

DATA STRUCTURES AND ALGORITHMS

ICT1018

Programmed using C#

By Mark Dingli

20703H

Plagiarism Declaration Form

Plagiarism is defined as “*the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines*" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the report submitted is my work, except where acknowledged and referenced. I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

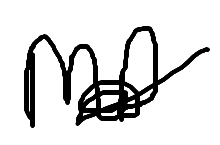
# Mark Dingli ICT 1018 27/5/22

**Student’s full name Study-unit code Date of submission**

# Data Structures and Algorithms Coursework

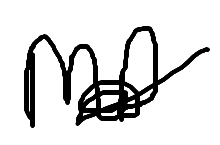
**Title of submitted work**:

**Student’s Signature**



**Statement of Completion**

* **Question 1 –** Attempted and works well.
* **Question 2 –** Attempted and works well.
* **Question 3** **–** Attempted and works well.
* **Question** **4 -** Attempted and works well.
* **Question 5** **-** Attempted and works well.
* **Question 6 -** Attempted and works well.
* **Question 7 -** Attempted and works well.
* **Question 8 -** Attempted and works well.
* **Question 9 -** Attempted and works well.
* **Question** **10 -** Attempted and works well.
* **Question 11** **–** Attempted but has the following bug:
* The final answers of sin() and cos() are not always correct
* **Question 12 –** Attempted and works well.

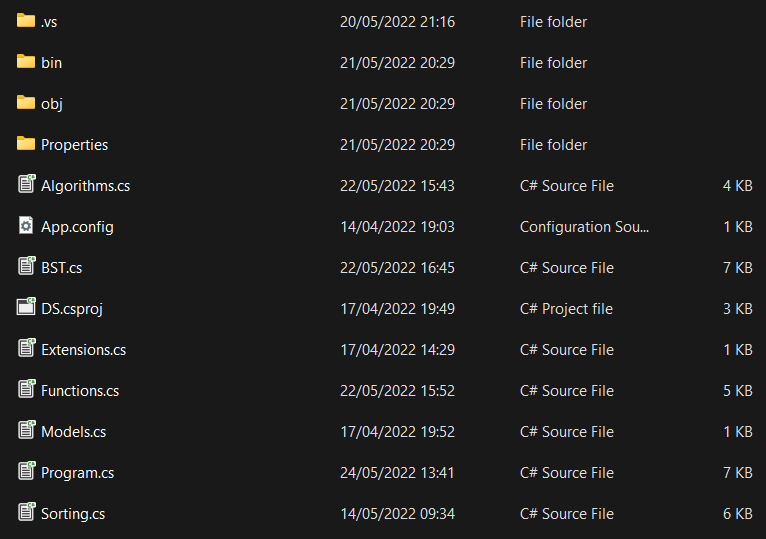


# Mark Dingli 27/5/22

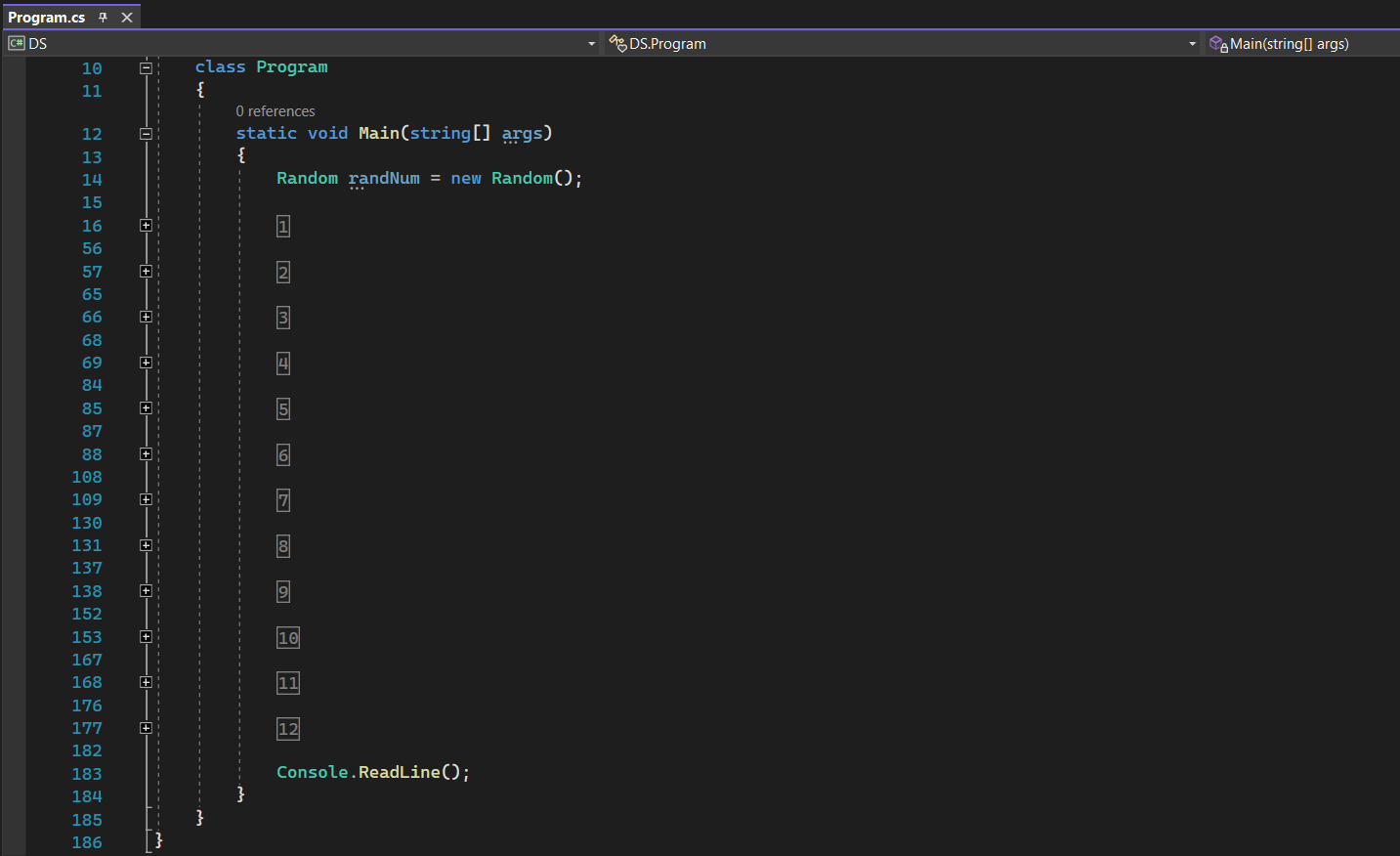
**Student’s full name Date Student Signature**

**Project Structure**

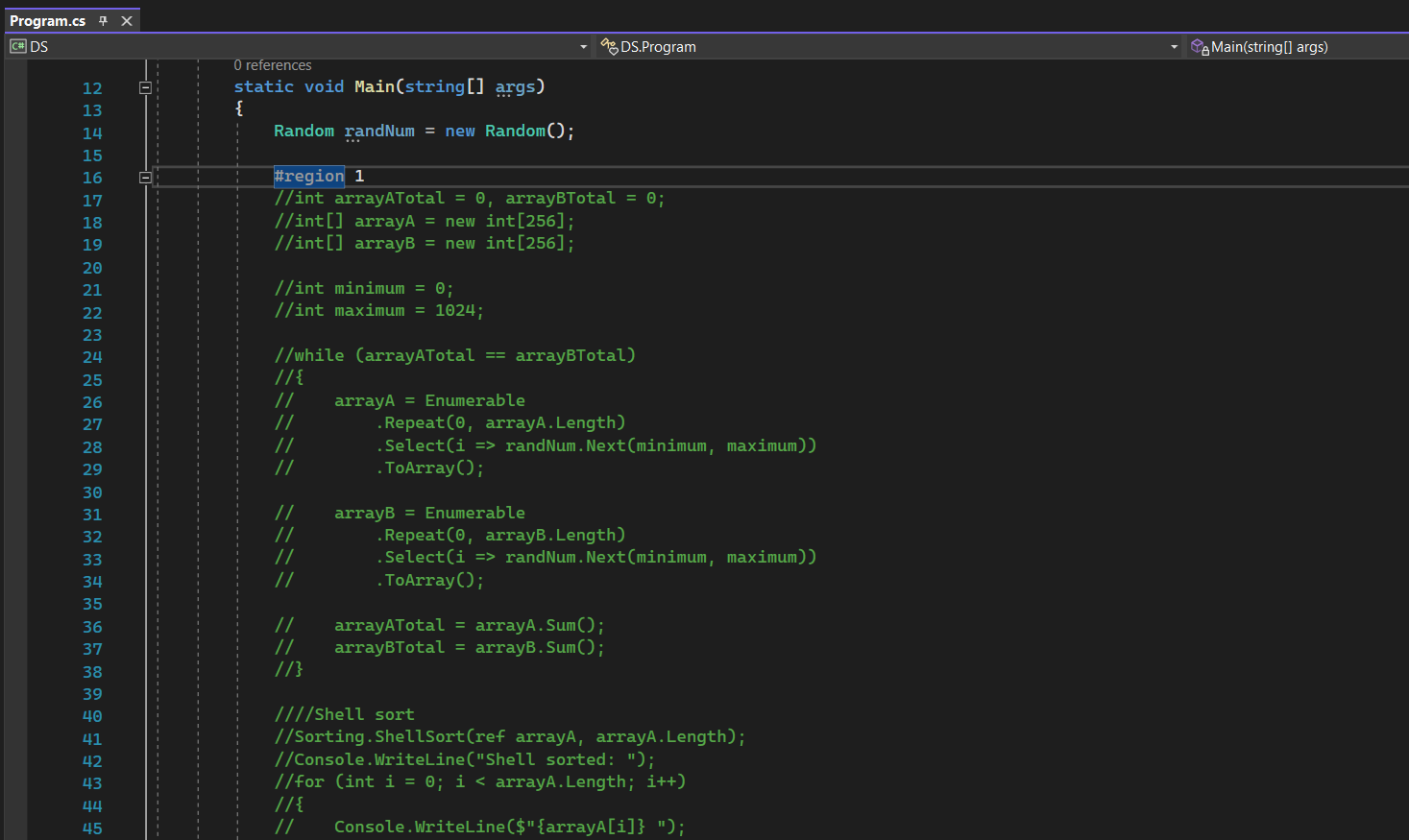
Visual Studio 2022 was used as an IDE in this Assignment. Language is C#.



Each question is in the main class *Program.cs*. The functions for each question can be found in the other classes which are; *Algorithms.cs*, *BST.cs*, *Functions.cs*, and *Sorting.cs*.



Each question in *Program.cs* is split into #regions. Each #region contains the code for the associated question number.



The code in the #region is commented and will need to be uncommented to run. When a new program needs to be executed, it is important to comment the other regions and uncomment the code which you wish to run.

All references to code snippets can be found in the last page of the documentation (page 36).

**Question 1**

**Code:**

#region 1

int arrayATotal = 0, arrayBTotal = 0;

int[] arrayA = new int[256];

int[] arrayB = new int[300];

int minimum = 0;

int maximum = 1024;

while (arrayATotal == arrayBTotal)

{

arrayA = Enumerable

.Repeat(0, arrayA.Length)

.Select(i => randNum.Next(minimum, maximum))

.ToArray();

arrayB = Enumerable

.Repeat(0, arrayB.Length)

.Select(i => randNum.Next(minimum, maximum))

.ToArray();

arrayATotal = arrayA.Sum();

arrayBTotal = arrayB.Sum();

}

//Shell sort

Sorting.ShellSort(ref arrayA, arrayA.Length);

Console.WriteLine("Shell sorted: ");

for (int i = 0; i < arrayA.Length; i++)

{

Console.WriteLine($"{arrayA[i]} ");

}

Console.WriteLine($"-------------------------------------------------");

//Quick sort

Sorting.QuickSorting(ref arrayB, 0, arrayB.Length - 1);

Console.WriteLine("Quick sorted: ");

for (int i = 0; i < arrayB.Length; i++)

{

Console.WriteLine($"{arrayB[i]} ");

}

Console.WriteLine($"-------------------------------------------------");

Console.WriteLine($"Sum of array A: {arrayA.Sum()} and length of {arrayA.Length}");

Console.WriteLine($"Sum of array A: {arrayB.Sum()} and length of {arrayB.Length}");

Console.WriteLine($"-------------------------------------------------");

#endregion

//Shell sort Function

public static void ShellSort(ref int[] arr, int n)

{

int i, j, pos, temp;

pos = 3;

while (pos > 0)

{

for (i = 0; i < n; i++)

{

j = i;

temp = arr[i];

while ((j >= pos) && (arr[j – pos] > temp))

{

arr[j] = arr[j – pos];

j = j – pos;

}

arr[j] = temp;

}

if (pos / 2 != 0)

{

pos = pos / 2;

}

else if (pos == 1)

{ pos = 0; }

else

{ pos = 1; }

}

}

//Quick Sort Function.

Public static void QuickSorting(ref int[] numbersArray, int lowestVal, int highestVal)

{

//Nothing to sort. The array is already sorted. Therefore nothing is returned.

If (highestVal – lowestVal < 1)

{

return;

}

// The low integer parameter is being stored in an integer called left

int left = lowestVal;

// The high integer parameter is being stored in an integer called right

int right = highestVal;

// An integer variable called pivot is being assigned the array value of left

// This means that the starting point will be from the left-most element

int pivot = numbersArray[left];

// A while-loop that will loop while the left value is less or equal to the right value

while (left <= right)

{

// A while-loop that will loop to increase the left value while the left is less than the pivot

while (numbersArray[left] < pivot)

{

// Left value being increased by 1

left++;

}

// A while-loop that will loop to decrease the right value while the right is greater than the pivot

while (numbersArray[right] > pivot)

{

// Right value being decreased by 1

right--;

}

// If loop condition to check whether the left value is smaller or equal to the right value

if (left <= right)

{

// The left value is smaller or equal to the right value

// Calling the element swapping method to sort out the array

SwappingTheElements(ref numbersArray, left, right);

// Left value is being increased by 1

left++;

// Right value is being decreased by 1

right--;

}

}

// Sorting using small values

QuickSorting(ref numbersArray, lowestVal, right);

// Sorting using big values

QuickSorting(ref numbersArray, left, highestVal);

}

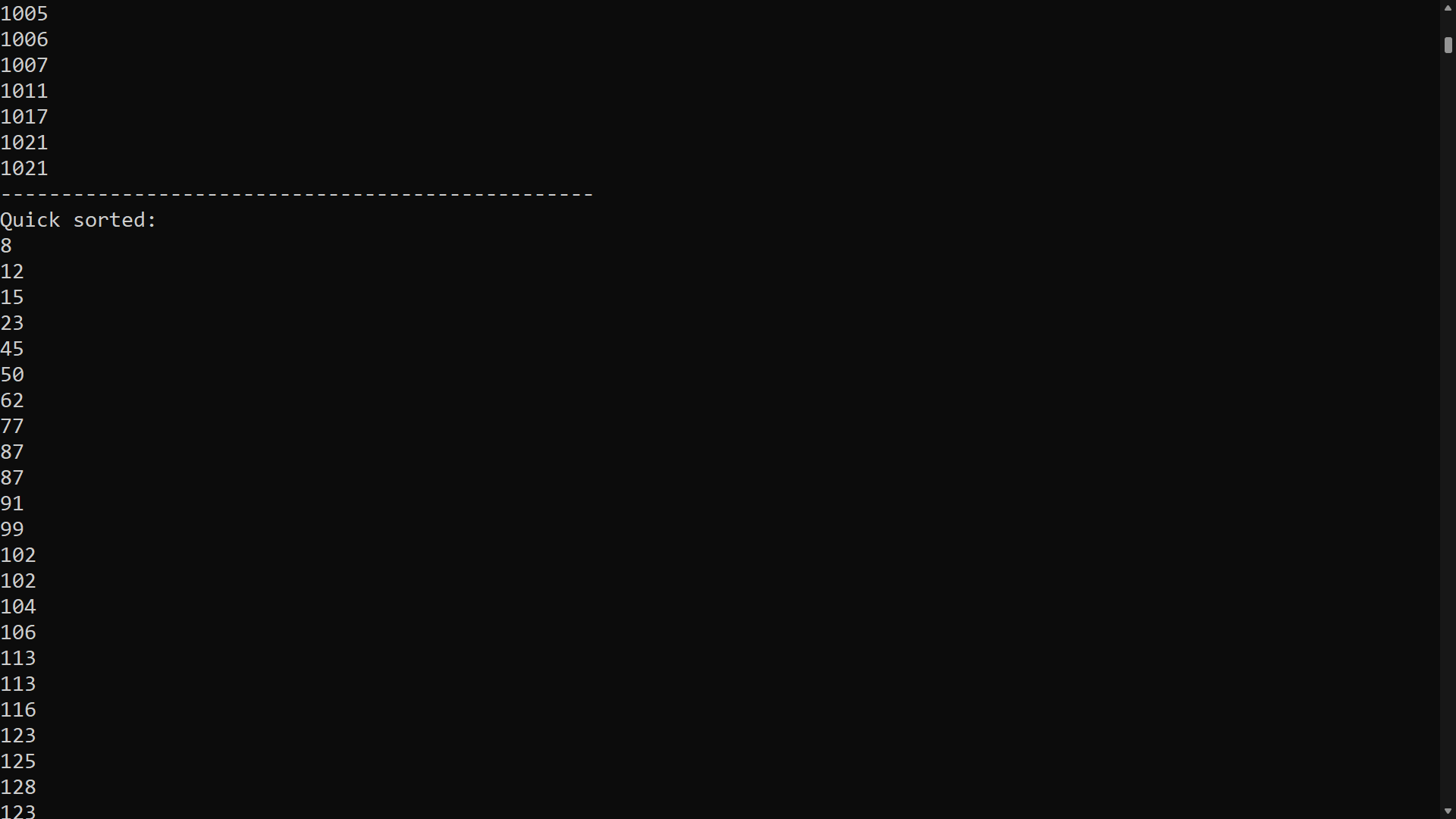
**Testing:**

This program is found in #region 1 of the *Program.cs* class. Both the functions (ShellSort and QuickSorting) used for Shell Sort as well as Quick Sort can be found in the *Sorting.cs* class. Random randNum = new Random(); generates a random number while the variables int minimum = 0; and int maximum = 1024; ensured that the arrays where populated with randomly generated integers between 0 and 1024. Both arrays contain at least 256 elements. ArrayA has 256 elements while ArrayB has 300 elements. Both arrays are of unequal size. This can be checked by using the *.Length* function. The sum of all the integers of the array as well as the length of each array is outputted at the end. Each time the program was run, a unique and different set of random numbers was generated.

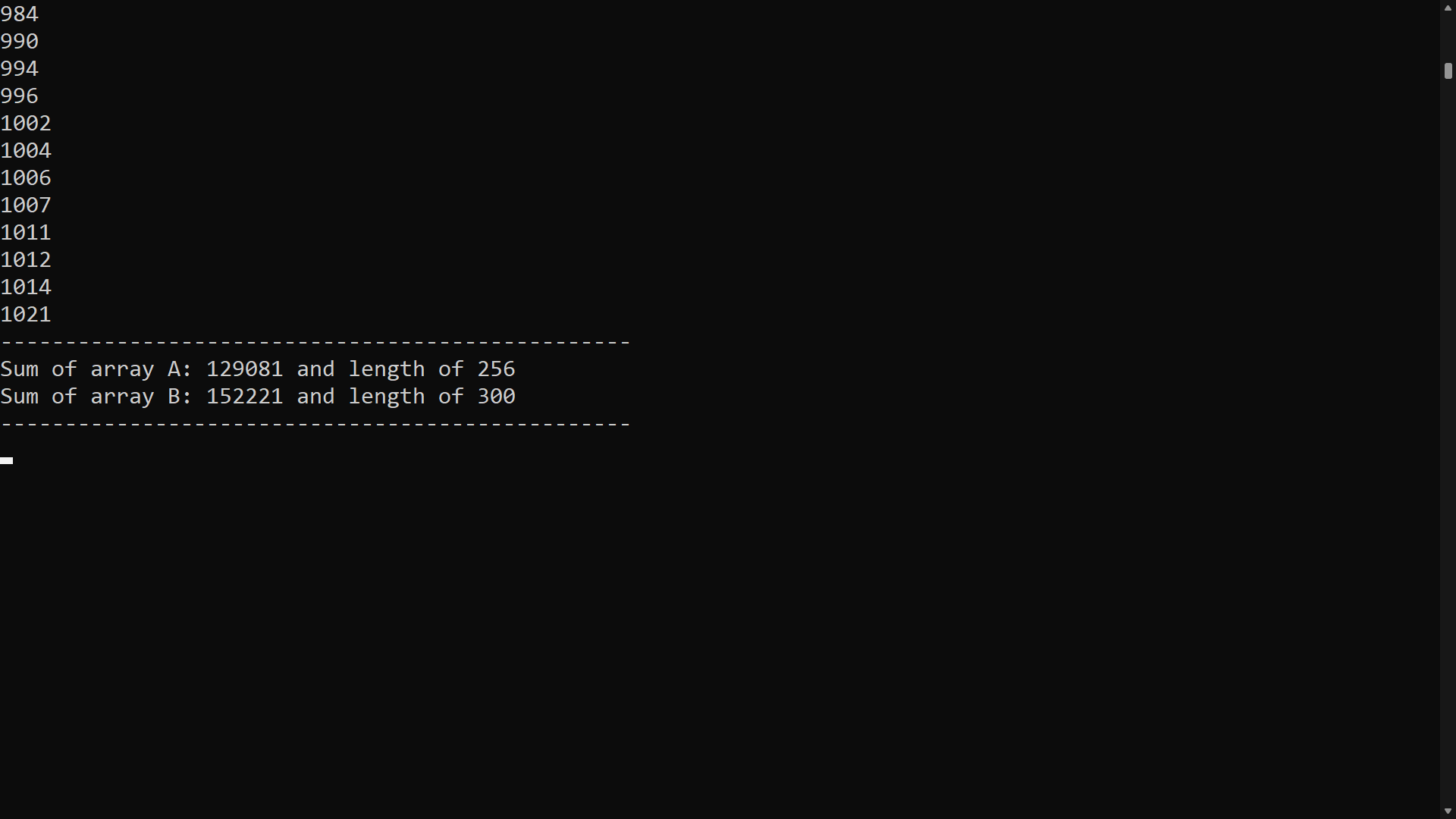
**Output:**



When the program is run, array A is sorted using Shell sort and outputted as seen in the screen shot above.



Array B is sorted using Quick sort and is also outputted as seen in the screen shot above.



The total sum and length of both arrays are outputted at the end as shown in the screenshot above.

**Question 2**

**Code:**

#region 2

var arrayResult = Sorting.MergeArrays(arrayA, arrayB);

Console.WriteLine("Merge:");

for (int i = 0; i < arrayResult.Length; i++)

{

Console.WriteLine($"{arrayResult[i]} ");

}

Console.WriteLine($"-------------------------------------------------");

Console.WriteLine($"Total sum of array A and array B is: {arrayA.Sum() + arrayB.Sum()}");

Console.WriteLine($"Array result of Merged array C is: {arrayResult.Sum()} and has length of: {arrayResult.Length}");

Console.WriteLine($"-------------------------------------------------");

#endregion

//Function to merge arrays

public static int[] MergeArrays(int[] arrayA, int[] arrayB)

{

int[] arrayResult = new int[arrayA.Length + arrayB.Length];

//index for result

int n = 0;

//index for arrayA

int i = 0;

//index for arrayB

int j = 0;

//until any one of arrays all elements are traversed

while (i < arrayA.Length && j < arrayB.Length)

{

//comparing items of arrayA and arrayB

if (arrayA[i] < arrayB[j])

{

//inserting arrayA item since it is lower

arrayResult[n] = arrayA[i];

//incrementing i, because arrayA's item is traversed

i++;

}

else

{

arrayResult[n] = arrayB[j];

j++;

}

//incrementing since one item has been inserted to result array

n++;

}

//if any elements are left in arrayA, simply inserting all to result

while (i < arrayA.Length)

{

arrayResult[n] = arrayA[i];

i++;

n++;

}

//if any elements are left in arrayB, simply inserting all to result

while (j < arrayB.Length)

{

arrayResult[n] = arrayB[j];

j++;

n++;

}

return arrayResult;

}

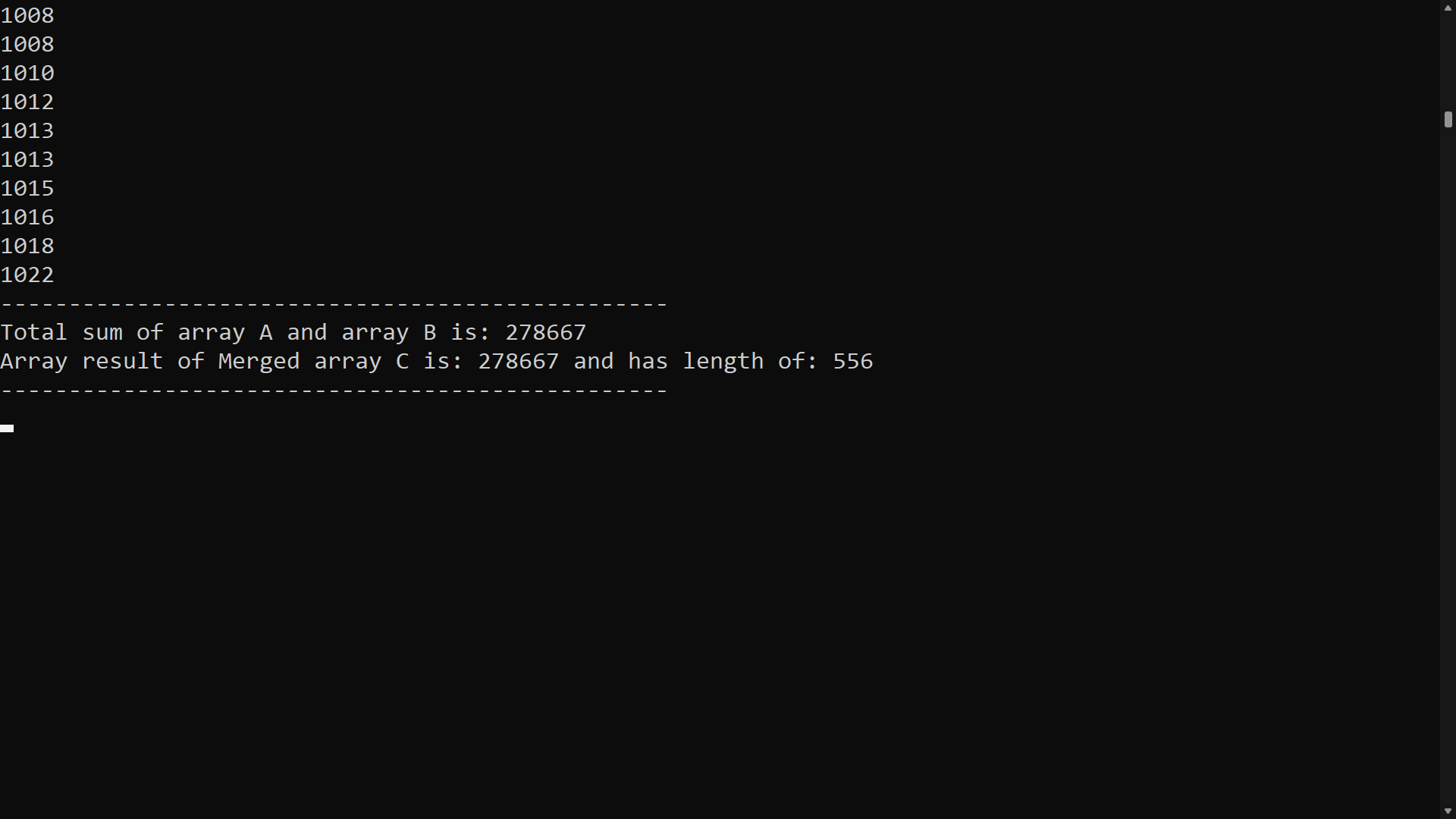
**Testing:**

This program is found in #region 2 of the *Program.cs* class. For this program to work, it is important that the code for #region 1 needs to be *uncommented* and executed with the code from #region 2. The function (MergeArrays) used for merging the arrays can be found in the *Sorting.cs* class. In this case since two different arrays, A and B from the previous question, are being merged together, some duplicate(same) numbers will be found in the merged array C since arrays A and B could have had the same numbers. The time complexity for this program is Big **O(3n)**, therefore it has a linear time complexity. The last two lines of the output can be used to show that ArrayC has all the elements from the merged array (total sum), and that ArrayC is of size arrayA + arrayB.

**Output:**



When the program is run, arrays A and B are merged into Array C and outputted as seen in the screen shot above.



The total sum and length of arrayA and arrayB as well as the total sum and length of ArrayC are outputted at the end as shown in the screenshot above.

**Question 3**

**Code:**

#region 3

// Inspired by https://stackoverflow.com/questions/67760895/c-finding-the-extreme-points-of-an-algorithm

int[] nums = { 0, 5, 3, 6, 8, 7, 15, 9 };

bool sorted = true;

Console.WriteLine($"Extreme points are:");

for (int i = 1; i < nums.Length - 1; i++)

{

if (nums[i] > nums[i - 1] && nums[i] > nums[i + 1] || nums[i] < nums[i - 1] && nums[i] < nums[i + 1])

{

sorted = false;

Console.WriteLine(nums[i]);

}

}

if (sorted) Console.WriteLine("SORTED");

#endregion

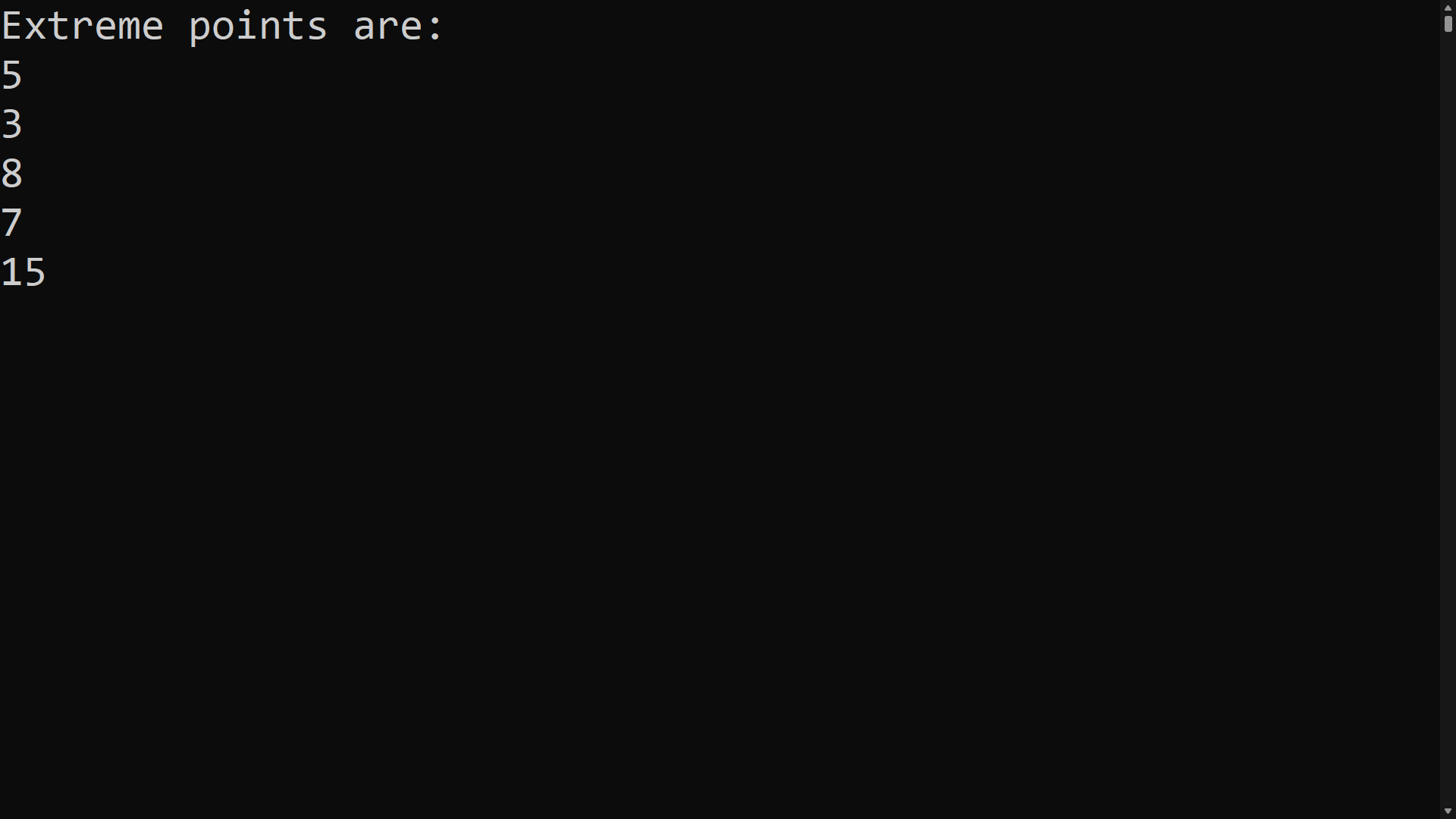
**Testing:**

This program is found in #region 3 of the *Program.cs* class. This program takes an array list called *nums* and will output “SORTED” if the array is sorted or it will output the extreme elements of the list if it is not sorted.

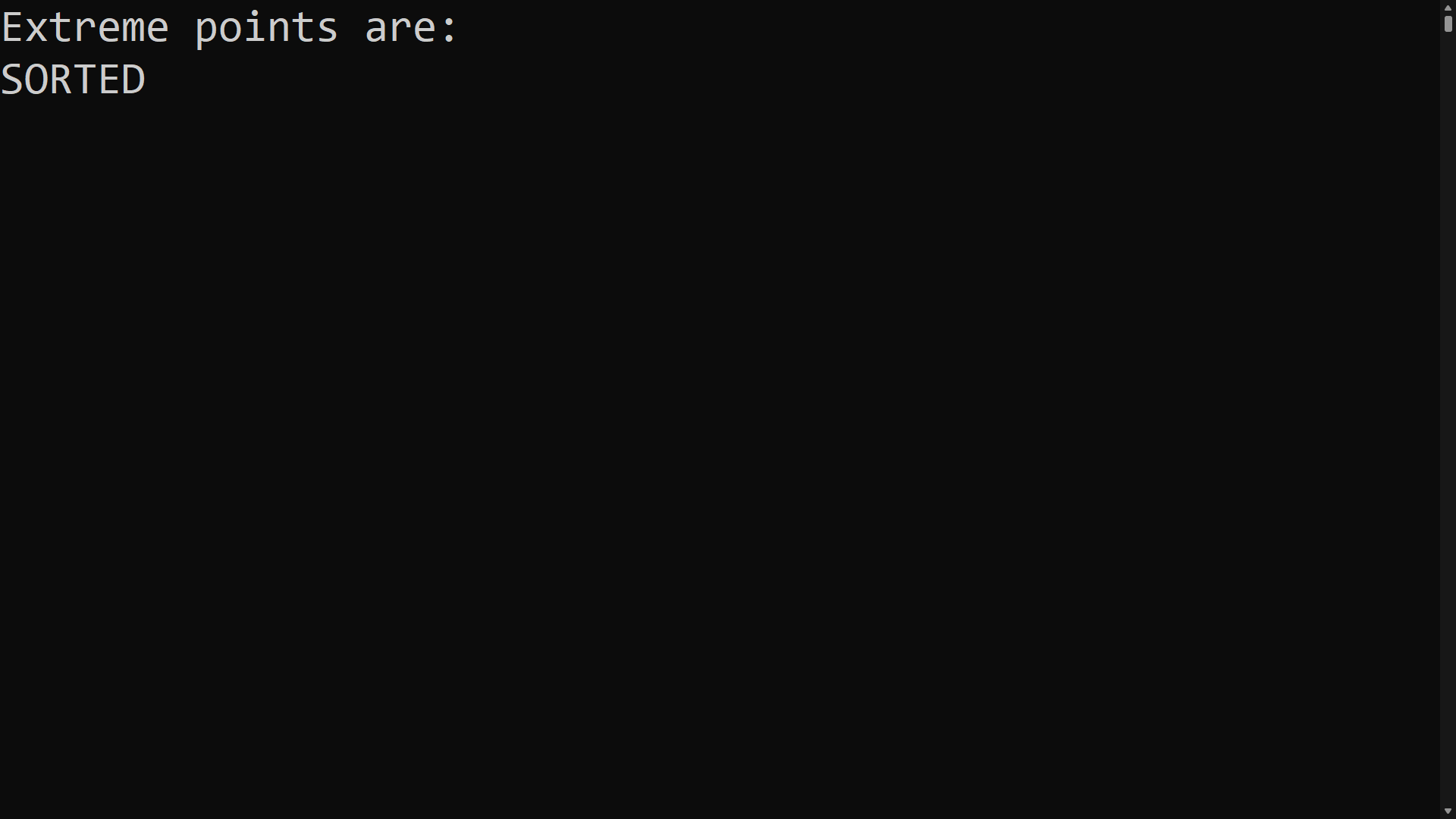
**Do you agree that an array has no extreme points if and only if it is sorted?**

- If the array is sorted, then no element has two adjacent neighbours which are both either higher or lower than the element. On the other hand, if an element has two adjacent neighbours both of which are either higher or lower, then the array is unsorted.

**Output:**



When the program is run with an unsorted array, the extreme points of the given array are printed as shown in the screenshot above.



When the program is run with a sorted array, “SORTED” is printed out as shown in the screenshot above.

**Question 4**

**Code:**

#region 4

// Inspired by https://www.geeksforgeeks.org/find-all-pairs-possible-from-the-given-array/

int[] arrayQuestion4 = new int[100];

int count = 0;

while (count < arrayQuestion4.Length)

{

int randomNumberQuestion4 = randNum.Next(1, 1025);

while (arrayQuestion4.Contains(randomNumberQuestion4))

{

randomNumberQuestion4 = randNum.Next(1, 1025);

}

arrayQuestion4[count] = randomNumberQuestion4;

count++;

}

Functions.FindPairs(arrayQuestion4, arrayQuestion4.Length);

#endregion

// Question 4

// Function to find out four elements

// in array whose product is ab = cd

public static void FindPairs(int[] arr, int n)

{

bool found = false;

Dictionary<int, Pair> hp = new Dictionary<int, Pair>();

for (int i = 0; i < n; i++)

{

for (int j = i + 1; j < n; j++)

{

int prod = arr[i] \* arr[j];

if (!hp.ContainsKey(prod))

{

hp.Add(prod, new Pair(i, j));

}

else

{

Pair p = hp[prod];

Console.WriteLine(arr[p.first]

+ " " + arr[p.second]

+ " " + "and" + " " +

arr[i] + " " + arr[j]);

found = true;

}

}

}

if (found == false)

{

Console.WriteLine("No pairs Found");

}

}

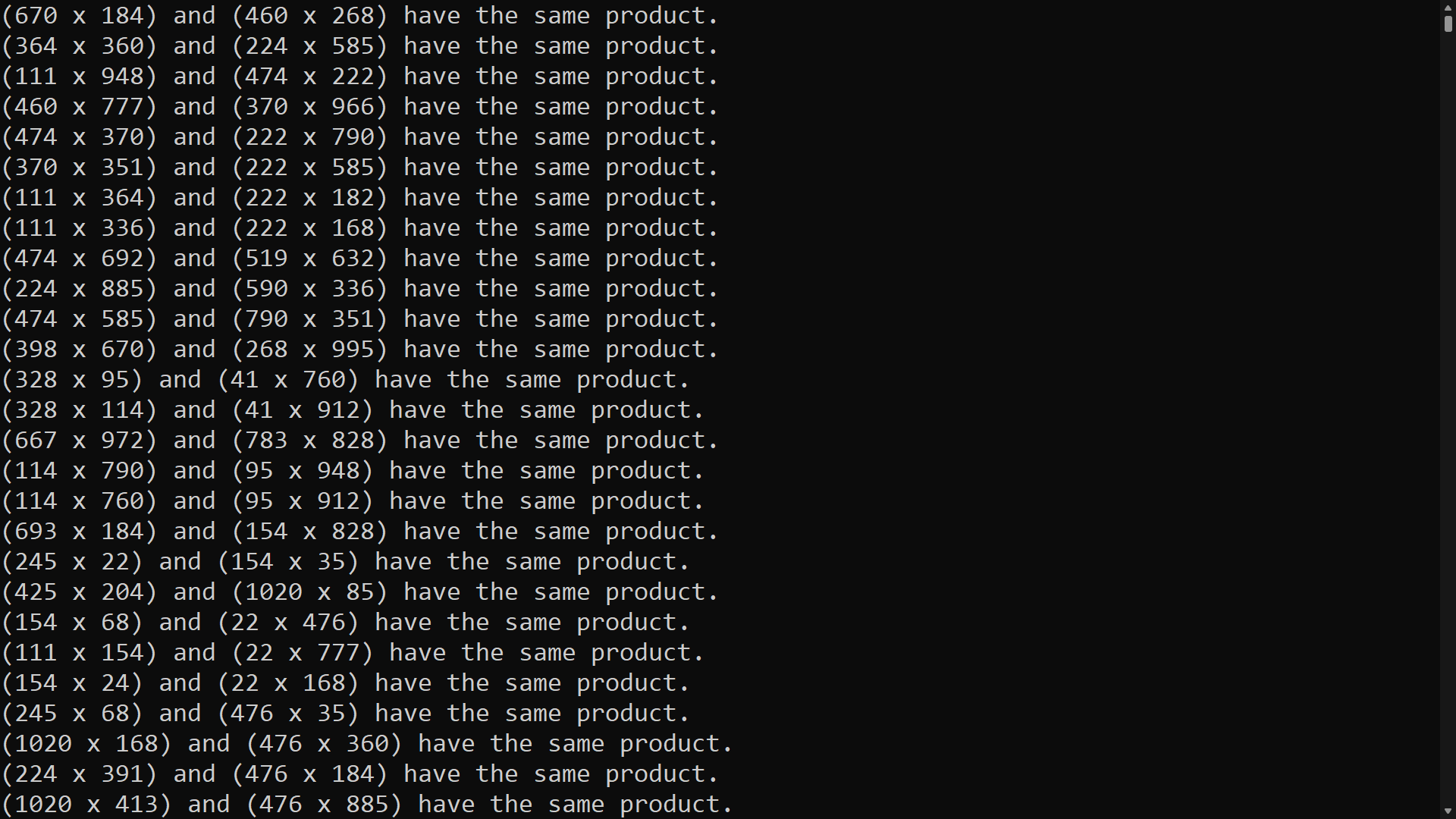
**Testing:**

This program is found in #region 4 of the *Program.cs* class. The function (FindPairs) used to find all 2-pairs of integers that have the same product can be found in the *Functions.cs* class. All the randomly generated integers are between 1 and 1024. Every time the program is run a random list of unsorted integers is generated and outputted. 2 different pairs that have the same product are also outputted. If no pairs are found the message "No pairs Found" will be outputted. The code was run multiple times to check that the product of both generated pairs match. The while loop is there to make sure that a ≠ b ≠ c ≠ d. If the integer already exists in the array, a new random number is generated.

**Output:**



When the program is run, an unsorted list of integers is outputted as shown above.



All the 2-pairs of integers that have the same product are outputted as shown above.

**Question 5**

**Code:**

#region 5

// Inspired by https://iq.opengenus.org/arithmetic-expression-evaluation-using-stack/

string equation = "6872-+\*";

// creating a stack

Stack<int> stack = new Stack<int>();

for (int i = 0; i < equation.Length; i++)

{

char x = equation[i];

// If the character is an operand, push it into the stack.

if (int.TryParse(x.ToString(), out int res))

{

stack.Push(x - '0');

}

// If the character is an operator, pop the 2 top most elements

// from the stack and perform the operation. Push the result back to the stack.

else

{

Console.WriteLine("Stack before operations:");

Functions.PrintStack(stack);

Console.WriteLine(Environment.NewLine);

int operand1 = stack.Pop();

int operand2 = stack.Pop();

switch (x)

{

case '+':

stack.Push(operand2 + operand1);

break;

case '-':

stack.Push(operand2 - operand1);

break;

case '\*':

stack.Push(operand2 \* operand1);

break;

case '/':

stack.Push(operand2 / operand1);

break;

}

Console.WriteLine("Stack after operations:");

// function to print stack

Functions.PrintStack(stack);

Console.WriteLine(Environment.NewLine);

}

}

Console.WriteLine($"RPN: {stack.Pop()}");

#endregion

//Inspired by: https://www.geeksforgeeks.org/print-stack-elements-from-top-to-bottom/

public static void PrintStack(Stack<int> s)

{

// If stack is empty

if (s.Count == 0)

return;

// Extract top of the stack

int x = s.Peek();

// Pop the top element

s.Pop();

// Print the current top

// of the stack i.e., x

Console.Write(x + " ");

// Proceed to print

// remaining stack

PrintStack(s);

// Push the element back

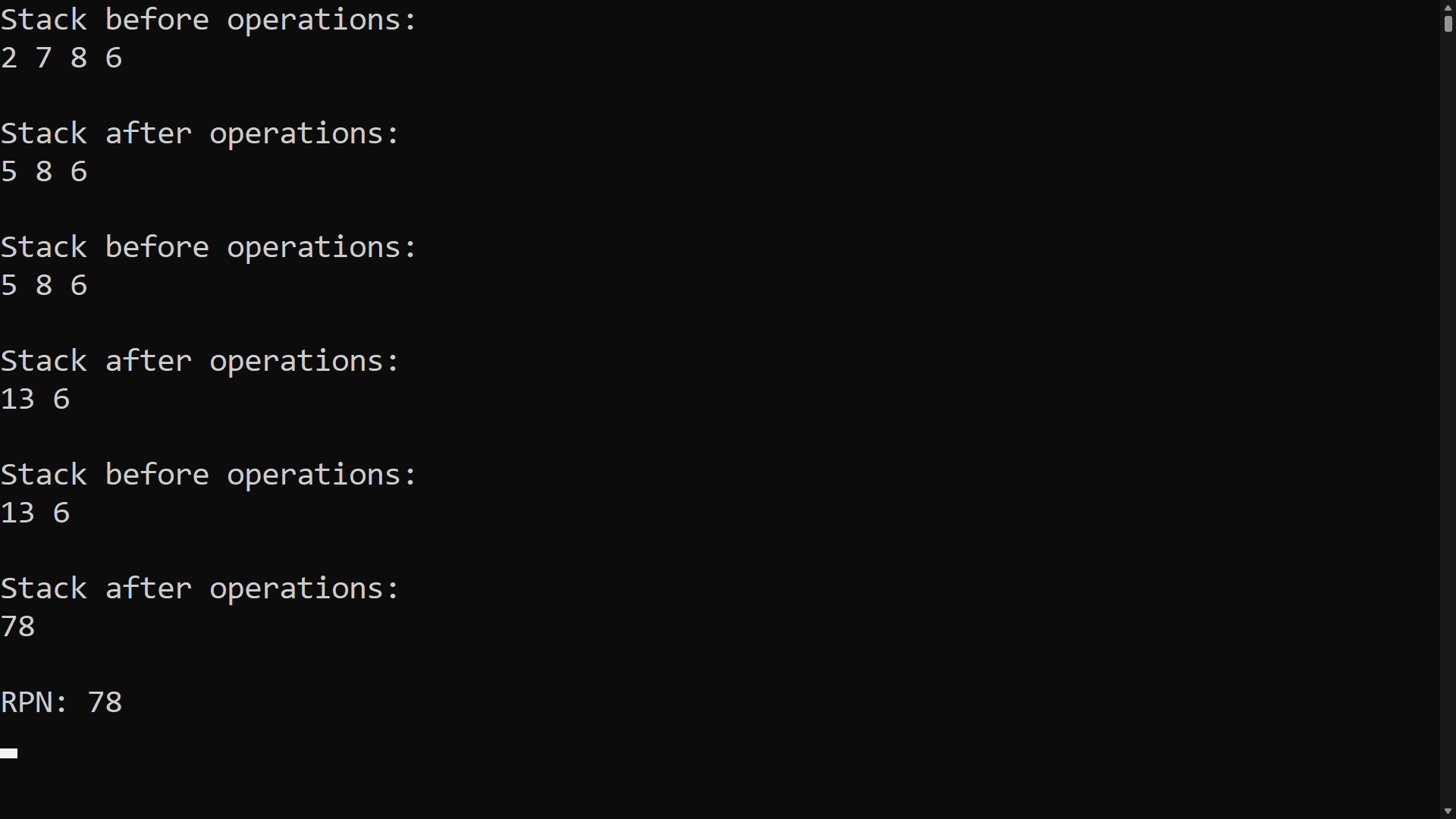
s.Push(x);

}

**Testing:**

This program is found in #region 5 of the *Program.cs* class. This program uses an ADT Stack to evaluate arithmetic expressions in RPN format. To change the expression just change the variable *“equation”* in string equation = "6872-+\*";. The function PrintStack() found in the *Functions.cs* class, prints the stack during evaluation. For testing, different equations were evaluated and tested with an online RPN calculator for which the final answers were correct.

**Output:**



When the program is run, the equation written in the code is pushed into an ADT Stack and evaluated using arithmetic expressions in RPN format. The answer is outputted.

**Question 6**

**Code:**

#region 6

//Check if number is prime

long question6 = 0;

do{

Console.WriteLine("Check if prime number:");

question6 = Convert.ToInt64(Console.ReadLine());

if(question6 <= 0)

{

Console.WriteLine("Enter a number greater than 0!");

Console.WriteLine("------------------------------------------");

}

} while(question6 <= 0);

if (question6.IsPrime())

{

Console.WriteLine("It is prime");

Console.WriteLine("------------------------------------------");

}

else

{

Console.WriteLine("It is not prime");

Console.WriteLine("------------------------------------------");

}

//Sieve of Eratosthenes

var ret = Algorithms.SieveofEratosthenes();

Console.WriteLine($"There are {ret.Count} primes up to {ret.Maximum}");

#endregion

// Inspired By: https://www.geeksforgeeks.org/sieve-of-eratosthenes/

public static SOEReturnObject SieveofEratosthenes()

{

// calculate how many prime numbers there are from 2 up to a MAX.

const int MAX = 200;

// Create an array of boolean values indicating whether a number is prime.

// Start by assuming all numbers are prime by setting them to true.

bool[] primes = new bool[MAX + 1];

for (int i = 0; i < primes.Length; i++)

{

primes[i] = true;

}

// Loop through a portion of the array (up to the square root of MAX). If

// it's a prime, ensure all multiples of it are set to false, as they

// clearly cannot be prime.

for (int i = 2; i < Math.Sqrt(MAX) + 1; i++)

{

if (primes[i - 1])

{

for (int j = (int)Math.Pow(i, 2); j <= MAX; j += i)

{

primes[j - 1] = false;

}

}

}

// Output the results

int total = 0;

for (int i = 2; i < primes.Length; i++)

{

if (primes[i - 1])

{

Console.WriteLine(i);

total++;

}

}

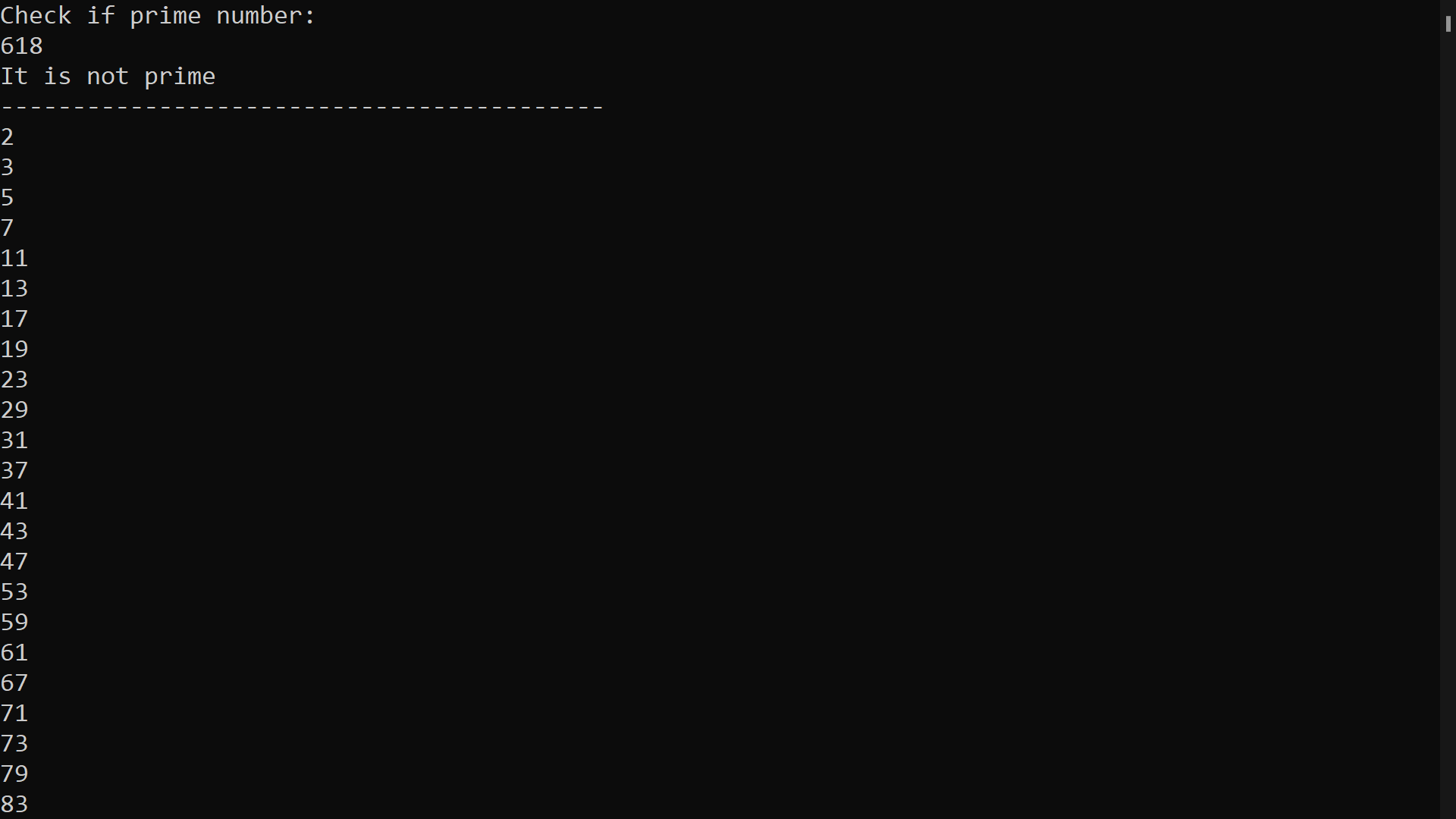
return new SOEReturnObject() { Count = total, Maximum = MAX };

}

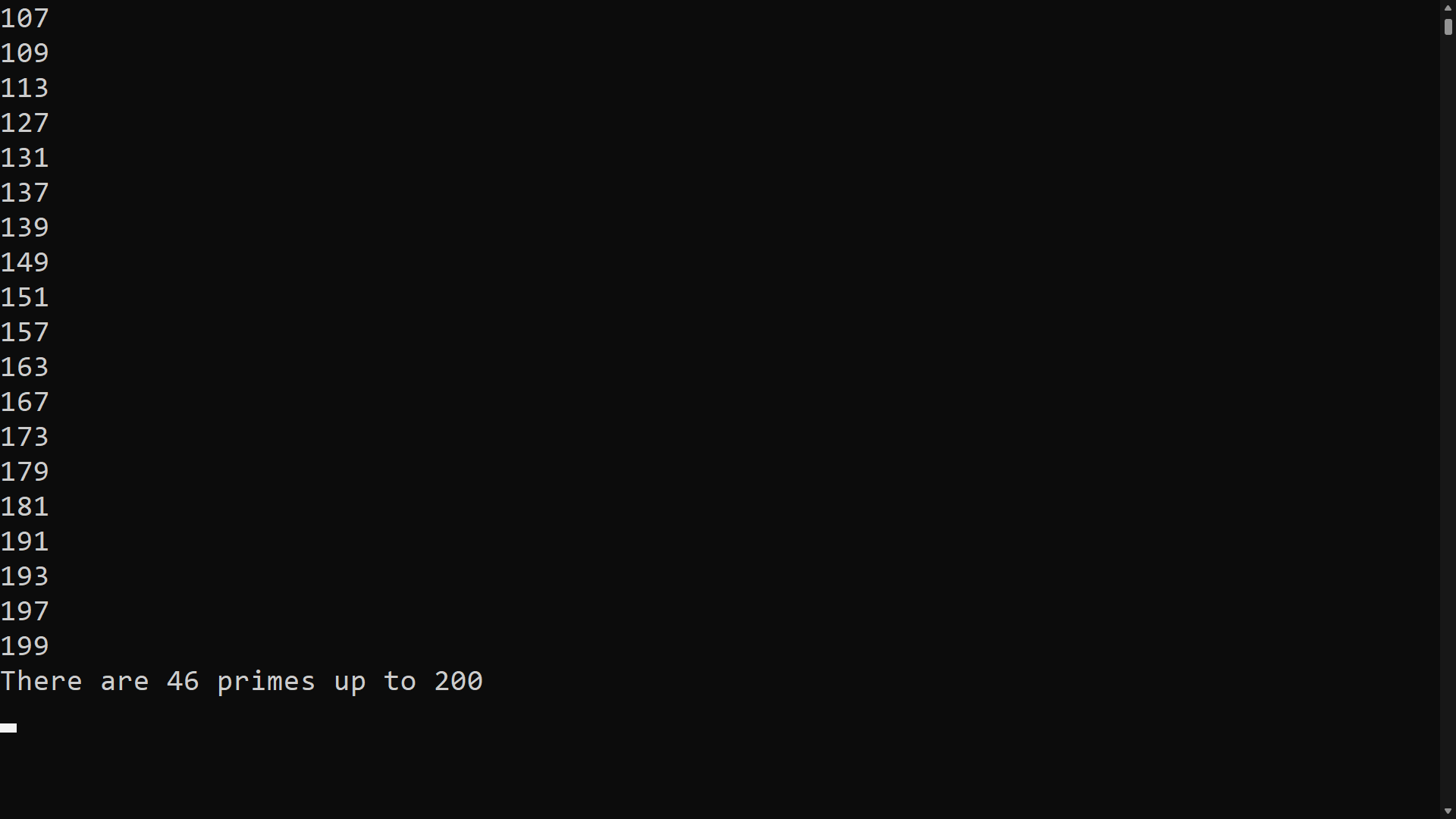
**Testing:**

This program is found in #region 6 of the *Program.cs* class. This program allows the user to enter a number and outputs "It is prime" if the inputted number is prime or "It is not prime" if it is not prime. Since you cannot find the prime of 0 or a negative number, a do, while loop was used for error checking. If 0 or a negative number is inputted, an error message will be outputted until the user enters a number that is greater than 0. The function SieveofEratosthenes() found in the *Algorithms.cs* class, creates an array and implements the Sieve of Eratosthenes algorithm which calculates how many prime numbers there are from 2 up to a specific number. It also outputs all the prime numbers. In this case from 2 to 200. To change this, just change the const int MAX = 200; (*line 9 in Algorithms.cs*) to any number.

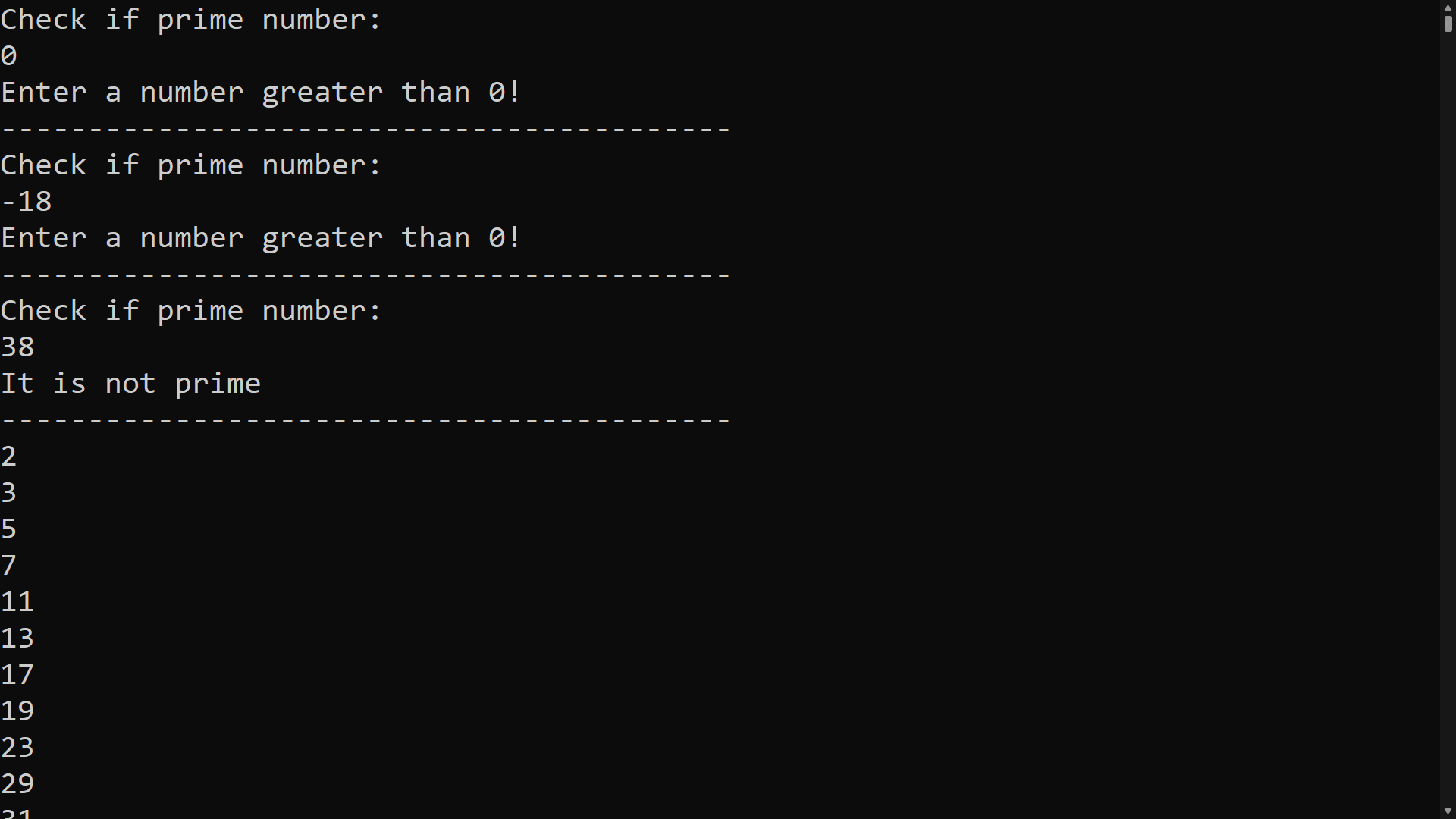
**Output:**



When the program is run, the user enters a number and the algorithm will check if it is prime or not.



All of the prime numbers as well as the number of primes between 2 up to 200 are printed.



Error message if inputted number is 0 or a negative.

**Question 7**

**Code:**

#region 7

BTree binarySearchTree = new BTree();

Node root = new Node();

int question7 = 1;

// Create a list

List<int> bstNumbers = new List<int>();

// User input

Console.WriteLine("Input numbers to form BST. Enter -1 to stop.");

// User input validation

while (question7 > 0)

{

question7 = Convert.ToInt32(Console.ReadLine());

if (question7 > 0)

{

bstNumbers.Add(question7);

}

}

foreach (int number in bstNumbers.Distinct())

{

binarySearchTree.Add(number);

}

binarySearchTree.Print();

#endregion

using System;

using System.Collections.Generic;

namespace DS

{

// Inspired by: https://stackoverflow.com/questions/36311991/c-sharp-display-a-binary-search-tree-in-console

public class BST

{

public enum NodePosition

{

left,

right,

center

}

public class BNode

{

// Variable declaration

public int item;

public BNode right;

public BNode left;

public BNode(int item)

{

this.item = item;

}

// Function to print Values

private void PrintValue(string value, NodePosition nodePostion)

{

switch (nodePostion)

{

case NodePosition.left:

PrintLeftValue(value);

break;

case NodePosition.right:

PrintRightValue(value);

break;

case NodePosition.center:

Console.WriteLine(value);

break;

default:

throw new NotImplementedException();

}

}

// Function to print left values

private void PrintLeftValue(string value)

{

// Output left node

Console.ForegroundColor = ConsoleColor.Magenta;

Console.Write("L:");

Console.ForegroundColor = (value == "-") ? ConsoleColor.Red : ConsoleColor.Gray;

Console.WriteLine(value);

Console.ForegroundColor = ConsoleColor.Gray;

}

// Function to print right values

private void PrintRightValue(string value)

{

// Output right node

Console.ForegroundColor = ConsoleColor.Green;

Console.Write("R:");

Console.ForegroundColor = (value == "-") ? ConsoleColor.Red : ConsoleColor.Gray;

Console.WriteLine(value);

Console.ForegroundColor = ConsoleColor.Gray;

}

// Function to print connecting lines between nodes and values

public void PrintPretty(string indent, NodePosition nodePosition, bool last, bool empty)

{

Console.Write(indent);

if (last)

{

Console.Write("└─");

indent += " ";

}

else

{

Console.Write("├─");

indent += "| ";

}

// Printing an empty node/value

var stringValue = empty ? "-" : item.ToString();

PrintValue(stringValue, nodePosition);

if (!empty && (this.left != null || this.right != null))

{

if (this.left != null)

this.left.PrintPretty(indent, NodePosition.left, false, false);

else

PrintPretty(indent, NodePosition.left, false, true);

if (this.right != null)

this.right.PrintPretty(indent, NodePosition.right, true, false);

else

PrintPretty(indent, NodePosition.right, true, true);

}

}

}

public class BTree

{

private BNode \_root;

private int \_count;

private IComparer<int> \_comparer = Comparer<int>.Default;

public BTree()

{

\_root = null;

\_count = 0;

}

public bool Add(int Item)

{

if (\_root == null)

{

\_root = new BNode(Item);

\_count++;

return true;

}

else

{

return Add\_Sub(\_root, Item);

}

}

private bool Add\_Sub(BNode Node, int Item)

{

if (\_comparer.Compare(Node.item, Item) < 0)

{

if (Node.right == null)

{

Node.right = new BNode(Item);

\_count++;

return true;

}

else

{

return Add\_Sub(Node.right, Item);

}

}

else if (\_comparer.Compare(Node.item, Item) > 0)

{

if (Node.left == null)

{

Node.left = new BNode(Item);

\_count++;

return true;

}

else

{

return Add\_Sub(Node.left, Item);

}

}

else

{

return false;

}

}

// Function to print

public void Print()

{

\_root.PrintPretty("", NodePosition.center, true, false);

}

}

}

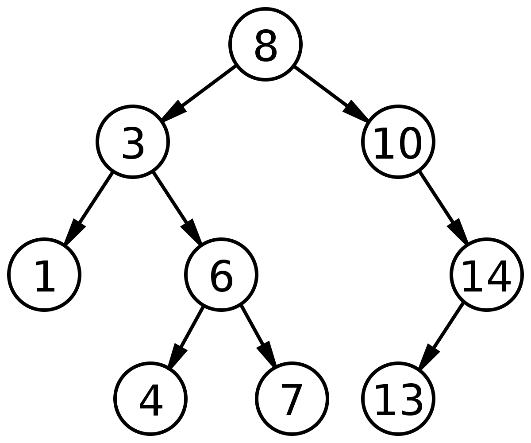
}

**Testing:**

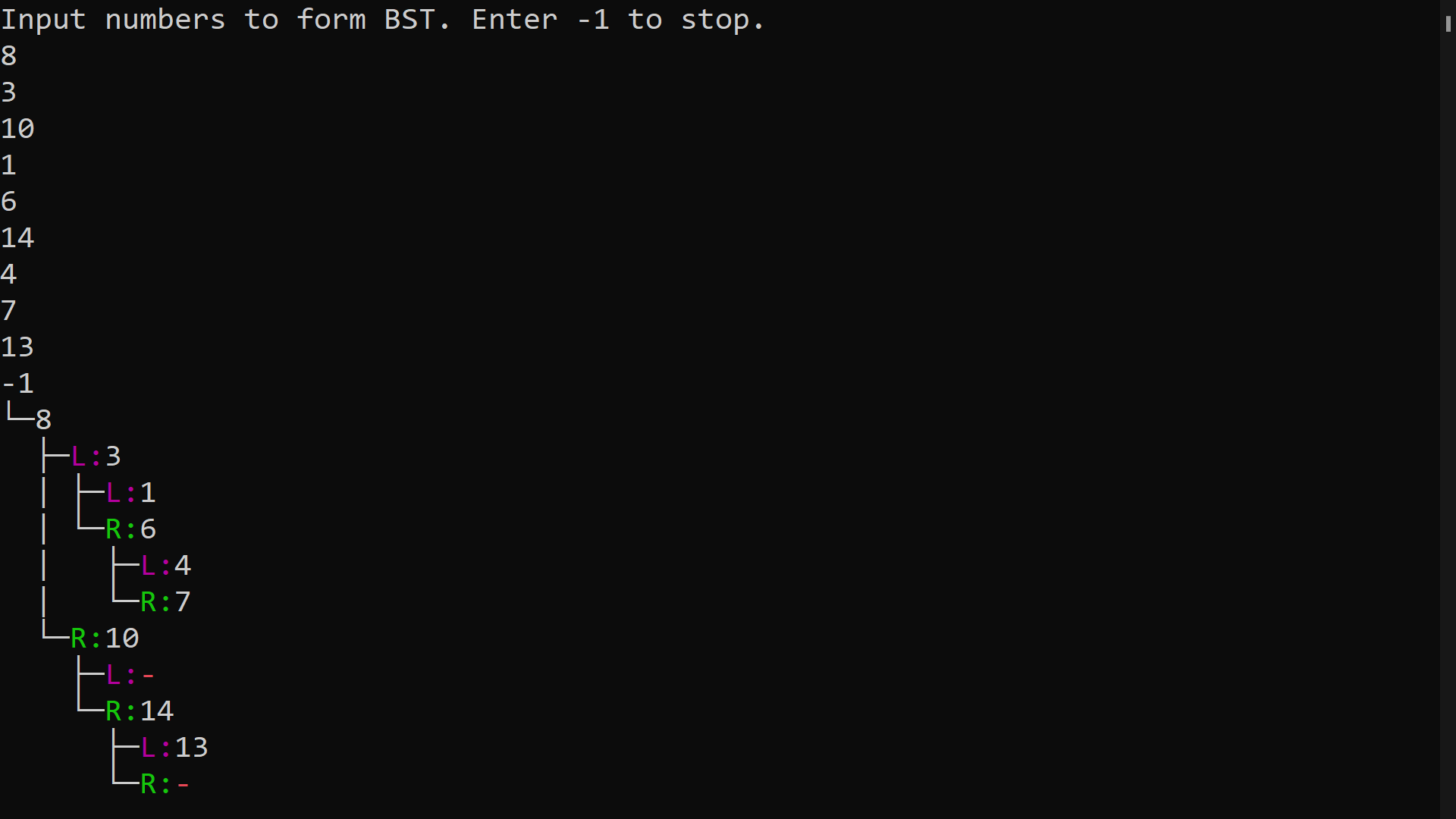
This program is found in #region 7 of the *Program.cs* class. This program allows the user to enter a numbers and then displays a Binary Search Tree (BST). The function BST(), found in the *BST.cs* class, takes the user input and will built and output the Binary Search Tree. The function Print(), found in the BST.cs class is used to print the BST. Since a BST cannot have repeated values, when repeated numbers are inputted, only one will be saved, the others will be ignored. Since negative numbers cannot be stored in a BST table, when a negative number is entered as an input, the program will output the tree with the numbers entered before excluding the negative number. This means, to stop the input, just enter any negative number.

**Output:**

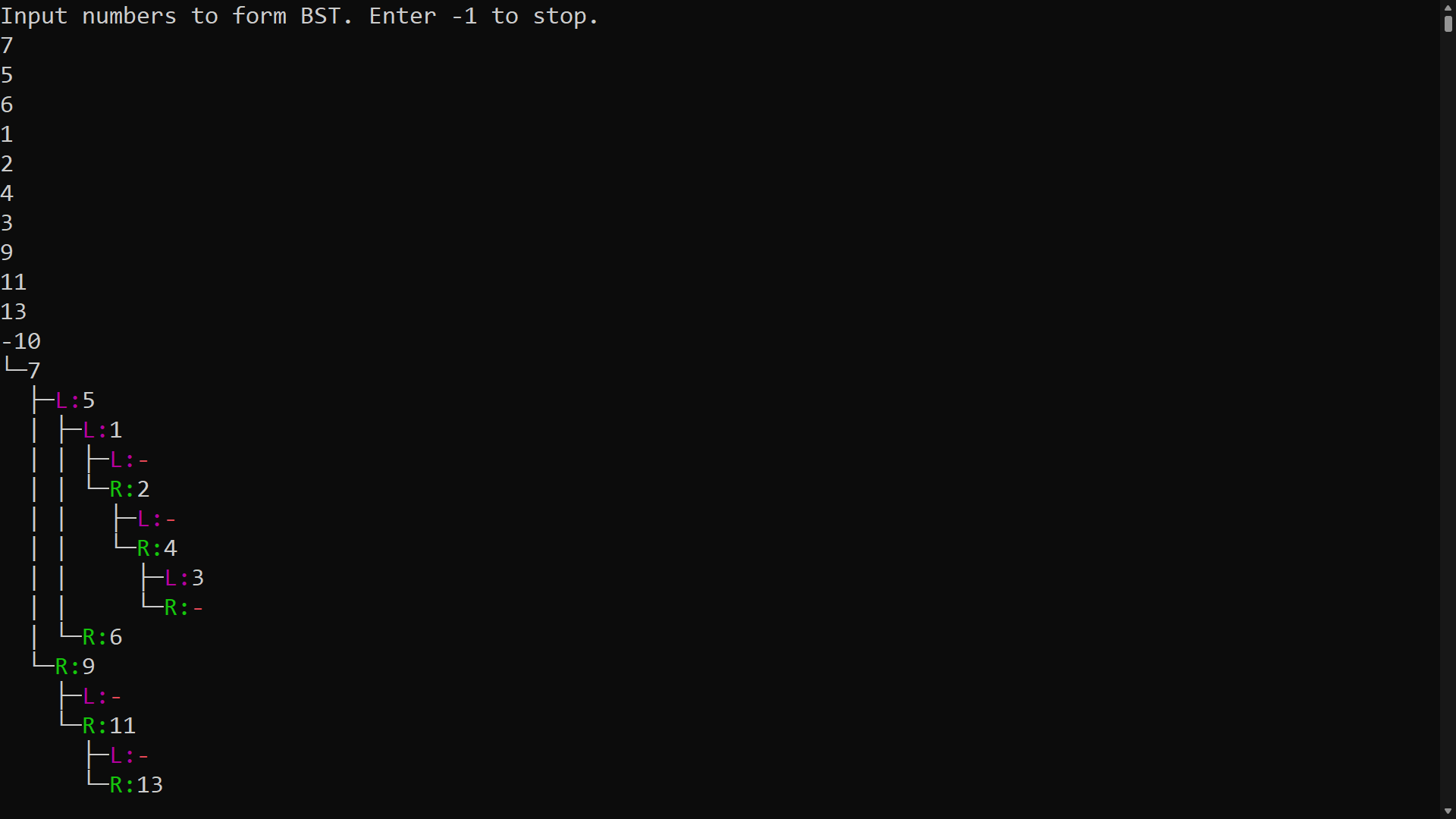
When entering the following numbers: 8,3,10,1,6,14,4,7,13. The BST output should be like the following:



**Actual Output:**



In this algorithm, the root (8) is outputted first followed by the left subtree(L : 3) and the right subtree (R : 10). This process is repeated for every node until all of the inputted numbers are outputted. (L : -) and (R : -) mean that it is an empty node.



Output when a negative number is entered. Execution stops and the BST is outputted with the previous numbers.

**Question 8**

**Code:**

#region 8

long question8 = 0;

do

{

Console.WriteLine("Input number to check approximation to the square root:");

question8 = Convert.ToInt64(Console.ReadLine());

if(question8 <= 0)

{

Console.WriteLine("Enter a number greater than 0!");

Console.WriteLine("--------------------------------------------------------");

}

} while (question8 <= 0);

Console.WriteLine($"The value of the root is: {Algorithms.NewtonsSquareRoot(question8)}");

Console.ReadLine();

#endregion

// Inspired by: https://www.geeksforgeeks.org/find-root-of-a-number-using-newtons-method/#:~:text=Let%20N%20be%20any%20number,to%20be%20N%20or%201.

public static double NewtonsSquareRoot(double num)

{

double EPSILON = 0.000000001;

// Assuming the sqrt of n as n only

double x = num;

// The closed guess will be stored in the root

double num\_root;

// To count the number of iterations

int count = 0;

while (true)

{

count++;

// Calculating more closed x

num\_root = 0.5 \* (x + (num / x));

// Checking for closeness

if (Math.Abs(num\_root - x) < EPSILON)

break;

// Updating the root

x = num\_root;

}

return num\_root;

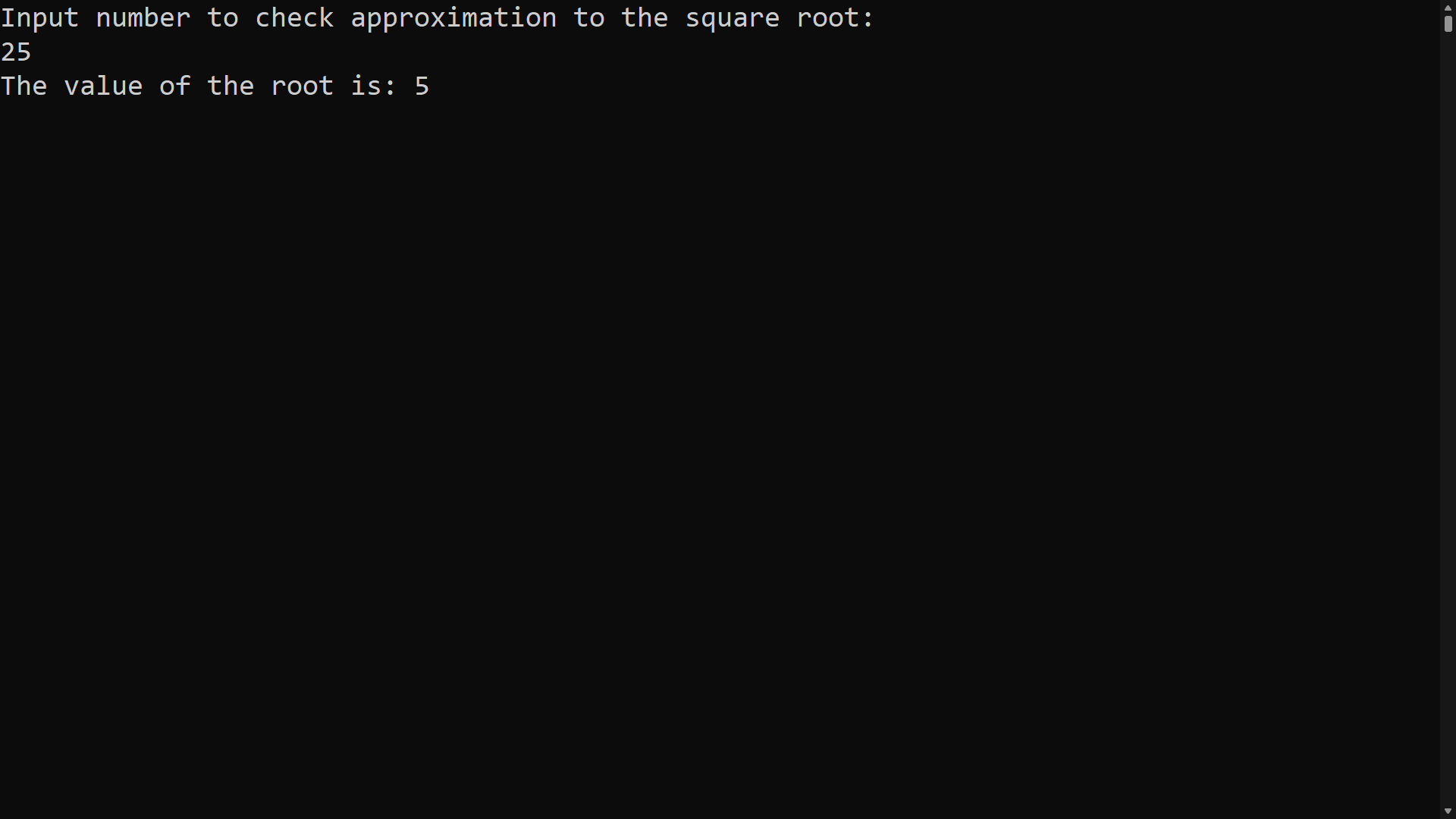
}

}

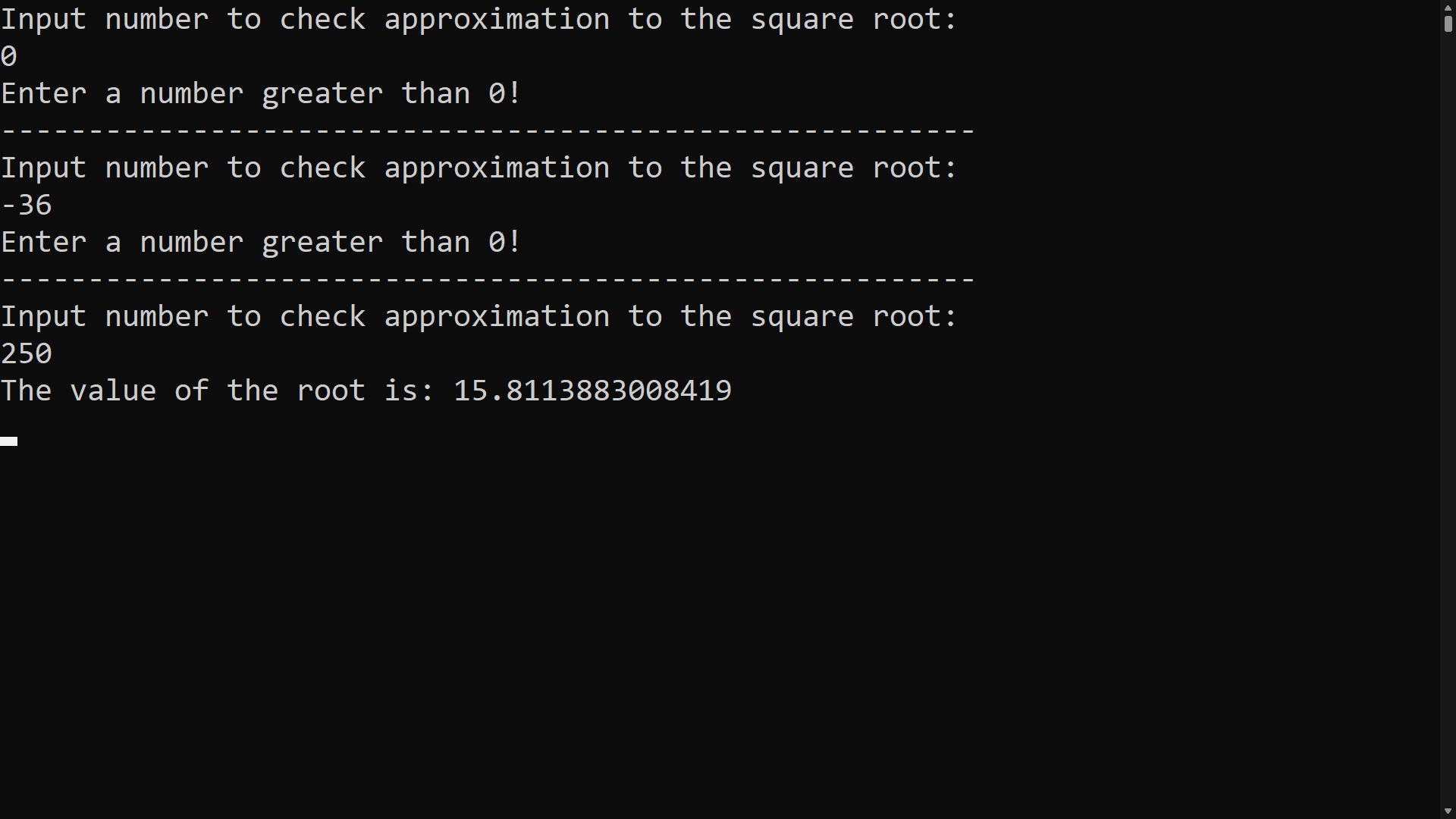
**Testing:**

This program is found in #region 8 of the *Program.cs* class. The function NewtonsSquareRoot() finds an approximation to the square root of the inputted number using the Newton-Raphson Method. This function can be found in the *Algorithms.cs* class. . Since you cannot find the square root of a negative number, a do, while loop was used for error checking. If a negative number is inputted, an error message will come up until the user enters a number that is greater than 0. Even though the square root of 0 is 0, while testing, when 0 was inputted the program would crash. Therefore, the do while loop was changed to not accept 0 or any negative numbers.

**Output:**



When the program is run, the user enters a number and the algorithm will output the root.



Error message if inputted number is 0 or a negative.

**Question 9**

**Code:**

#region 9

int question9 = 0;

List<int> question9Numbers = new List<int>();

Console.WriteLine("Input numbers and find repeated ones. Enter -1 to stop.");

while (question9 != -1)

{

question9 = Convert.ToInt32(Console.ReadLine());

if (question9 != -1)

{

question9Numbers.Add(question9);

}

}

Console.WriteLine($"Repeated numbers are: {string.Join(", ", Functions.FindDuplicates(question9Numbers.ToArray()).Distinct())}");

#endregion

public static List<int> FindDuplicates(int[] numbers)

{

List<int> duplicated = new List<int>();

Array.Sort(numbers);

for (int i = 1; i < numbers.Length; i++)

{

if (numbers[i] == numbers[i - 1])

{

duplicated.Add(numbers[i]);

}

}

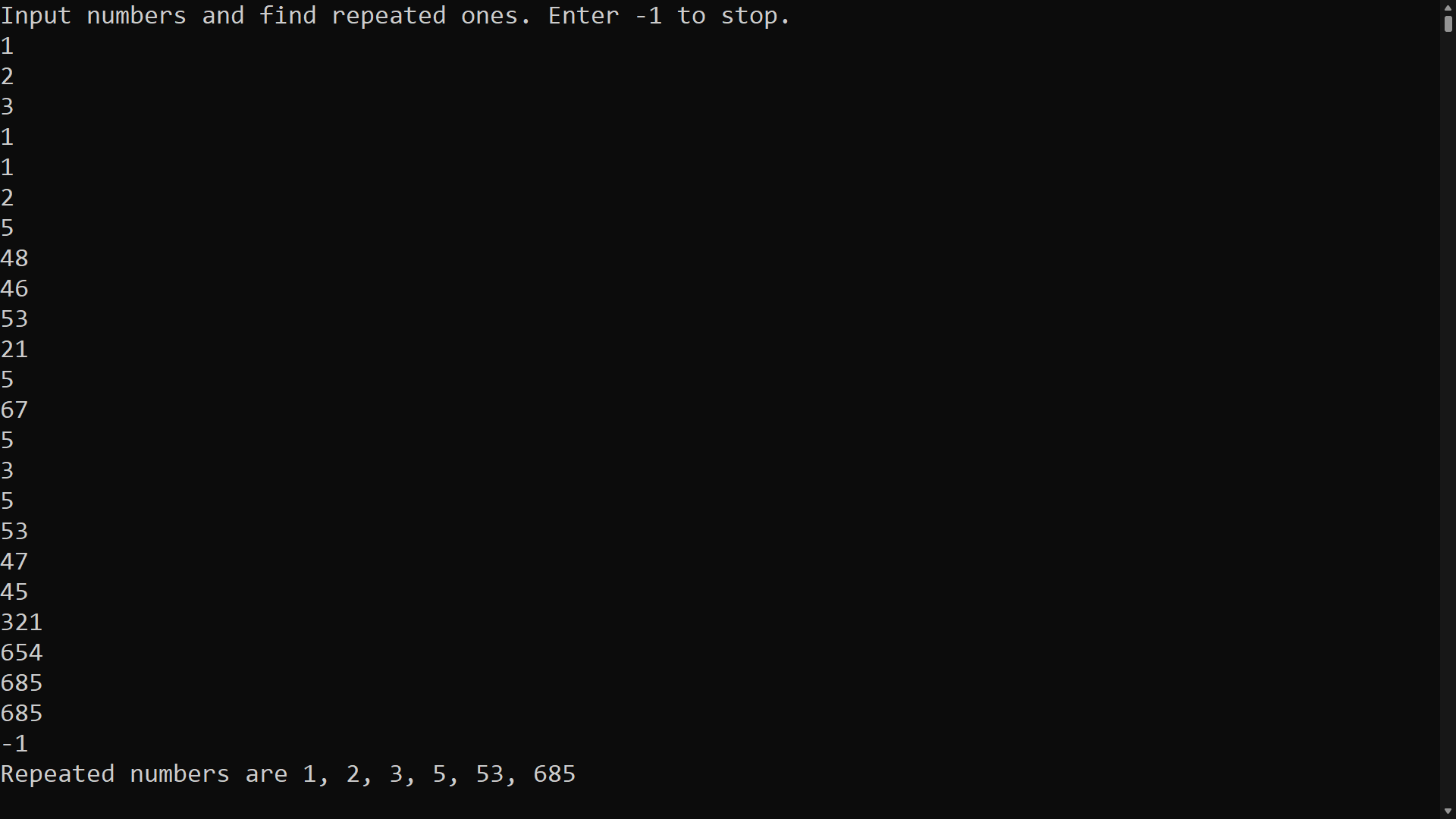
return duplicated;

}

**Testing:**

This program is found in #region 9 of the *Program.cs* class. This program ask the user to input numbers into a list and then will output all the integers that were repeated more than once. -1 can be entered into the list to stop the input. The FindDuplicates() function will compare all the numbers in the array and return all of the duplicates. This function can be found in the *Functions.cs* Class. For the program to be a fast and memory-efficient, this program runs in linear time.

**Output:**



When the program is run, the user inputs the numbers into the array and the algorithm will output all of the repeated integers.

**Question 10**

**Code:**

#region 10

int question10 = 0;

List<int> question10Numbers = new List<int>();

Console.WriteLine("Input numbers and find largest number. Enter -1 to stop.");

while (question10 != -1)

{

question10 = Convert.ToInt32(Console.ReadLine());

if (question10 != -1)

{

question10Numbers.Add(question10);

}

}

Console.WriteLine($"Largest number is: {Functions.FindMaximumNumber(question10Numbers.ToArray(), question10Numbers.ToArray().Length)}");

#endregion

public static int FindMaximumNumber(int[] A, int n)

{

// if size = 0 means whole array

// has been traversed

if (n == 1)

return A[0];

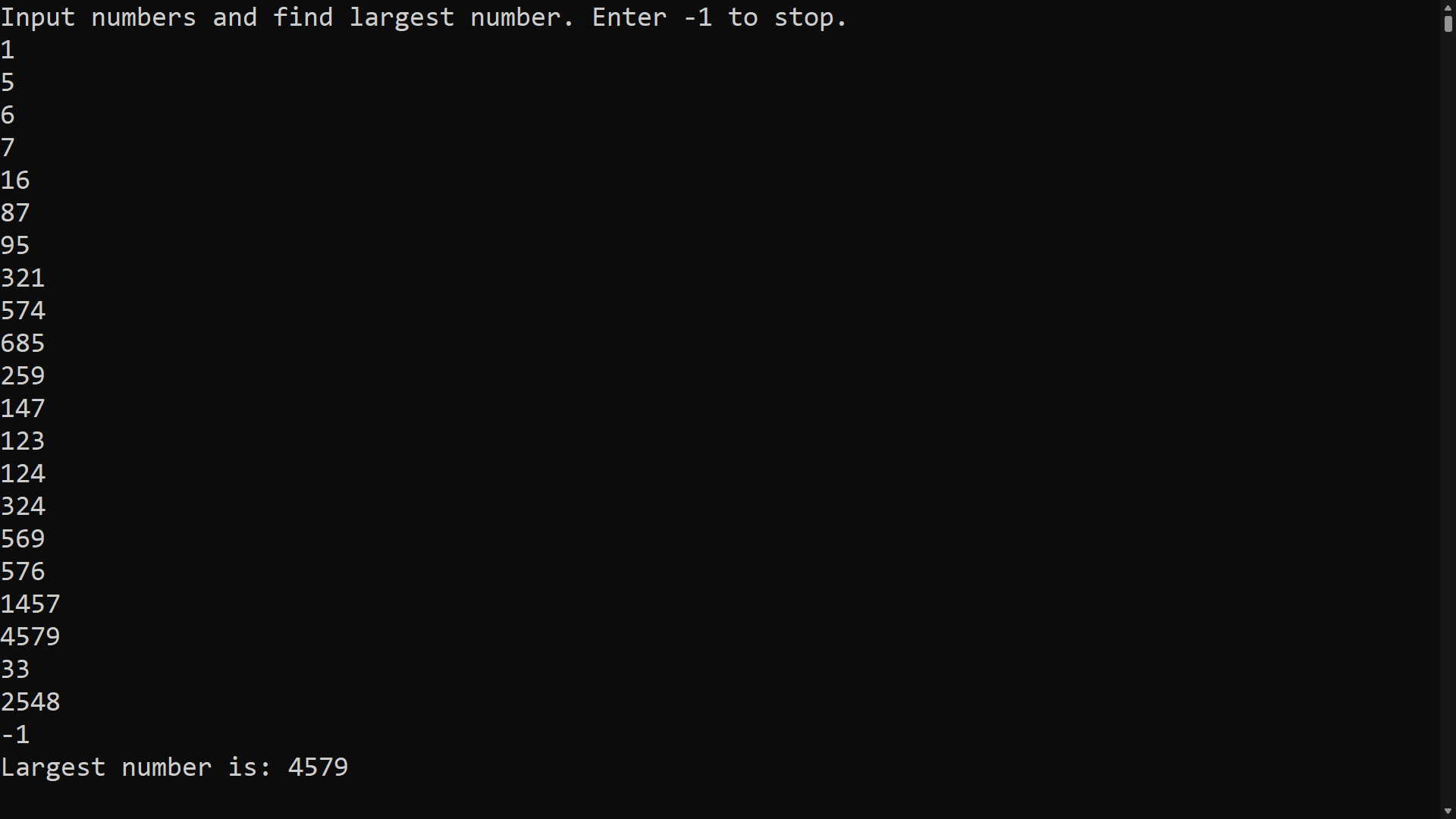
return Math.Max(A[n - 1], FindMaximumNumber(A, n - 1));

}

**Testing:**

This program is found in #region 10 of the *Program.cs* class. This program asks the user to input numbers to a list and uses recursion to find the largest number in the list. User is allowed to enter repeated numbers. The function FindMaximumNumber() found in the *Functions.cs* class find the maximum number in the list. To stop the input, just enter -1 into the input list.

**Output:**



When the program is run, the user inputs the numbers into the array and the algorithm will output the largest number.

**Question 11**

**Code:**

#region 11

Console.WriteLine("Input number to compute cosine or sine by taking the first n terms of the appropriate (Maclaurin?) series expansion");

int question11 = Convert.ToInt32(Console.ReadLine());

int factorial = Enumerable.Range(1, question11).Aggregate(1, (x, y) => x \* y);

Console.WriteLine($"Cos: {Functions.CalculateCos(factorial)}");

Console.WriteLine($"Sin: {Functions.CalculateSin(factorial)}");

#endregion

// Inspired by: https://www.geeksforgeeks.org/program-to-calculate-the-value-of-sinx-and-cosx-using-expansion/

public static float CalculateSin(float number)

{

float accuracy = (float)0.0001, denominator, sinx, sinval;

// Converting degrees to radian

number = number \* (float)(3.142 / 180.0);

float x1 = number;

// maps the sum along the series

sinx = number;

// holds the actual value of sin(n)

sinval = (float)Math.Sin(number);

int i = 1;

do

{

denominator = 2 \* i \* (2 \* i + 1);

x1 = -x1 \* number \* number / denominator;

sinx = sinx + x1;

i = i + 1;

} while (accuracy <= sinval - sinx);

return sinx;

}

public static float CalculateCos(float number)

{

float accuracy = (float)0.0001, x1, denominator, cosx, cosval;

// Converting degrees to radian

number = number \* (float)(3.142 / 180.0);

x1 = 1;

// maps the sum along the series

cosx = x1;

// holds the actual value of sin(n)

cosval = (float)Math.Cos(number);

int i = 1;

do

{

denominator = 2 \* i \* (2 \* i - 1);

x1 = -x1 \* number \* number / denominator;

cosx = cosx + x1;

i = i + 1;

} while (accuracy <= cosval - cosx);

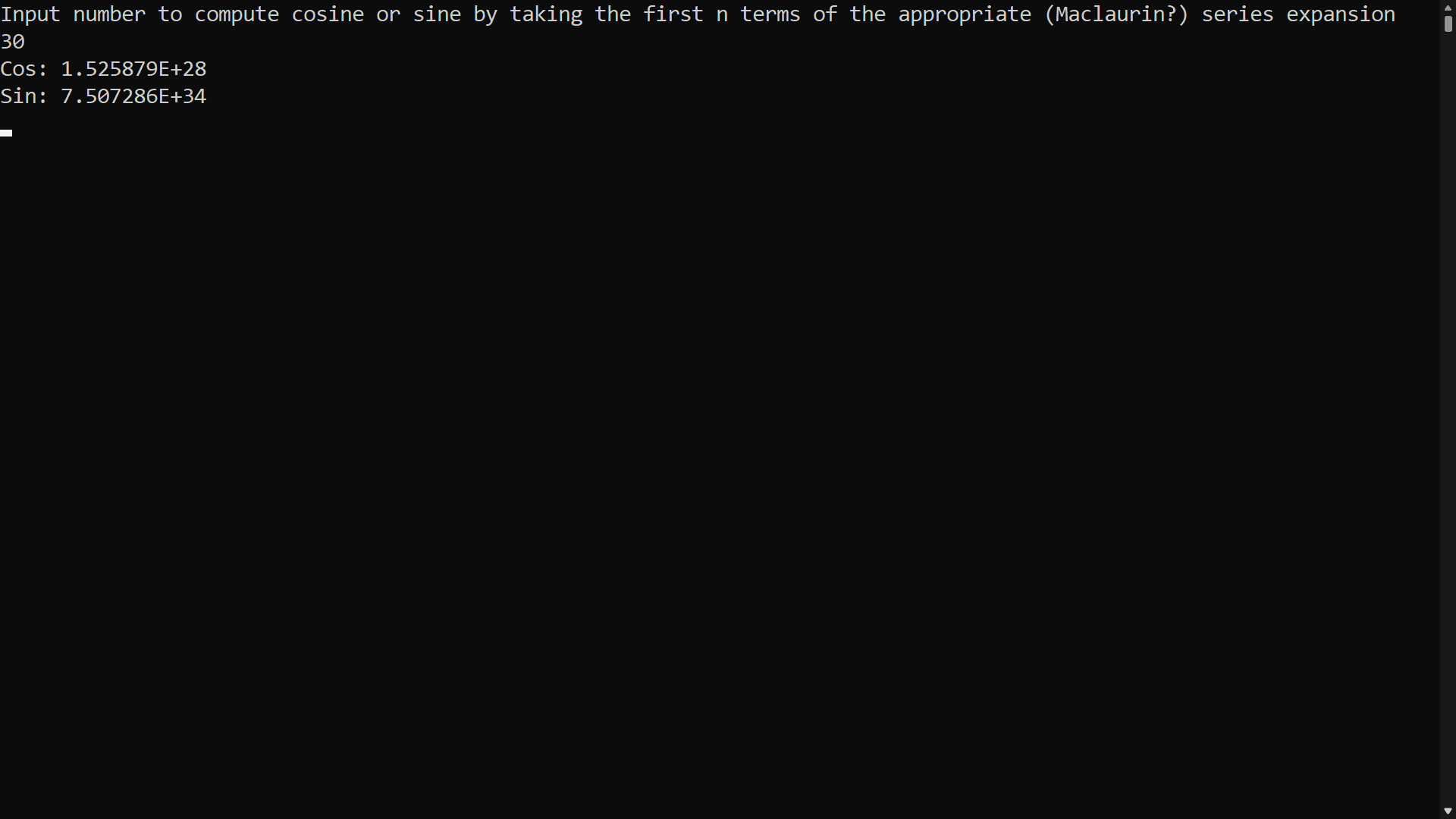
return cosx;

}

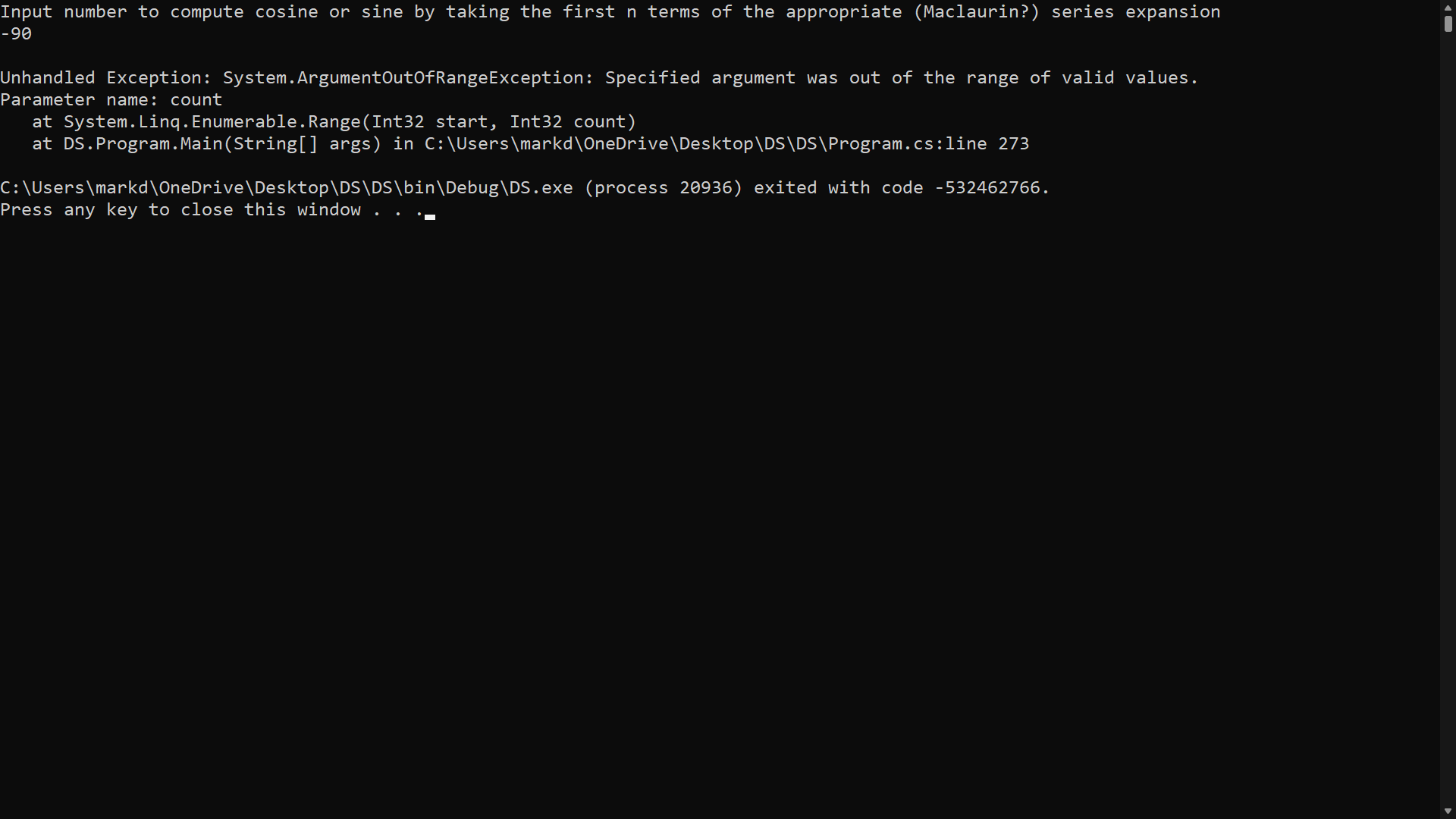
**Testing:**

This program is found in #region 11 of the *Program.cs* class. This program uses Maclaurin series expansion to compute the cosine and sine of an inputted number. Both the CalculateCos() and CalculateSin() functions can be found in the F*unctions.cs* class. When testing was being sone on this program the following problems occurred. When the user enters a negative number the program will crash. The final answer of the cosine and sine were not always correct (checked with answer from calculator).

**Output:**



Output when the user inputs the integer 30.



Output when the user inputs a negative number.

**Question 12**

**Code:**

#region 12

int question12 = -1;

do {

Console.WriteLine("Input number to find the sum of the first n numbers using the Fibonacci sequence.");

question12 = Convert.ToInt32(Console.ReadLine());

// validating input

if (question12 < 0){

Console.WriteLine("Enter number larger than 0!");

Console.WriteLine("------------------------------------------------------------------------------------");

}

}while (question12 < 0);

Console.WriteLine($"Fibonacci sum: {Functions.FibonacciSequenceSum(question12)}");

#endregion

// Inspired by: https://www.geeksforgeeks.org/sum-fibonacci-numbers/

public static int FibonacciSequenceSum(int n)

{

if (n <= 0)

{

return 0;

}

int[] fibo = new int[n + 1];

fibo[0] = 0; fibo[1] = 1;

// Initialize result

int sum = fibo[0] + fibo[1];

Console.WriteLine($"{fibo[0]} + {fibo[1]} = {sum}");

// Add remaining terms

for (int i = 2; i <= n; i++)

{

Console.WriteLine($"Term {i}");

fibo[i] = fibo[i - 1] + fibo[i - 2];

Console.WriteLine($"Term total = {fibo[i - 1]} + {fibo[i - 2]} = {fibo[i]}");

sum += fibo[i];

Console.WriteLine($"Sum = {sum} + {fibo[i]} = {sum}");

Console.WriteLine("--------------------------------------------------------");

}

return sum;

}

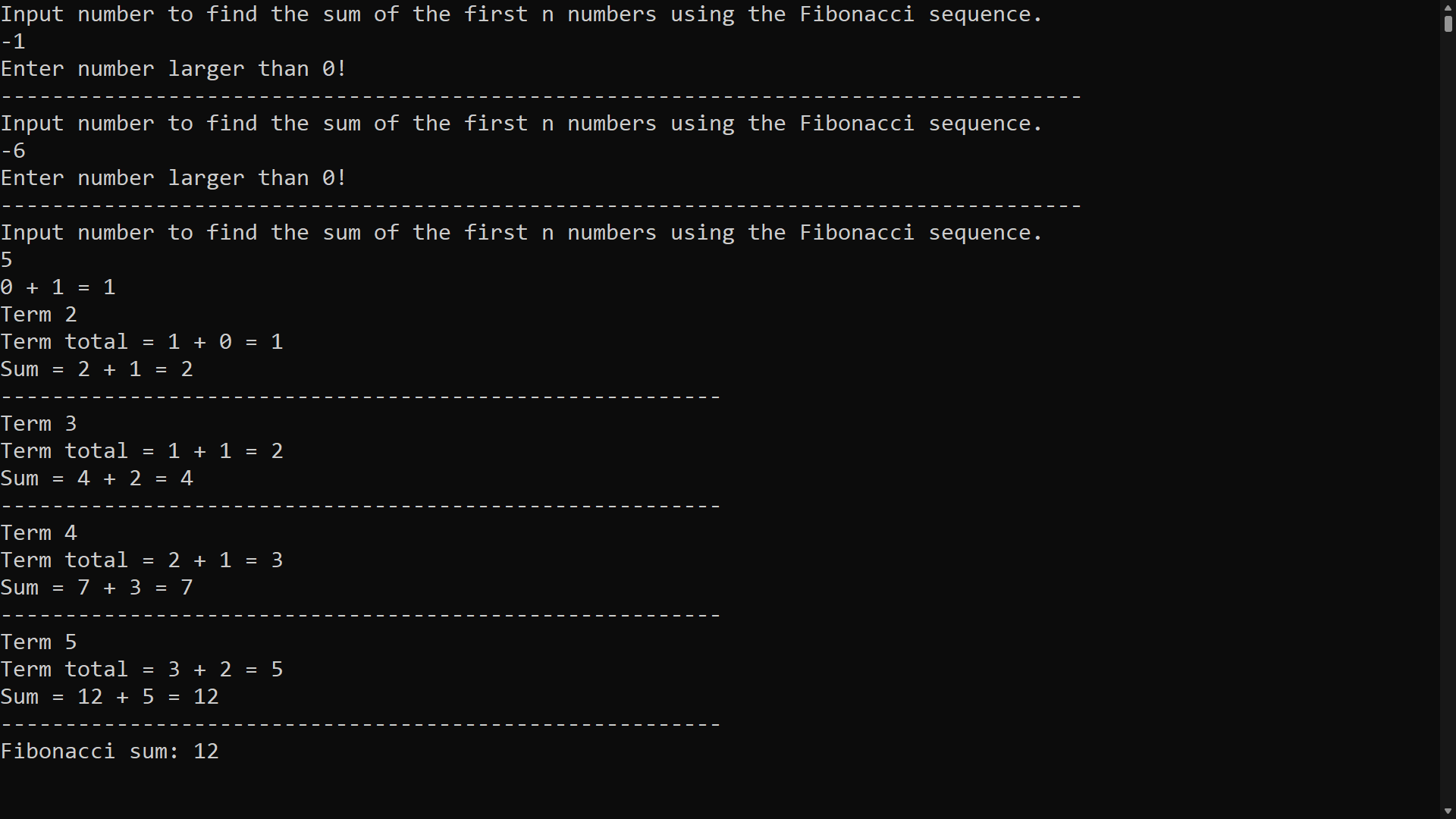
**Testing:**

This program is found in #region 12 of the *Program.cs* class. This program asks the user to input number to find the sum of the first n numbers of the Fibonacci sequence. The sum of the first n numbers using the Fibonacci sequence cannot be calculated for negative numbers, therefore, if the user enters a negative number, an error message will be will be outputted until the user enters a number that is 0 or not negative. The function FibonacciSequenceSum(), found in the *Functions.cs* Class, is used to find the sum of the first n numbers using Fibonacci. The function will also output all the equations that make up the final sum of the Fibonacci sequence.

**Output:**



When the program is run, the user is asked to enter a number to find the sum of the first n numbers using the Fibonacci Sequence. All the equations and the final sum is outputted.



Error message if user enters a negative number.

**References**

* **Question 1 –** none.
* **Question 2 –** none.
* **Question 3 –** <https://stackoverflow.com/questions/67760895/c-finding-the-extreme-points-of-an-algorithm>
* **Question** **4 -** <https://www.geeksforgeeks.org/find-all-pairs-possible-from-the-given-array/>
* **Question 5** **-** <https://iq.opengenus.org/arithmetic-expression-evaluation-using-stack/> & https://www.geeksforgeeks.org/print-stack-elements-from-top-to-bottom/
* **Question 6 -** <https://www.geeksforgeeks.org/sieve-of-eratosthenes/>
* **Question 7 -** <https://stackoverflow.com/questions/36311991/c-sharp-display-a-binary-search-tree-in-console>
* **Question 8 -** <https://www.geeksforgeeks.org/find-root-of-a-number-using-newtons->
* **Question 9 –** none.
* **Question** **10 –** none.
* **Question 11 –** <https://www.geeksforgeeks.org/program-to-calculate-the-value-of-sinx-and-cosx-using-expansion/>
* **Question 12 –** <https://www.geeksforgeeks.org/sum-fibonacci-numbers/>