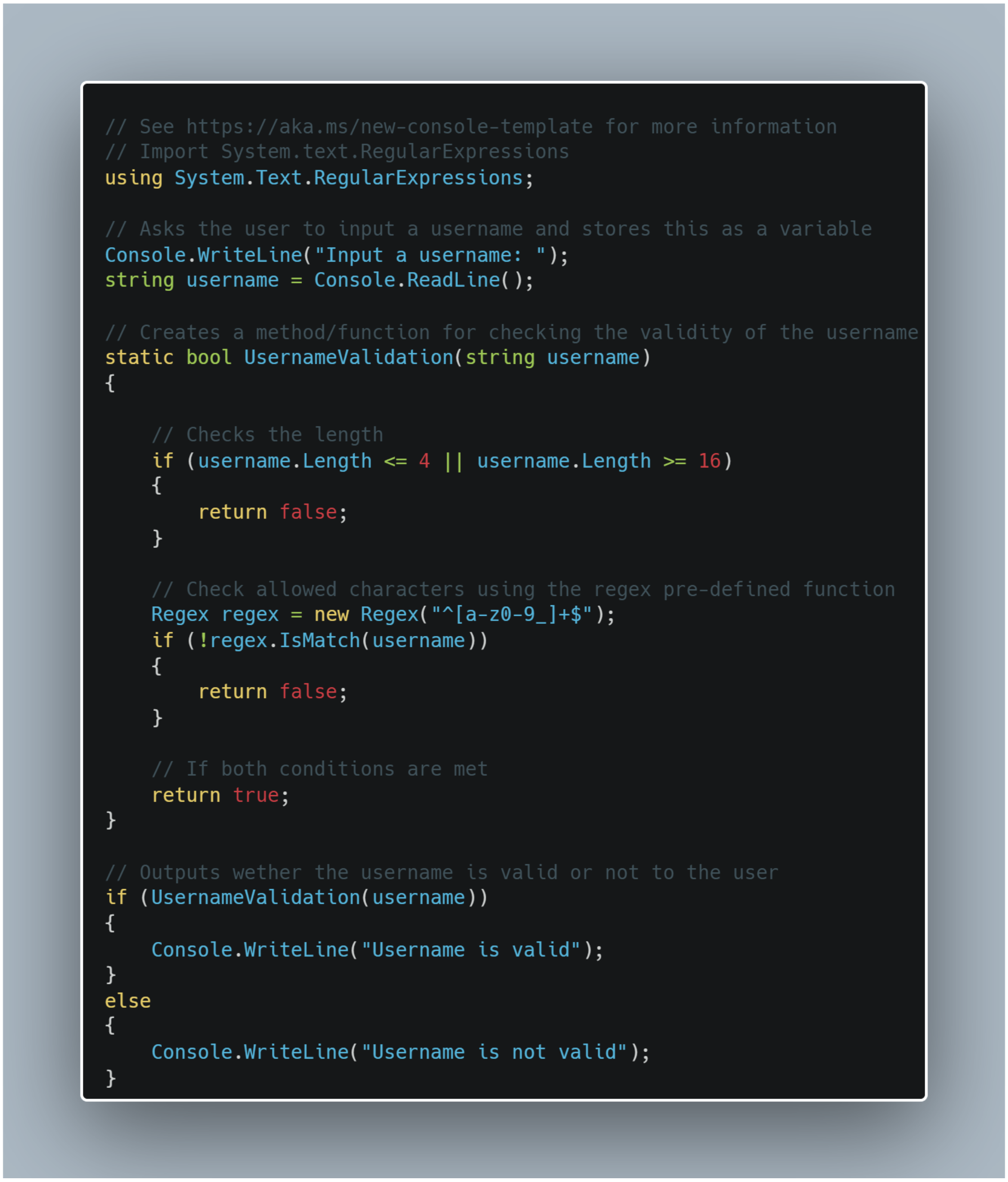
SCDV41 – Programming & Software Fundamentals: Assignment 1

Program 1: Solution 1

Decomposition

1. Import ‘System.Text.RegularExpressions’ to allow for the RegEx pre-defined function to be used to validate the username
2. Prompt the user to input their username
3. Store username in a ‘username’ variable
4. Make a method to validate the username
5. Create an IF statement to check that the username is between 4 and 16 characters long
6. Create an IF statement to check that the username only includes valid characters using the RegEx pre-defined function
7. Outputs if the username is valid or invalid to the user

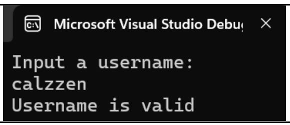
Evidence

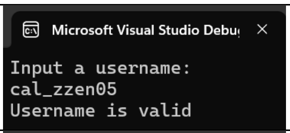
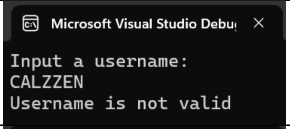
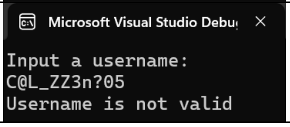


```csharp
// See https://aka.ms/new-console-template for more information
// Import System.text.RegularExpressions
using System.Text.RegularExpressions;

// Asks the user to input a username and stores this as a variable
Console.WriteLine("Input a username: ");
string username = Console.ReadLine();

// Creates a method/function for checking the validity of the username
static bool UsernameValidation(string username)
{

    // Checks the length
    if (username.Length <= 4 || username.Length >= 16)
    {
        return false;
    }

    // Check allowed characters using the regex pre-defined function
    Regex regex = new Regex("^[a-z0-9_]+$");
    if (!regex.IsMatch(username))
    {
        return false;
    }

    // If both conditions are met
    return true;
}

// Outputs wether the username is valid or not to the user
if (UsernameValidation(username))
{
    Console.WriteLine("Username is valid");
}
else
{
    Console.WriteLine("Username is not valid");
}
```

Figure 1 - Program 1: Solution 1 evidence of code

Testing

Test Data	Expected Outcome	Result
calzzen	Username is valid	 Input a username: calzzen Username is valid

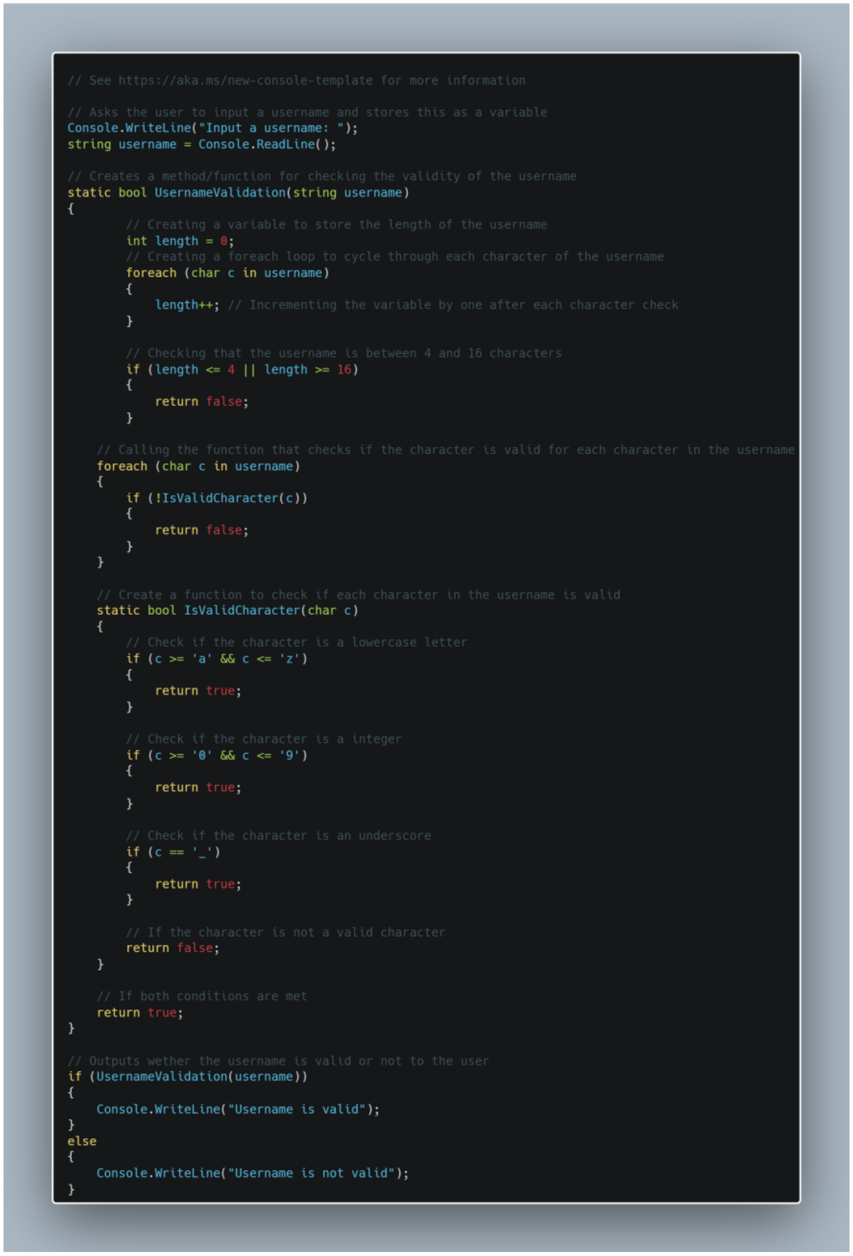
cal_zzen05	Username is valid	 Input a username: cal_zzen05 Username is valid
CALZZEN	Username is not valid	 Input a username: CALZZEN Username is not valid
C@l_ZZ3n?05	Username is not valid	 Input a username: C@L_ZZ3n?05 Username is not valid

Program 1: Solution 2

Decomposition

1. Prompt the user to input their username
2. Store username in a ‘username’ variable
3. Make a method to validate the username
4. Create an IF statement to check that the username is between 4 and 16 characters long
5. Create a variable that stores the length of the username
6. Create a foreach loop that increments the length variable for each character
7. Checks if the length is between 4 and 16
8. Call the method to check if each character is valid
9. Create the method that validates each character in the username
10. Use a foreach loop to validate if each character is a lowercase letter
11. Use a foreach loop to check if each character is a number
12. Use a foreach loop to validate if each character is an underscore
13. Output the outcome to the user

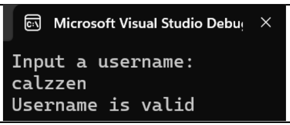
Evidence

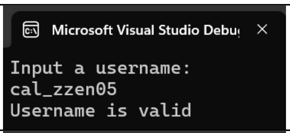
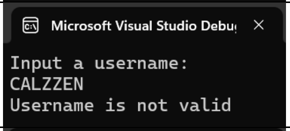
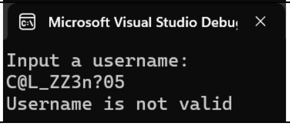


```
// See https://aka.ms/new-console-template for more information

// Asks the user to input a username and stores this as a variable
Console.WriteLine("Input a username: ");
string username = Console.ReadLine();

// Creates a method/function for checking the validity of the username
static bool UsernameValidation(string username)
{
        // Creating a variable to store the length of the username
        int length = 0;
        // Creating a foreach loop to cycle through each character of the username
        foreach (char c in username)
        {
            length++; // Incrementing the variable by one after each character check
        }

        // Checking that the username is between 4 and 16 characters
        if (length <= 4 || length >= 16)
        {
            return false;
        }

    // Calling the function that checks if the character is valid for each character in the username
    foreach (char c in username)
    {
        if (!IsValidCharacter(c))
        {
            return false;
        }
    }

    // Create a function to check if each character in the username is valid
    static bool IsValidCharacter(char c)
    {
        // Check if the character is a lowercase letter
        if (c >= 'a' && c <= 'z')
        {
            return true;
        }

        // Check if the character is a integer
        if (c >= '0' && c <= '9')
        {
            return true;
        }

        // Check if the character is an underscore
        if (c == '_')
        {
            return true;
        }

        // If the character is not a valid character
        return false;
    }

    // If both conditions are met
    return true;
}

// Outputs wether the username is valid or not to the user
if (UsernameValidation(username))
{
    Console.WriteLine("Username is valid");
}
else
{
    Console.WriteLine("Username is not valid");
}
```

Figure 2 – Program 1: Solution 2

Testing

Test Data	Expected Outcome	Result
calzzen	Username is valid	 Input a username: calzzen Username is valid

cal_zzen05	Username is valid	 Input a username: cal_zzen05 Username is valid
CALZZEN	Username is not valid	 Input a username: CALZZEN Username is not valid
C@l_ZZ3n?05	Username is not valid	 Input a username: C@L_ZZ3n?05 Username is not valid

Program 1 Solution Comparison and Evaluation

For program 1, the difference between solution 1 and solution 2 is how they validate the characters in the username. The use of Regular Expressions (RegEx) in solution 1 allows for the source code to be concise and highly readable, as the characters within the username can be validated in a single operation. This clear structure helps developers quickly understand the logic, allowing for easier modifications and debugging. This would be particularly beneficial when debugging a program. Conversely, solution 2 uses a series of FOR loops, resulting in longer, more complex code that could be harder to maintain and expand without further sacrifices to readability and performance. However, solution 2 can be more understandable to beginner C# developers, making it ideal for those learning the language. Solution 1 is more efficient than solution 2, as it operates with a time complexity of O(n) (GeeksforGeeks, 2018), where solution 2 might have a higher time complexity due to the multiple iterations of the FOR loops. This will allow for solution 1 to outperform solution 2, particularly if long usernames are inputted by the user.

Overall, I feel that solution 1 is more suitable due to its increased readability, high efficiency and higher performance.

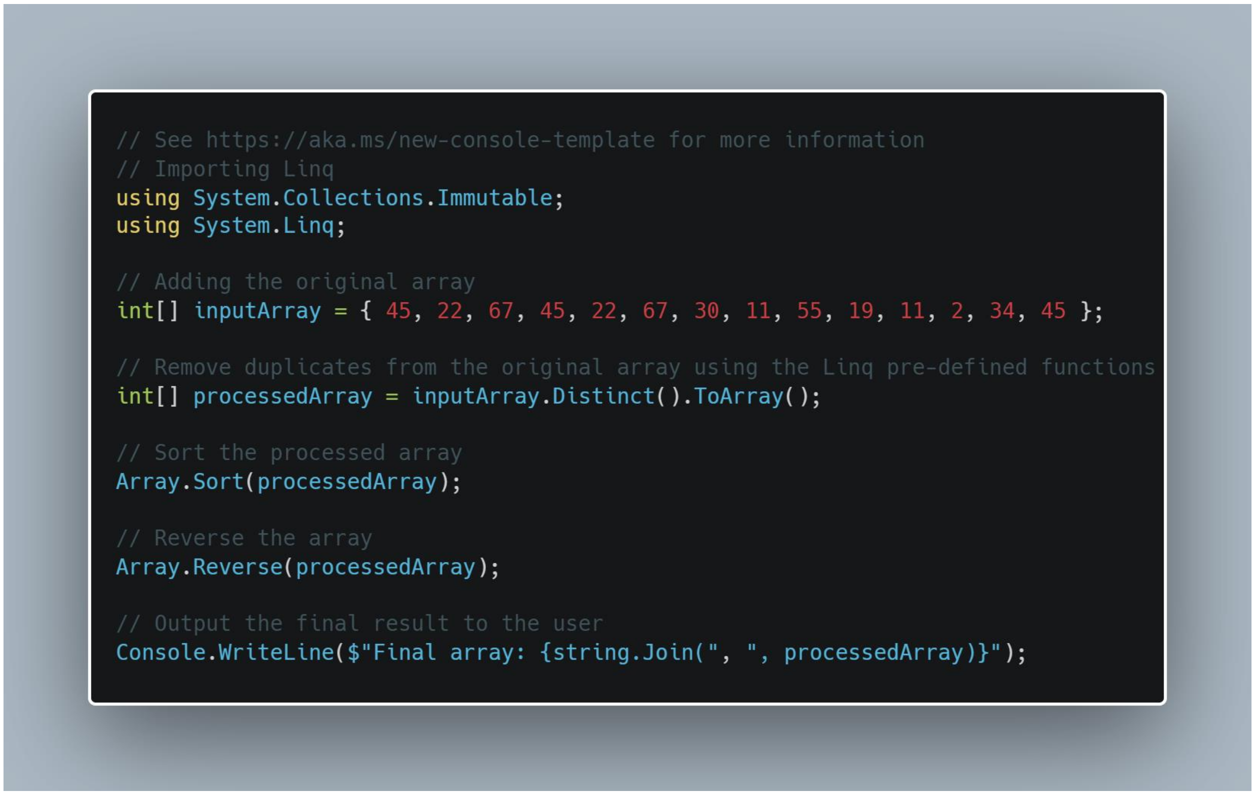
Program 2: Solution 1

Decomposition

1. Import ‘Linq’
2. Create an array to store the values before it is processed
3. Create a new array to store the processed array values into

4. Use the Distinct() function to remove duplicates from the array
5. Use the Sort() function to order the array in ascending order
6. Use the Reverse() function to reverse the array
7. Output the array to the user

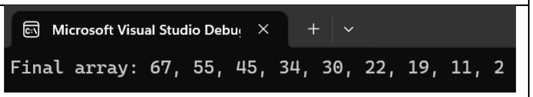
Evidence



```csharp
// See https://aka.ms/new-console-template for more information
// Importing Linq
using System.Collections.Immutable;
using System.Linq;

// Adding the original array
int[] inputArray = { 45, 22, 67, 45, 22, 67, 30, 11, 55, 19, 11, 2, 34, 45 };

// Remove duplicates from the original array using the Linq pre-defined functions
int[] processedArray = inputArray.Distinct().ToArray();

// Sort the processed array
Array.Sort(processedArray);

// Reverse the array
Array.Reverse(processedArray);

// Output the final result to the user
Console.WriteLine($"Final array: {string.Join(", ", processedArray)}");
```

Figure 3 – Program 2: Solution 1

Testing

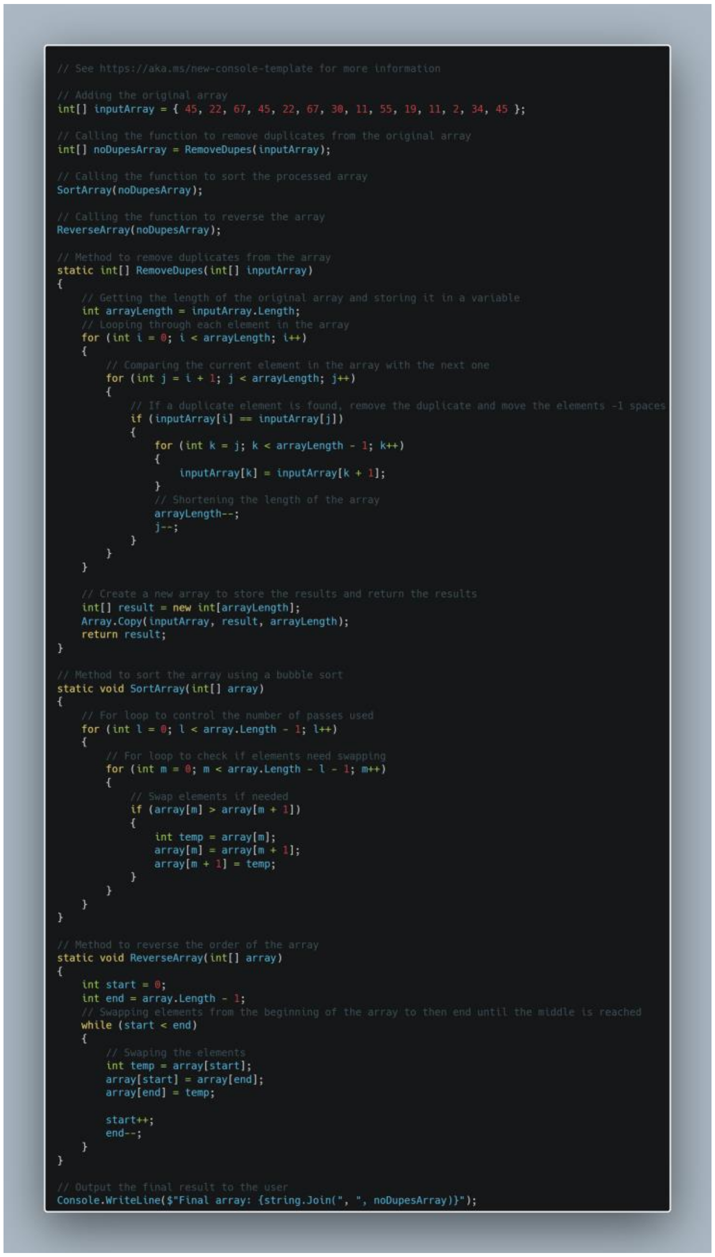
Test Data	Expected Outcome	Result
45, 22, 67, 45, 22, 67, 30, 11, 55, 19, 11, 2, 34, 45	67, 55, 45, 34, 30, 22, 19, 11, 2	 `Final array: 67, 55, 45, 34, 30, 22, 19, 11, 2`

Program 2: Solution 2

Decomposition

1. Create an array to store the values before it is processed
2. Create a new array to store the processed array values into and call a method to remove the duplicate values from the array
3. Create the method to remove duplicates
4. Get the length of the array and store it in a variable
5. Use a FOR loop to compare the current element with the next one in the array
6. Remove the duplicate value from the array, and shorten the array length
7. Return the results
8. Call a method to sort the array
9. Create a method to sort the array using a bubble sort
10. Call a method to reverse the array
11. Create the method to reverse the array
12. Swap element from the beginning of the array to the end, until the middle is reached
13. Output the processed array to the user

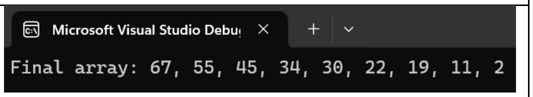
Evidence



```
// See https://aka.ms/new-console-template for more information

// Adding the original array
int[] inputArray = { 45, 22, 67, 45, 22, 67, 30, 11, 55, 19, 11, 2, 34, 45 };

// Calling the function to remove duplicates from the original array
int[] noDupesArray = RemoveDupes(inputArray);

// Calling the function to sort the processed array
SortArray(noDupesArray);

// Calling the function to reverse the array
ReverseArray(noDupesArray);

// Method to remove duplicates from the array
static int[] RemoveDupes(int[] inputArray)
{
    // Getting the length of the original array and storing it in a variable
    int arrayLength = inputArray.Length;
    // Looping through each element in the array
    for (int i = 0; i < arrayLength; i++)
    {
        // Comparing the current element in the array with the next one
        for (int j = i + 1; j < arrayLength; j++)
        {
            // If a duplicate element is found, remove the duplicate and move the elements -1 spaces
            if (inputArray[i] == inputArray[j])
            {
                for (int k = j; k < arrayLength - 1; k++)
                {
                    inputArray[k] = inputArray[k + 1];
                }
                // Shortening the length of the array
                arrayLength--;
                j--;
            }
        }
    }

    // Create a new array to store the results and return the results
    int[] result = new int[arrayLength];
    Array.Copy(inputArray, result, arrayLength);
    return result;
}

// Method to sort the array using a bubble sort
static void SortArray(int[] array)
{
    // For loop to control the number of passes used
    for (int l = 0; l < array.Length - 1; l++)
    {
        // For loop to check if elements need swapping
        for (int m = 0; m < array.Length - l - 1; m++)
        {
            // Swap elements if needed
            if (array[m] > array[m + 1])
            {
                int temp = array[m];
                array[m] = array[m + 1];
                array[m + 1] = temp;
            }
        }
    }
}

// Method to reverse the order of the array
static void ReverseArray(int[] array)
{
    int start = 0;
    int end = array.Length - 1;
    // Swapping elements from the beginning of the array to then end until the middle is reached
    while (start < end)
    {
        // Swaping the elements
        int temp = array[start];
        array[start] = array[end];
        array[end] = temp;

        start++;
        end--;
    }
}

// Output the final result to the user
Console.WriteLine($"Final array: {string.Join(", ", noDupesArray)}");
```

Figure 4 – Program 2: Solution 2

Testing

Test Data	Expected Outcome	Result
45, 22, 67, 45, 22, 67, 30, 11, 55, 19, 11, 2, 34, 45	67, 55, 45, 34, 30, 22, 19, 11, 2	 `Final array: 67, 55, 45, 34, 30, 22, 19, 11, 2`

Program 2 Solutions Comparison and Evaluation

The difference between solution 1 and 2 for program 2 is the method used to process the array. Solution 1 uses the built-in 'LINQ' functions (Distinct(), Sort() and Reverse()) to process the array concisely and efficiently, allowing for fewer lines of code to be used to perform the same tasks as solution 2. Using these functions benefits the readability and maintainability of solution 1, whereas solution 2 uses programmed bubble sorts which require more code. The use of LINQ functions in solution 1 offers performance benefits due to their optimised implementation. For example, the Sort() function operates with a time complexity of O(n log n) (GeeksforGeeks, 2018), whereas the manual bubble sort in solution 2 is far less efficient, with a time complexity of O(n^2). This will allow solution 1 to outperform solution 2 in execution speed, especially if larger arrays will be used in the future.

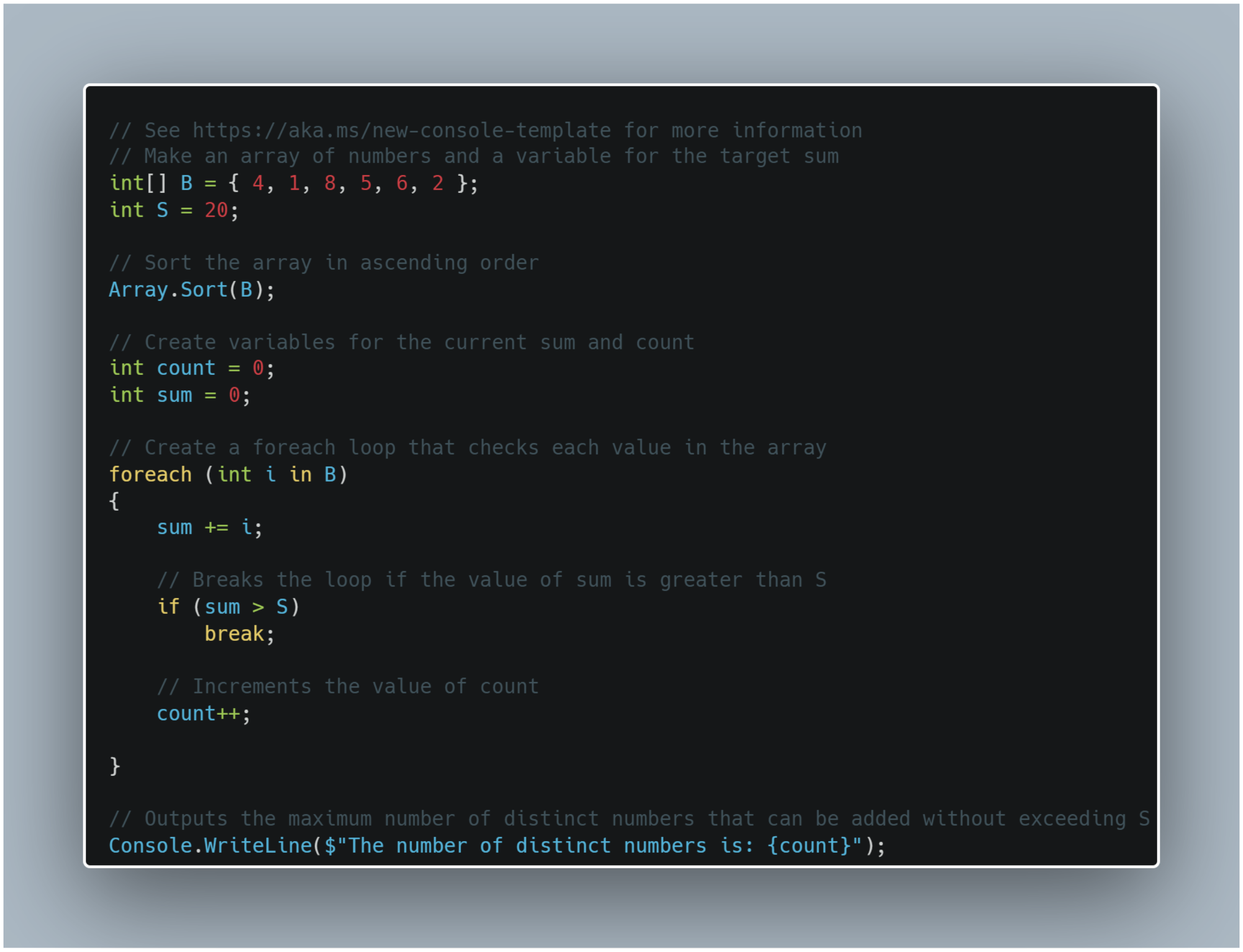
Overall, I feel that the ease of implementation and high efficiency of the LINQ functions make solution 1 the preferable option compared to solution 2.

Program 3: Solution 1

Decomposition

1. Create an array to store the numbers
2. Create a variable to store the target sum
3. Sort the array using the Sort() pre-defined function
4. Create a variable to store the sum of the numbers
5. Create a variable to store the number of distinct numbers
6. Create a foreach loop that increments for every value in the array until sum is greater than the target sum
7. Output the number of distinct numbers that can be added before the target sum is exceeded

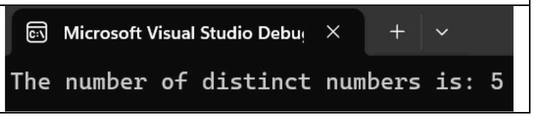
Evidence



```
// See https://aka.ms/new-console-template for more information
// Make an array of numbers and a variable for the target sum
int[] B = { 4, 1, 8, 5, 6, 2 };
int S = 20;

// Sort the array in ascending order
Array.Sort(B);

// Create variables for the current sum and count
int count = 0;
int sum = 0;

// Create a foreach loop that checks each value in the array
foreach (int i in B)
{
    sum += i;

    // Breaks the loop if the value of sum is greater than S
    if (sum > S)
        break;

    // Increments the value of count
    count++;

}

// Outputs the maximum number of distinct numbers that can be added without exceeding S
Console.WriteLine($"The number of distinct numbers is: {count}");
```

Figure 5 – Program 3: Solution 1

Testing

Test Data	Expected Outcome	Result
4, 1, 8, 5, 6, 2	The number of distinct numbers is: 5	 The number of distinct numbers is: 5

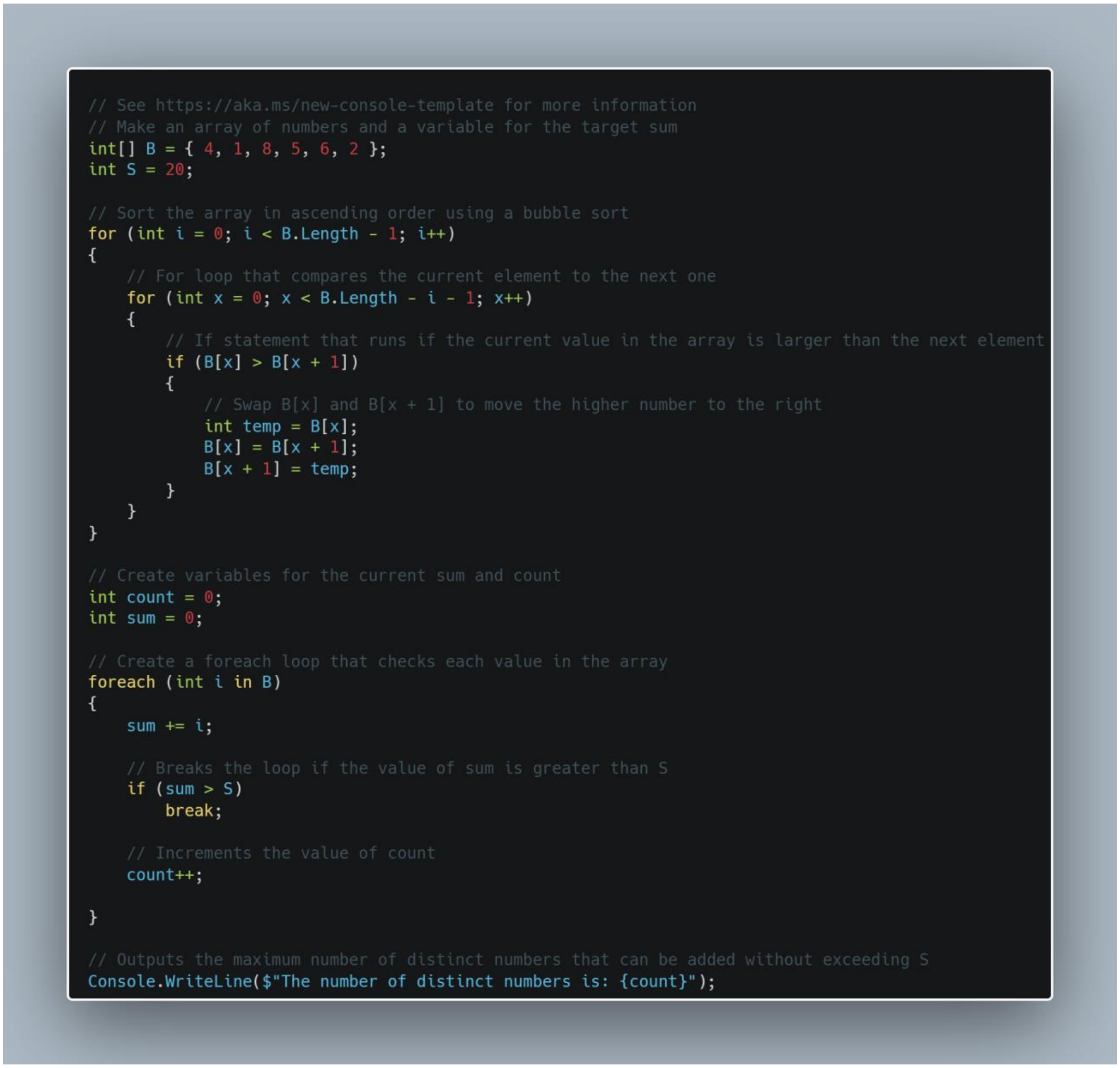
Program 3: Solution 2

Decomposition

1. Create an array to store the numbers
2. Create a variable to store the target sum

3. Sort the array using a bubble sort
4. Create a variable to store the sum of the numbers
5. Create a variable to store the number of distinct numbers
6. Create a foreach loop that increments for every value in the array until sum is greater than the target sum
7. Output the number of distinct numbers that can be added before the target sum is exceeded

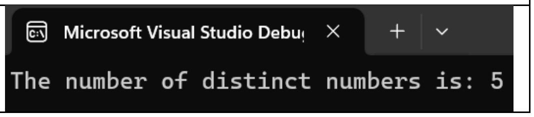
Evidence



```
// See https://aka.ms/new-console-template for more information
// Make an array of numbers and a variable for the target sum
int[] B = { 4, 1, 8, 5, 6, 2 };
int S = 20;

// Sort the array in ascending order using a bubble sort
for (int i = 0; i < B.Length - 1; i++)
{
    // For loop that compares the current element to the next one
    for (int x = 0; x < B.Length - i - 1; x++)
    {
        // If statement that runs if the current value in the array is larger than the next element
        if (B[x] > B[x + 1])
        {
            // Swap B[x] and B[x + 1] to move the higher number to the right
            int temp = B[x];
            B[x] = B[x + 1];
            B[x + 1] = temp;
        }
    }
}

// Create variables for the current sum and count
int count = 0;
int sum = 0;

// Create a foreach loop that checks each value in the array
foreach (int i in B)
{
    sum += i;

    // Breaks the loop if the value of sum is greater than S
    if (sum > S)
        break;

    // Increments the value of count
    count++;

}

// Outputs the maximum number of distinct numbers that can be added without exceeding S
Console.WriteLine($"The number of distinct numbers is: {count}");
```

Figure 6 – Program 3: Solution 2

Testing

Test Data	Expected Outcome	Result
4, 1, 8, 5, 6, 2	The number of distinct numbers is: 5	 The number of distinct numbers is: 5

Program 3 Comparisons and Evaluation

In solution 1 for this program, the Sort() function is used. This ensures that the array 'B' is sorted efficiently in a single line of code, eliminating the need for any manual sorting logic. This allows the code to remain concise compared to solution 2, which relies on a bubble sort to order the array. While the bubble sort produces the same result as the Sort() function, it is significantly less efficient, a difference that becomes more pronounced as the size of the array increases as it has a time complexity of $O(n^2)$ (GeeksforGeeks, 2018). The sorting method in solution 1 typically has a time complexity of O(n log n), making it more efficient and allowing it to outperform solution 2, particularly for larger arrays. As solution 1's code is a singular line, it is much easier to read and understand for developers, allowing for increased readability and scalability in the future compared to the longer and more complex second solution.

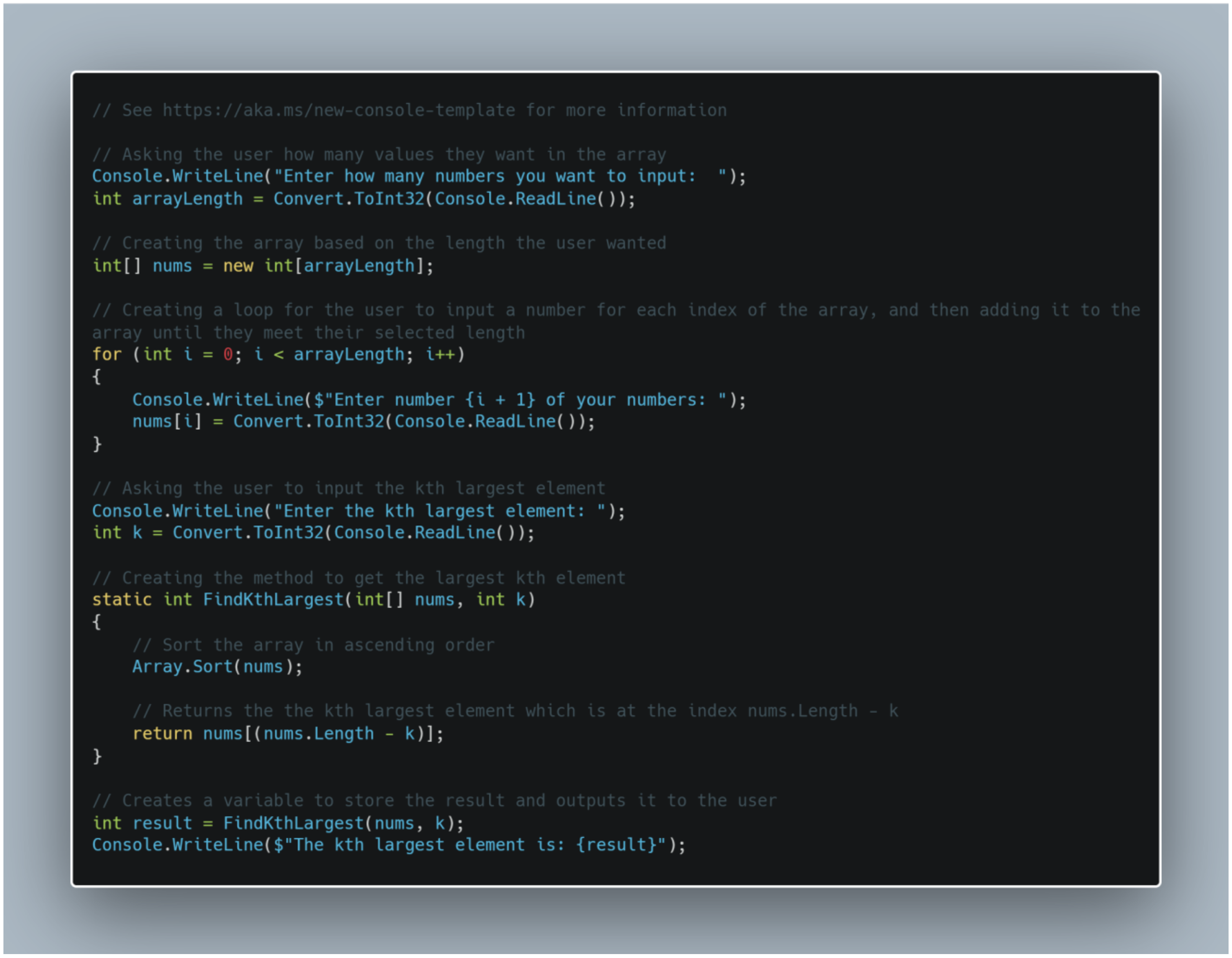
Overall, I feel that the concise and easily readable code of solution 1 is superior to solution 2. The optimisation behind the Sort() function also allows solution 1 to outperform solution 2, further making it the more favourable option.

Program 4: Solution 1

Decomposition

1. Ask the user to input the size of array they require
2. Create an array based on the size inputted
3. Create a FOR loop to ask the user to input a number for each index value in the array
4. Ask the user to input the kth value
5. Create a method to find the largest kth value
6. Sort the array using the Sort() function
7. Return the kth value
8. Outputs the result to the user

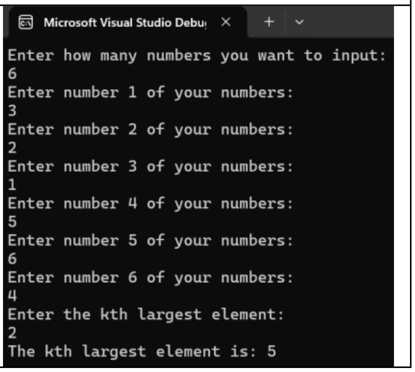
Evidence

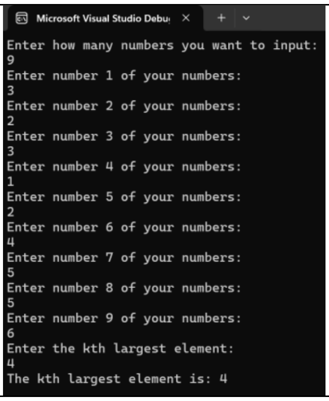


```
// See https://aka.ms/new-console-template for more information

// Asking the user how many values they want in the array
Console.WriteLine("Enter how many numbers you want to input:  ");
int arrayLength = Convert.ToInt32(Console.ReadLine());

// Creating the array based on the length the user wanted
int[] nums = new int[arrayLength];

// Creating a loop for the user to input a number for each index of the array, and then adding it to the
array until they meet their selected length
for (int i = 0; i < arrayLength; i++)
{
    Console.WriteLine($"Enter number {i + 1} of your numbers: ");
    nums[i] = Convert.ToInt32(Console.ReadLine());
}

// Asking the user to input the kth largest element
Console.WriteLine("Enter the kth largest element: ");
int k = Convert.ToInt32(Console.ReadLine());

// Creating the method to get the largest kth element
static int FindKthLargest(int[] nums, int k)
{
    // Sort the array in ascending order
    Array.Sort(nums);

    // Returns the the kth largest element which is at the index nums.Length - k
    return nums[(nums.Length - k)];
}

// Creates a variable to store the result and outputs it to the user
int result = FindKthLargest(nums, k);
Console.WriteLine($"The kth largest element is: {result}");
```

Figure 7 – Program 4: Solution 1

Testing

Test Data	Expected Outcome	Result
nums = [3,2,1,5,6,4], k = 2	The kth largest element is: 5	 Enter how many numbers you want to input: 6 Enter number 1 of your numbers: 3 Enter number 2 of your numbers: 2 Enter number 3 of your numbers: 1 Enter number 4 of your numbers: 5 Enter number 5 of your numbers: 6 Enter number 6 of your numbers: 4 Enter the kth largest element: 2 The kth largest element is: 5

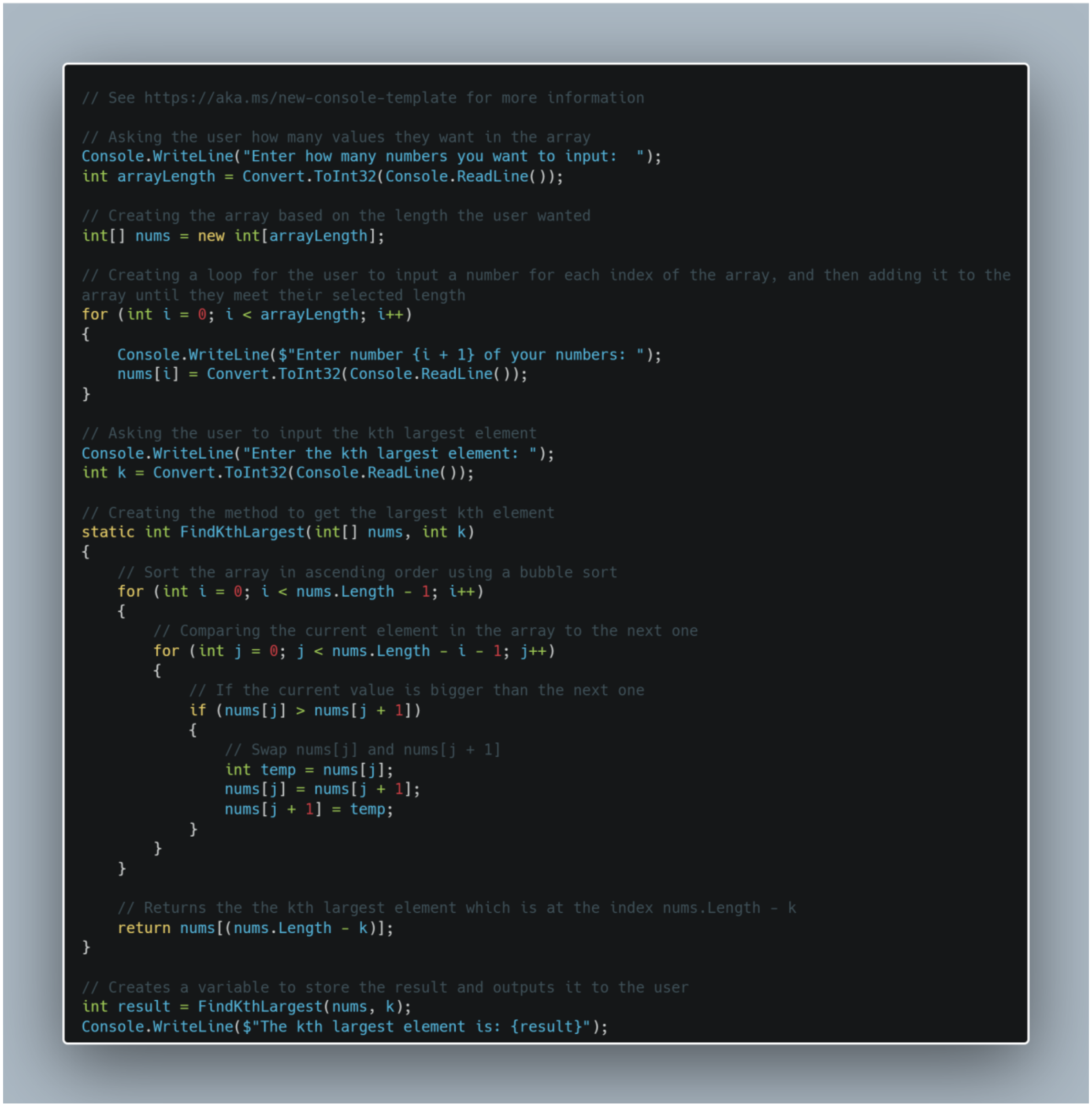
nums = [3,2,3,1,2,4,5,5,6], k = 4	The kth largest element is: 4	 Enter how many numbers you want to input: 9 Enter number 1 of your numbers: 3 Enter number 2 of your numbers: 2 Enter number 3 of your numbers: 3 Enter number 4 of your numbers: 1 Enter number 5 of your numbers: 2 Enter number 6 of your numbers: 4 Enter number 7 of your numbers: 5 Enter number 8 of your numbers: 5 Enter number 9 of your numbers: 6 Enter the kth largest element: 4 The kth largest element is: 4

Program 4: Solution 2

Decomposition

9. Ask the user to input the size of array they require
10. Create an array based on the size inputted
11. Create a FOR loop to ask the user to input a number for each index value in the array
12. Ask the user to input the kth value
13. Create a method to find the largest kth value
14. Sort the array using a bubble sort
15. Return the kth value
16. Outputs the result to the user

Evidence

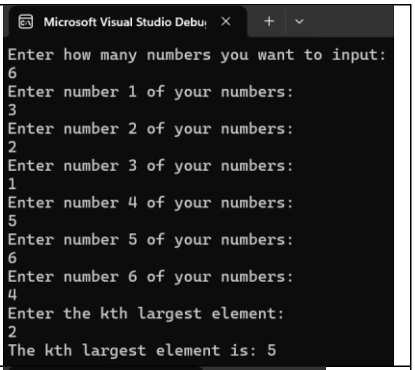
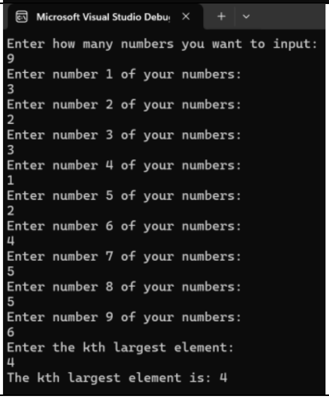


```
// See https://aka.ms/new-console-template for more information

// Asking the user how many values they want in the array
Console.WriteLine("Enter how many numbers you want to input:  ");
int arrayLength = Convert.ToInt32(Console.ReadLine());

// Creating the array based on the length the user wanted
int[] nums = new int[arrayLength];

// Creating a loop for the user to input a number for each index of the array, and then adding it to the
array until they meet their selected length
for (int i = 0; i < arrayLength; i++)
{
    Console.WriteLine($"Enter number {i + 1} of your numbers: ");
    nums[i] = Convert.ToInt32(Console.ReadLine());
}

// Asking the user to input the kth largest element
Console.WriteLine("Enter the kth largest element: ");
int k = Convert.ToInt32(Console.ReadLine());

// Creating the method to get the largest kth element
static int FindKthLargest(int[] nums, int k)
{
    // Sort the array in ascending order using a bubble sort
    for (int i = 0; i < nums.Length - 1; i++)
    {
        // Comparing the current element in the array to the next one
        for (int j = 0; j < nums.Length - i - 1; j++)
        {
            // If the current value is bigger than the next one
            if (nums[j] > nums[j + 1])
            {
                // Swap nums[j] and nums[j + 1]
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
        }
    }

    // Returns the the kth largest element which is at the index nums.Length - k
    return nums[(nums.Length - k)];
}

// Creates a variable to store the result and outputs it to the user
int result = FindKthLargest(nums, k);
Console.WriteLine($"The kth largest element is: {result}");
```

Figure 8 – Program 4: Solution 2

Testing

Test Data	Expected Outcome	Result

nums = [3,2,1,5,6,4], k = 2	The kth largest element is: 5	 Enter how many numbers you want to input: 6 Enter number 1 of your numbers: 3 Enter number 2 of your numbers: 2 Enter number 3 of your numbers: 1 Enter number 4 of your numbers: 5 Enter number 5 of your numbers: 6 Enter number 6 of your numbers: 4 Enter the kth largest element: 2 The kth largest element is: 5
nums = [3,2,3,1,2,4,5,5,6], k = 4	The kth largest element is: 4	 Enter how many numbers you want to input: 9 Enter number 1 of your numbers: 3 Enter number 2 of your numbers: 2 Enter number 3 of your numbers: 3 Enter number 4 of your numbers: 1 Enter number 5 of your numbers: 2 Enter number 6 of your numbers: 4 Enter number 7 of your numbers: 5 Enter number 8 of your numbers: 5 Enter number 9 of your numbers: 6 Enter the kth largest element: 4 The kth largest element is: 4

Program 4 Comparison and Evaluation

Solution 1 for this program makes use of the Sort() function, whereas solution 2 uses a manual bubble sort to sort the array. The predefined Sort() function operates using the efficient time complexity of O(n log n) (GeeksforGeeks, 2018), making it able to outperform solution 2's bubble sort which uses the less efficient O(n^2). This will be especially beneficial in use cases that feature large arrays, making solution 1 more suitable for the vast majority of scenarios. The code of solution 1 is also more concise and readable than solution 2, making it easier to follow and debug. This also makes solution 2 more susceptible to logic errors.

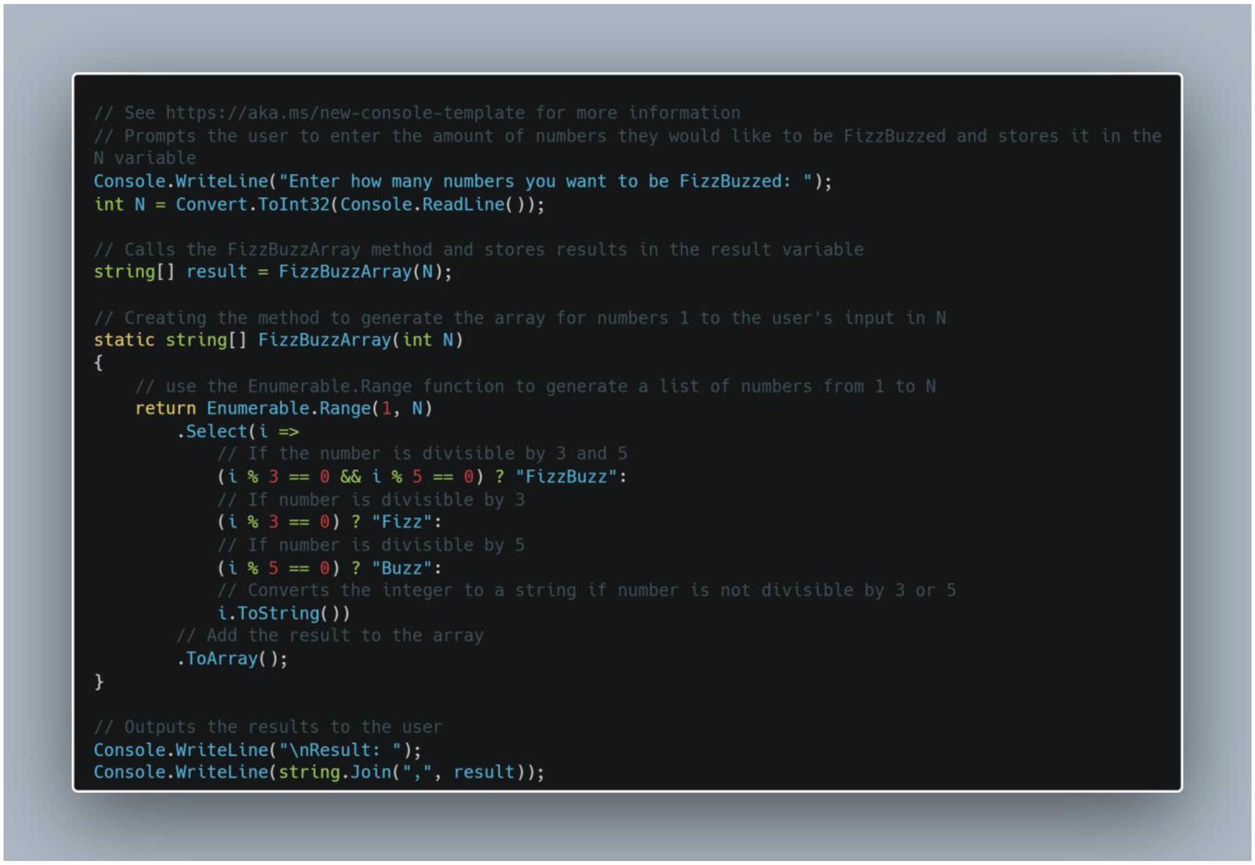
Overall, I think that solution 1 is superior in the vast majority of use cases as it is able to outperform solution 2 due to its more optimised and efficient time complexity. It also offers increased readability, allowing the code to remain easily maintainable and scalable for the future compared to the second solution.

Program 5: Solution 1

Decomposition

1. Ask the user to enter how many numbers they would like to be FizzBuzzed
2. Store that value in a variable
3. Create a results variable that stores the returned array from the FizzBuzz method
4. Create the method to generate the array of numbers from 1 to the user's input, using the Enumerate.Range() function
5. Perform FizzBuzz check using modulus operator to check if number is divisible by 3 or 5
6. Convert results to string and then add to the array
7. Output the results to the user

Evidence

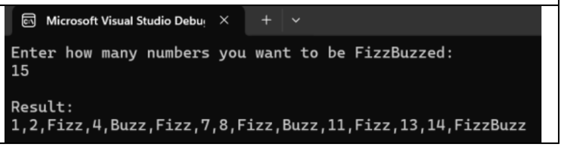


```
// See https://aka.ms/new-console-template for more information
// Prompts the user to enter the amount of numbers they would like to be FizzBuzzed and stores it in the
N variable
Console.WriteLine("Enter how many numbers you want to be FizzBuzzed: ");
int N = Convert.ToInt32(Console.ReadLine());

// Calls the FizzBuzzArray method and stores results in the result variable
string[] result = FizzBuzzArray(N);

// Creating the method to generate the array for numbers 1 to the user's input in N
static string[] FizzBuzzArray(int N)
{
    // use the Enumerable.Range function to generate a list of numbers from 1 to N
    return Enumerable.Range(1, N)
        .Select(i =>
            // If the number is divisible by 3 and 5
            (i % 3 == 0 && i % 5 == 0) ? "FizzBuzz":
            // If number is divisible by 3
            (i % 3 == 0) ? "Fizz":
            // If number is divisible by 5
            (i % 5 == 0) ? "Buzz":
            // Converts the integer to a string if number is not divisible by 3 or 5
            i.ToString())
        // Add the result to the array
        .ToArray();
}

// Outputs the results to the user
Console.WriteLine("\nResult: ");
Console.WriteLine(string.Join(",", result));
```

Figure 9 – Program 5: Solution 1

Testing

Test Data	Expected Outcome	Result

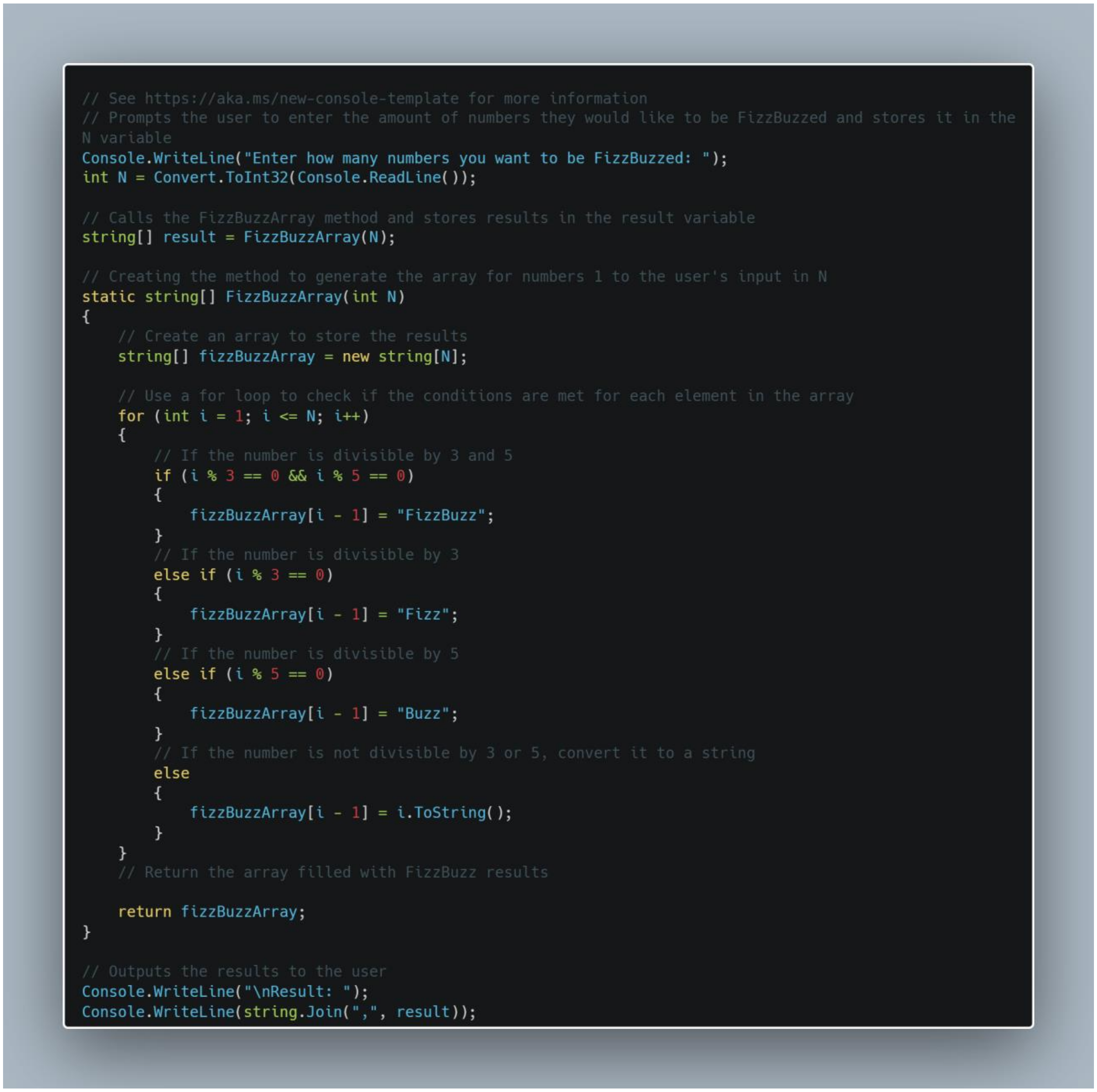
15	Result:
	1, 2, Fizz, 4. Buzz. Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz
		
		`Enter how many numbers you want to be FizzBuzzed:`
		`15`
		
		`Result:`
		`1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz`

Program 5: Solution 2

Decomposition

1. Ask the user to enter how many numbers they would like to be FizzBuzzed
2. Store that value in a variable
3. Create a results variable that stores the returned array from the FizzBuzz method
4. Create the method to generate the array of numbers from 1 to the user's input
5. Perform FizzBuzz check using an IF statament to check if each condition is met
6. Convert results to string and then add to the array
7. Return the array with the FizzBuzz result
8. Output the results to the user

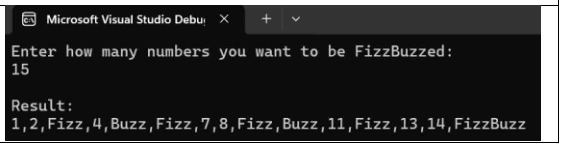
Evidence



```
// See https://aka.ms/new-console-template for more information
// Prompts the user to enter the amount of numbers they would like to be FizzBuzzed and stores it in the
N variable
Console.WriteLine("Enter how many numbers you want to be FizzBuzzed: ");
int N = Convert.ToInt32(Console.ReadLine());

// Calls the FizzBuzzArray method and stores results in the result variable
string[] result = FizzBuzzArray(N);

// Creating the method to generate the array for numbers 1 to the user's input in N
static string[] FizzBuzzArray(int N)
{
    // Create an array to store the results
    string[] fizzBuzzArray = new string[N];

    // Use a for loop to check if the conditions are met for each element in the array
    for (int i = 1; i <= N; i++)
    {
        // If the number is divisible by 3 and 5
        if (i % 3 == 0 && i % 5 == 0)
        {
            fizzBuzzArray[i - 1] = "FizzBuzz";
        }
        // If the number is divisible by 3
        else if (i % 3 == 0)
        {
            fizzBuzzArray[i - 1] = "Fizz";
        }
        // If the number is divisible by 5
        else if (i % 5 == 0)
        {
            fizzBuzzArray[i - 1] = "Buzz";
        }
        // If the number is not divisible by 3 or 5, convert it to a string
        else
        {
            fizzBuzzArray[i - 1] = i.ToString();
        }
    }
    // Return the array filled with FizzBuzz results

    return fizzBuzzArray;
}

// Outputs the results to the user
Console.WriteLine("\nResult: ");
Console.WriteLine(string.Join(",", result));
```

Figure 10 – Program 5: Solution 2

Testing

Test Data	Expected Outcome	Result
15	Result: 1, 2, Fizz, 4. Buzz. Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz	 Enter how many numbers you want to be FizzBuzzed: 15 Result: 1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz

Program 5 Comparison and Evaluation

Solution 1 for this program makes use of the Enumerate.Range() and Select() functions from LINQ, which allows for a more concise method of generating the array to store the results and less code validate if the element is a Fizz, Buzz or FizzBuzz. While this can allow for shorter code, the syntax for this can be quite complex and confusing, especially for those less familiar with LINQ functions. This makes solution 2 more readable, despite the fact more code is used. This can then benefit solution 2 in terms of maintainability for the future. Solution 1 uses the predefined functions that are optimised for high performance to fulfill their purpose, this would benefit the solution in terms of efficiency and speed, particularly if a larger number is input from the user to be FizzBuzzed. However, for this program, the performance is unlikely to have a big impact.

Therefore, I feel that solution 2 is the preferred solution for this use case due to its easily readable and logical code that is still capable of performing the required task.

Reference List

GeeksforGeeks (2018) *Analysis of Algorithms | Big-O analysis - GeeksforGeeks*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/ (Accessed: 12 January 2025).