for an algorithm.



- **Theta Notation: θ(n)** is a way to represent both the upper and lower asymptotes of the algorithm's runtime.

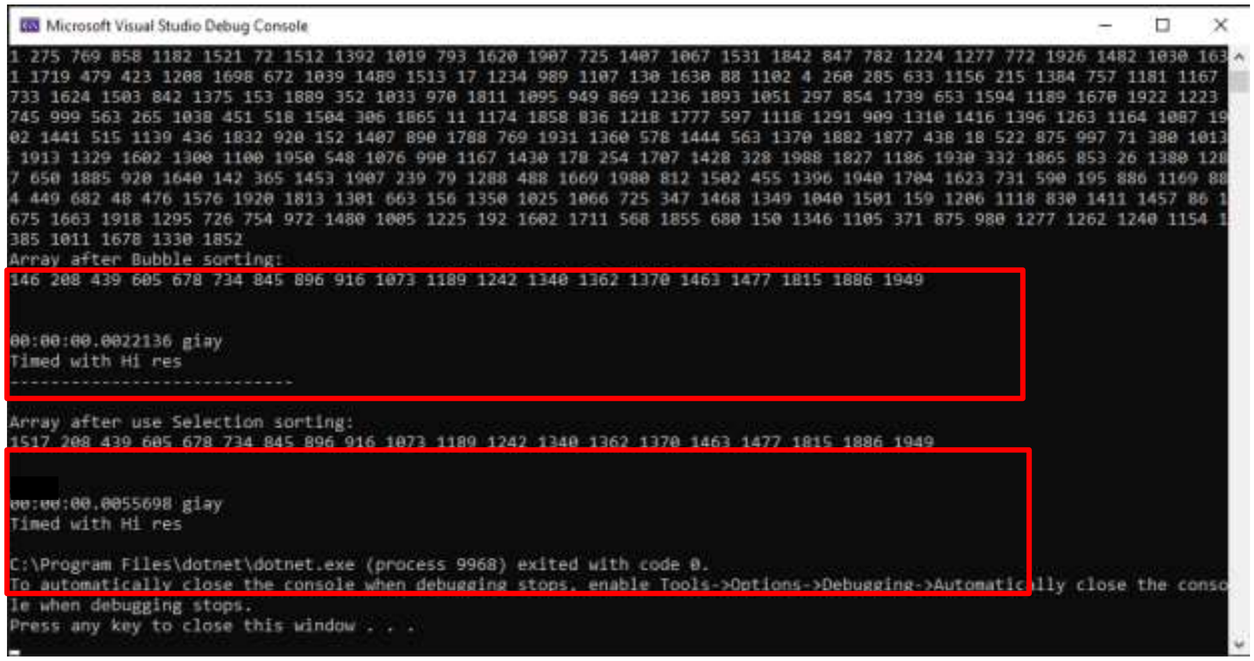


2. **P7 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example:** (Anon., 2017)

❖ Algorithm analysis is the determination of an algorithm's complexity through characteristics: **The amount of time it takes to perform the algorithm, the amount of storage resources to execute the algorithm (RAM & CPU).** This involves determining the complexity of the input to the algorithm and the steps for storing and processing that input. An algorithm is said to be effective when the values of time, RAM & CPU Usage are balanced and only change based on the size of the input. The complexity of Inputs also makes the algorithm different in efficiency, so when measuring the performance of an algorithm to evaluate the effectiveness, it must pay attention to three cases: Best case, best case and field average.

2.1. Time Complexity of an Algorithm:

❖ Is execution time important to evaluate an efficient algorithm? The time depends on whether the programmer has optimized the algorithm well or not, the amount of data input also affects the execution time of an algorithm. Or are the calculations complex? Or it depends on the speed of the processor, what other computer is running. To evaluate the execution time of an algorithm. In theory, they talk about BigO notation to show the execution time of an algorithm through different inputs, for example: For 100, 1000 elements, the time is expressed as **$O(n^{100})$** or **$O(n^{1000})$** but with millions and billions of elements we cannot perform the BigO notation. So, it came with a time function to perform a math problem, for example in C #: I used the reference `System.Diagnostics;` ; To measure the actual time the algorithm takes how long it takes to complete.

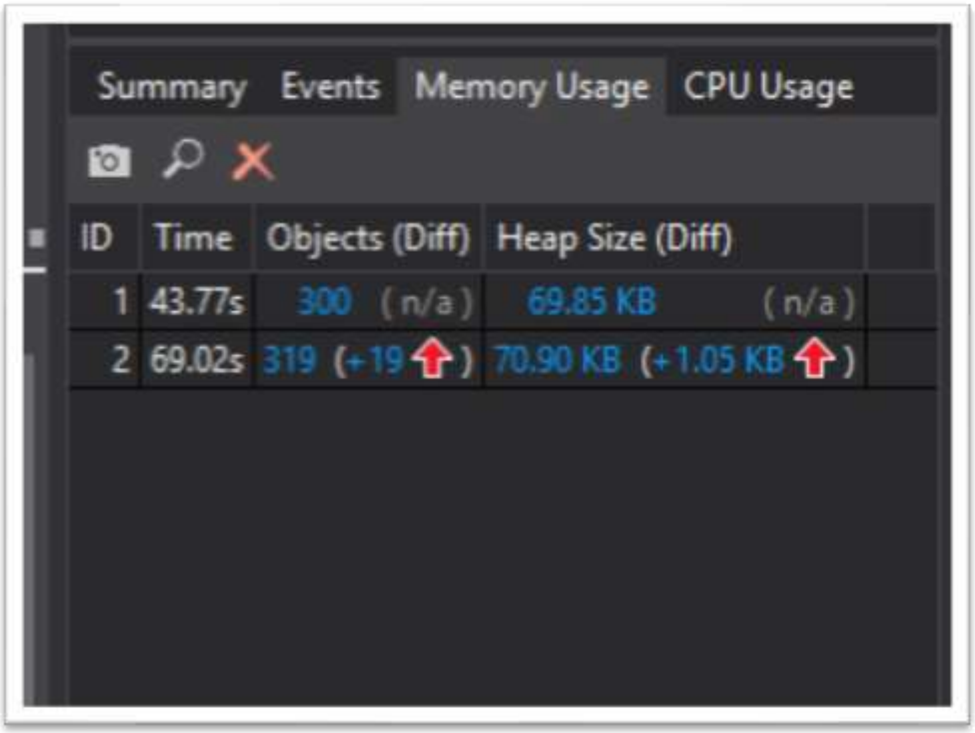


```
1 275 769 858 1182 1521 72 1512 1392 1019 793 1620 1907 725 1407 1067 1531 1842 847 782 1224 1277 772 1926 1482 1030 163
1 1719 479 423 1208 1698 672 1039 1489 1513 17 1234 989 1107 130 1630 88 1102 4 260 285 633 1156 215 1384 757 1181 1167
733 1624 1503 842 1375 153 1889 352 1033 970 1811 1095 949 869 1236 1893 1051 297 854 1739 653 1594 1189 1670 1922 1223
745 999 563 265 1038 451 518 1504 306 1865 11 1174 1858 836 1218 1777 597 1118 1291 909 1310 1416 1396 1263 1164 1087 19
02 1441 515 1139 436 1832 920 152 1407 890 1788 769 1931 1360 578 1444 563 1370 1882 1877 438 18 522 875 997 71 380 1013
 1913 1329 1602 1300 1100 1950 548 1076 990 1167 1430 178 254 1707 1428 328 1988 1827 1186 1930 332 1865 853 26 1380 128
7 650 1885 920 1640 142 365 1453 1907 239 79 1288 488 1669 1980 812 1502 455 1396 1940 1704 1623 731 590 195 886 1169 88
4 449 682 48 476 1576 1920 1813 1301 663 156 1350 1025 1066 725 347 1468 1349 1040 1501 159 1206 1118 830 1411 1457 86 1
675 1663 1918 1295 726 754 972 1480 1005 1225 192 1602 1711 568 1855 680 150 1346 1105 371 875 980 1277 1262 1240 1154 1
385 1011 1678 1330 1852
Array after Bubble sorting:
146 208 439 605 678 734 845 896 916 1073 1189 1242 1340 1362 1370 1463 1477 1815 1886 1949


00:00:00.0022136 giay
Timed with Hi res
----------------------------

Array after use Selection sorting:
1517 208 439 605 678 734 845 896 916 1073 1189 1242 1340 1362 1370 1463 1477 1815 1886 1949


00:00:00.0055698 giay
Timed with Hi res

C:\Program Files\dotnet\dotnet.exe (process 9968) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

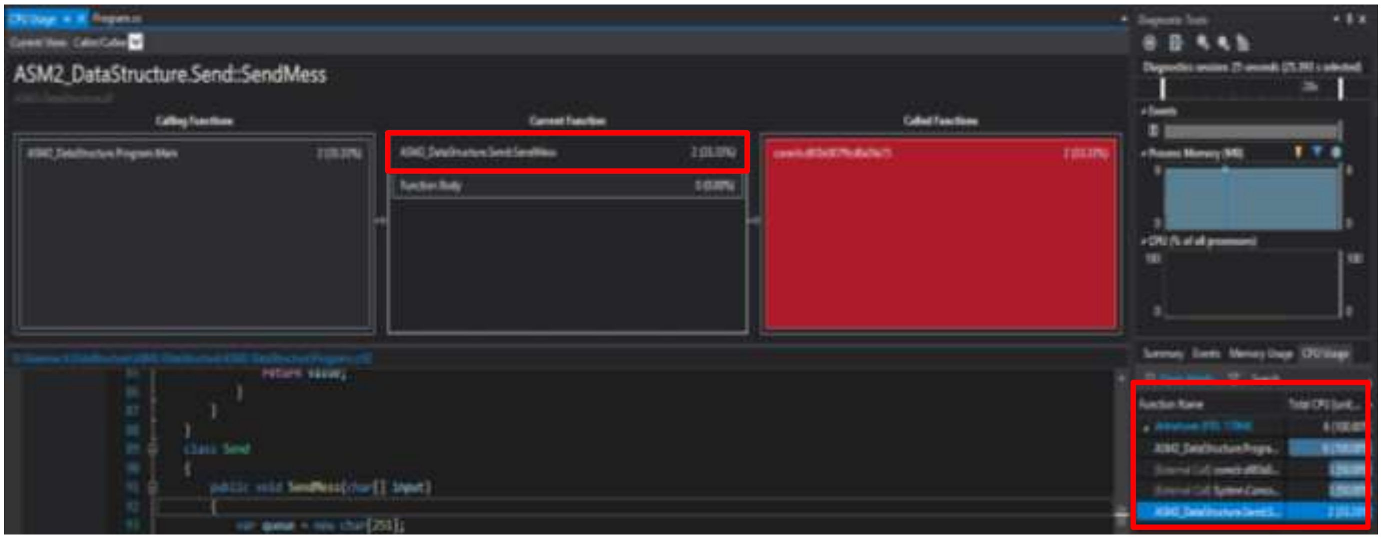
❖ For example: As shown in the image above, I have compared 2 sorting algorithms. The results show that, with different algorithms but processing the same input, the execution time of the algorithmswill be different. Therefore, we need to choose the type of algorithm that best suits the input to optimize the program.

2.2. Space Complexity (RAM usage & CPU usage):

- In a programming language, the spatial complexity of an algorithm is the amount of memory space that must be devoted to an algorithm that solves the problem of calculating an input. In other words, it is the memory required by an algorithm to execute a program and generate output.
- Similar to time complexity, spatial complexity is represented by Big O, such as: O(n), O(n^2), O(n^a),….
- But in reality, I use C # language on Visual Studio 2019 environment. In Visual Studio environment, it supports us to measure the program space needed to perform algorithms, measure and output. display certain values that are not the same as the BigO notation. For example: *I made the program send a message with 2 buffers, with 250 characters including special characters. The program initialized a string of 250 characters and moved it through the buffers. As the figure below: The program occupied 319kb for* ***Objects*** *and 70.90kb for* ***Heap***



ID	Time	Objects (Diff)	Heap Size (Diff)
1	43.77s	300 (n/a)	69.85 KB (n/a)
2	69.02s	319 (+19)	70.90 KB (+1.05 KB)


ASM2_DataStructure.Send::SendMess

```
                    return value;
                }
            }
        }
        class Send
        {
            public void SendMess(char[] input)
            {
                var queue = new char[251];
```

❖ **For example:** In a program that sends messages with 250 characters, the SendMess function has been called and accounts for 2/6 of the program's CPU usage.

3. **M5 Interpret what a trade-off is when specifying an ADT using an example to support your answer:**

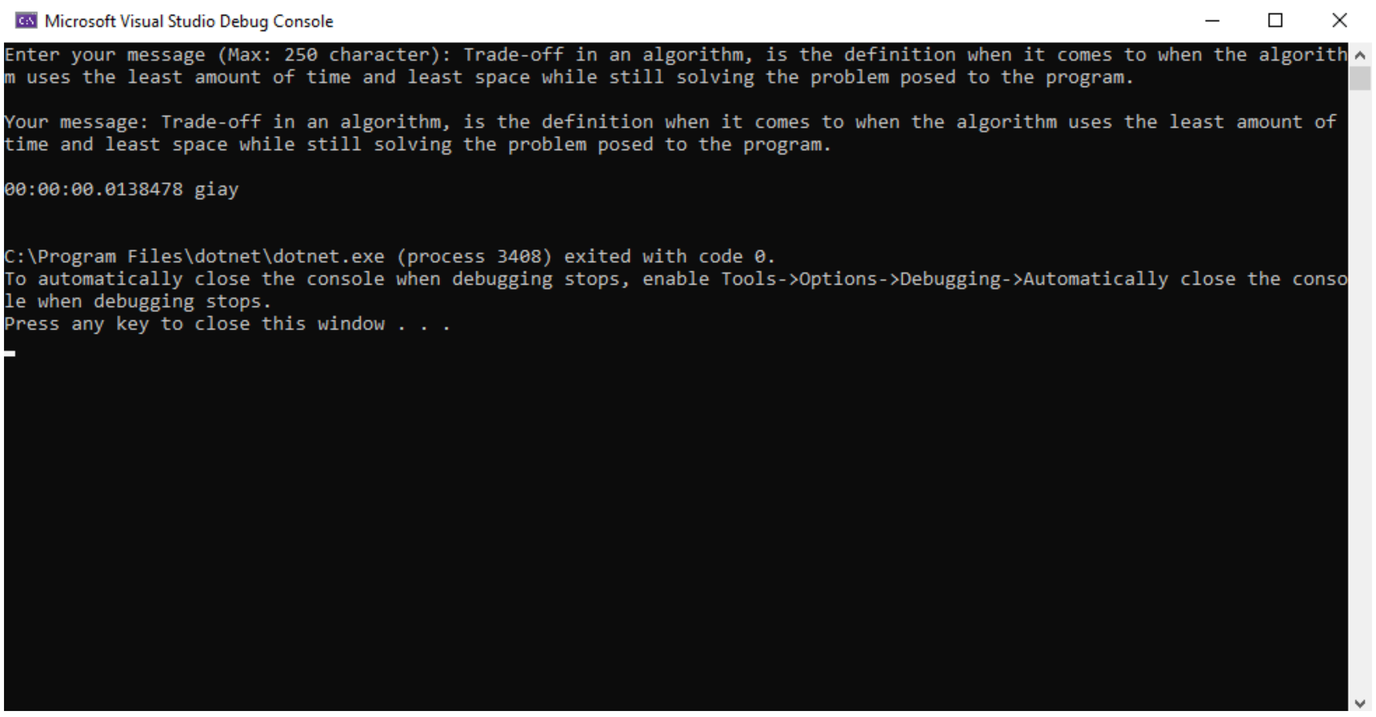
3.1. What is a trade-off:

❖ Trade-off in an algorithm, is the definition when it comes to when the algorithm uses the least amount of time and least space while still solving the problem posed to the program.

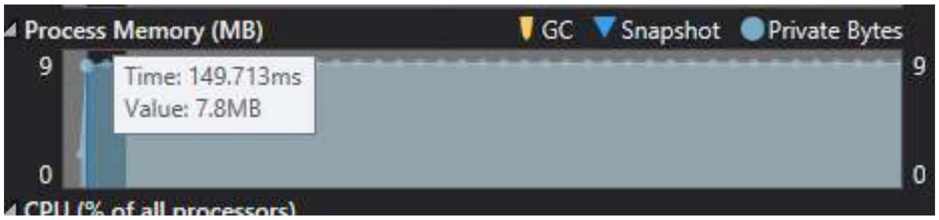
3.2. Explain trade-off:

❖ The best algorithm is a problem-solving algorithm that requires less space and time to execute its commands or to generate input. But in fact, time and space are closely related, we need to trade in one of the two characteristics to suit the problem-solving situation we want. It is a feature of trade-off, hitting space or time in an algorithm.

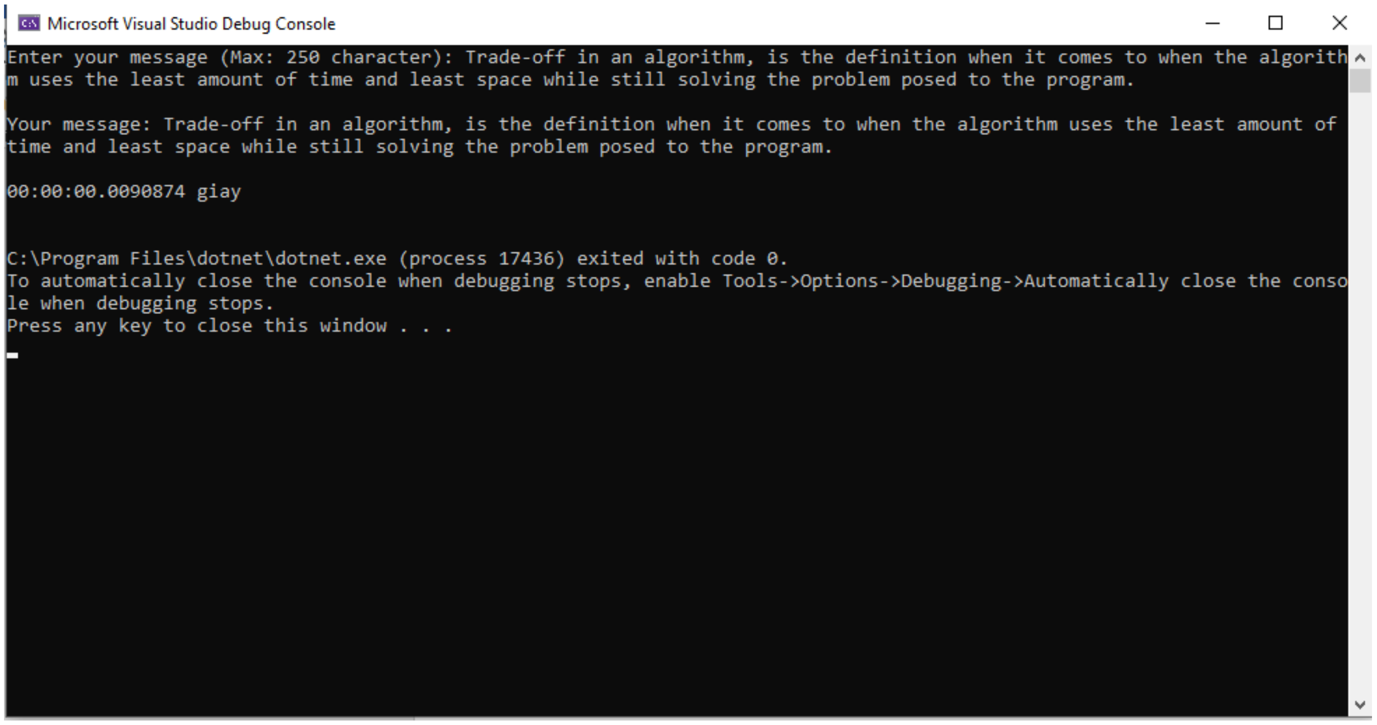
❖ Therefore, in reality, when we want to reduce the processing time of the algorithm, we have to increase the space, when we want the space to be less, we have to spend more time. For example: *I set out 2 cases, the same math and handle the same input (With the input string as shown below). Case 1 is declared string [50] and Case 2 is declared string [250]*


Microsoft Visual Studio Debug Console

```
Enter your message (Max: 250 character): Trade-off in an algorithm, is the definition when it comes to when the algorith
m uses the least amount of time and least space while still solving the problem posed to the program.

Your message: Trade-off in an algorithm, is the definition when it comes to when the algorithm uses the least amount of
time and least space while still solving the problem posed to the program.

00:00:00.0138478 giay


C:\Program Files\dotnet\dotnet.exe (process 3408) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

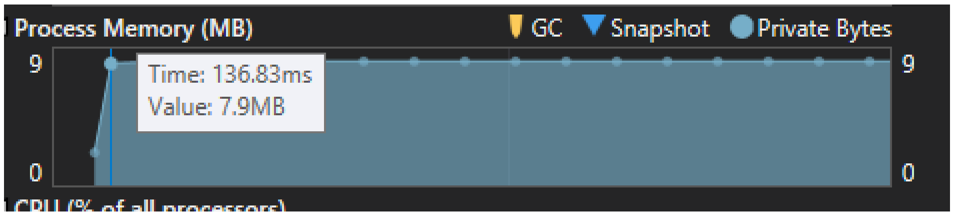


- In case 1: I declare **string [50]** and the algorithm is completed using **00.0138478** second. So in case 1, when reducing the variable declared at the time of receiving the input, the time will increase because it needs to be repeated.



Microsoft Visual Studio Debug Console

```
Enter your message (Max: 250 character): Trade-off in an algorithm, is the definition when it comes to when the algorith
m uses the least amount of time and least space while still solving the problem posed to the program.

Your message: Trade-off in an algorithm, is the definition when it comes to when the algorithm uses the least amount of
time and least space while still solving the problem posed to the program.

00:00:00.0090874 giay


C:\Program Files\dotnet\dotnet.exe (process 17436) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```



- In case 2: Now I have increased it to a **string[250]** and the time taken is really unexpected **00,0090874** second because the increase in the variable declared has reduced the execution time of the algorithm significantly because it processed the character only once.

	Time	Space
String[250]	00,0090874 second	7.9MB
String[50]	00.0138478 second	7.8MB

❖ To sum up, with **string[50]** the result is **00.0138478** second. For **string[250],** the result is after **00,0090874**. So, Space & Time is always the opposite of each other in the algorithm, increasing space decreases time and vice versa reducing space increases time. So, in the algorithm we need to find a **trade-off** at which the program can accept the value of time & space to suit the purpose of each program that serves its own input.

4. **D4 Evaluate three benefits of using implementation independent data structures:** (Rouse, 28 Feb 2006) (gsenviro, 2015)

- **Definition:** Data structure used to organize data storage and retrieval. There are many types of data structures (including: basic & advanced) but any type of data structure can be appropriately designed to organize and store data for different purposes. In computer programming, data structures can be selected and designed to store data for different algorithms. In it it includes information about the data value, the relationship between the data and the functions that can be applied to it.
- **Characteristics:**
  - Linear or non-linear: This feature describes how data is sorted by time, such as an **null array or a null string.**
  - Homogeneous or non-homogeneous: This characteristic talks about the data types of data in an archive with **one or more types.**
  - Static or dynamic: In the data structure, there are static data with the size, structure, fixed position and also the dynamic data whose size, structure and location can vary depending on the purpose of use.
- **Types of data structures:** Types of data structures classified by use purposes or algorithms applied, including: Arrays, Stacks, Queues, Linked List, Trees, Graphs, Tries – a trie, Hash tables -a hash table.
- **Benefits of Imdependent Data Structure:**
  - **Are necessary for design of efficient algorithms:** Data structure is the way data is organized so that some kind of activity on them is facilitated. Algorithms are the way to do something with data in the most efficient way. Basically, an algorithm uses a data structure to store process data for input, algorithm performance and output. For example: *Queue (not an algorithm, but close enough) can be built upon several structures (arrays, heaps, etc. with various pros / cons), and so on.*
  - **Allows safe storage of information on a computer. The information is then available for later use and can be used by multiple programs:** A data structure is an organized way to store and retrieve data, so a data structure can be applied by many algorithms. For example: ***An Array** is initialized and stores data for the **Bubble sorting algorithm**, but we can use that Array for the **Quick Sort sorting algorithm**. This helps us save less time and space in the computer.*
  - **Allows easier processing of data:** Because an algorithm uses a data structure to store process data for input, algorithm performance and output. And, a data structure is a specialized format for organizing, processing, retrieving and storing data. So, When using an algorithm on a data structure, the processing of an algorithm's data is a lot easier than not using a data structure. For example: ***Sorting algorithm** uses **Array data structure**, then the algorithm will use the properties of Array (Length, Location, input, output) to manipulate data easily.*

IV. CONCLUSION:

Through this report, I understand how to calculate the complexity of an algorithm / ADT, I further understand the nature of the data structure thanks to which I have built a Send Message program with up to 250 characters. .

V. REFERENCE:

Anon., 2016. *Độ phức tạp của thuật toán.* [Online]
Available at: https://techtalk.vn/do-phuc-tap-cua-thuat-toan.html
[Accessed 21 10 2019].

Anon., 2017. *Độ phức tạp thuật toán.* [Online]
Available at:
https://vi.wikipedia.org/wiki/%C4%90%E1%BB%99_ph%E1%BB%A9c_t%E1%BA%A1p_thu%E1%BA%ADt_to%C3%A1n
[Accessed 21 10 2019].

Anon., n.d. *Data Structure and Algorithms - Queue.* [Online]
Available at: https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm
[Accessed 21 10 2019].

gsenviro, 2015. *Advantages & Disadvantages of Data Structure.* [Online]
Available at: https://www.enotes.com/homework-help/what-advantages-disadvantages-data-stucture-479073
[Accessed 21 10 2019].

Rouse, M., 28 Feb 2006. *Data Structures.* [Online]
Available at: https://searchsqlserver.techtarget.com/definition/data-structure
[Accessed 21 10 2019].