2019
SEM 1

CAB301 ASSIGNMENT 2

EMPIRICAL COMPARISION OF TWO ALGORITHMS

JEFF LO (N9620699)
ARIK INTENAM MIR (N9637567)

17/05/2019

Abstract

This report is intended to analyze and compare two distinct algorithms both theoretically and empirically. Though these algorithms differ in efficiency, the task of both of these algorithms are same, they both find out the minimum distance between two elements in an array. Identifying the basic operations and execution time of both of the algorithm was processed, while the average-case theoretical efficiency was done by thorough calculations and in-depth observations. The aforementioned algorithms were implemented in C# on a Windows environment. Number of basic operations and time to execute the algorithms, were known after taking arrays of varying lengths to account. The obtained results for both of these algorithms are compared against the theoretical predictions to understand the trend of how these algorithms operate. The functional correctness of the algorithms were also tested, while the experimental results were analyzed and graphed in Microsoft Excel. It was observed that the theoretical predictions of the basic operations and execution time for the algorithms matches with the experimental results.

Contents

A brief Description of the *MinDistance* Algorithm

The minimum distance algorithm, when implemented correctly, reads data from an array of numbers as input and computes the difference between any of its two numbers with the closest value, the difference between these two numbers would be the minimum distance and returned as output of the given array. More specifically, the algorithm starts by iterating through the same input array twice, the values from the second iteration is then subtracted from the values from the first iteration. The outputs are then compared against each other and only the least value output is assigned to a variable and returned. For example, a number array of [1, 60, 22, 10, 5, 8, 121, 2, 11], would return a minimum distance of 1 because the two numbers with the closest value are 1 & 2 and their difference is 1. This algorithm has more iterations because it always starts comparing the same elements in the iterations.

A brief Description of the *MinDistance2* Algorithm

The minimum distance algorithm, when implemented correctly, reads data from an array of numbers as input and computes the difference between any of its two numbers with the closest value, the difference between these two numbers would be the minimum distance and returned as output of the given array. More specifically, the algorithm starts by iterating through the same input array twice, the values from the second iteration is then subtracted from the values from the first iteration. The outputs are then assigned to a variable and compared against each other and only the least value output is returned. For example, a number array of [1, 60, 22, 10, 5, 8, 121, 2, 11], would return a minimum distance of 1 because the two numbers with the closest value are 1 & 2 and their difference is 1. This algorithm has less iterations because it always starts comparing the previous element with the next element in the iterations.

Theoretical Analysis of the Two Minimum Distance Algorithms

This section, the algorithms are analyzed theoretically to achieve the hypothetical results of the algorithm which then can be used to compare with the experimental results achieved after the numerous tests. The comparison focuses mostly on the theoretical time efficiency and experimental results for both algorithms, which is this case, the MinDistance2 algorithm is theoretically more time efficient than the MinDistance algorithm.

Identifying the basic operation and problem size for both algorithms.

Basic Operation explanation of choice for MinDistance

For the first algorithm, according to figure 1 in the appendix, there are 5 lines of commands. There is a nested for loop, a condition check, where a value is assigned when the condition is met and then a value is returned. After careful analysis, we found there is only one basic operation in the MinDistance algorithm and it is the condition check, which compares any two numbers in the inputted array and returns the lowest difference found. This was the choice because the condition check line is the most executed line of command and contributes most towards the running time of the algorithm. It is the third line of command in the MinDistance algorithm.

Basic Operation explanation of choice for MinDistance2

For the second algorithm, according to figure 2 in the appendix, there are 6 lines of commands. There is a nested for loop, a value assignment, a condition check, where a value is assigned when the condition is met and then a value is returned. After careful analysis, we found there is only one basic operation in the

MinDistance2 algorithm and it is the condition check, which compares any two numbers in the inputted array and returns the lowest difference found. This was the choice because the condition check line is the most executed line of command and contributes most towards the running time of the algorithm. It is the fourth line of command in the MinDistance2 algorithm.

Problem size of the algorithms

The problem size of the algorithms is referred to as the size of the input array used in the algorithms before execution. Both algorithms use the same problem sizes starting with an input size of 500 elements to 15,000 elements with increments of 500. These values were chosen carefully because they produce sufficient amount of results for analysis and the total execution time required to achieve the results is not very long.

Theoretical analysis of time efficiency for both algorithms.

Algorithm 1, MinDistance

Theoretically, MinDistance is less time efficient than the second algorithm, MinDistance2, which means more time is needed to complete all operations given the same input and problem size is used for the testing. Below this is the mathematical calculations for it.

$$MinDistance(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1$$

Using the summation formula:

$$\sum_{i=l}^{u} u - l + 1$$

We get:

$$MinDistance(n) = \sum_{i=0}^{n-1} ((n-1) - 0 + 1)$$

$$MinDistance(n) = \sum_{i=0}^{n-1} n$$

$$MinDistance(n) = n \sum_{i=0}^{n-1} 1$$

Using the same summation formula, we get:

$$MinDistance(n) = n\big((n-1) - 0 + 1\big)$$

$$= n(n)$$

$$\therefore \; MinDistance(n) = n^2$$

Therefore, algorithm one belongs to the efficiency class $\epsilon\theta(n^2)$

Algorithm 2, MinDistance2

Algorithm 2 is theoretically more efficient than algorithm 1. Below is the calculation to prove it.

$$MinDistance2(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-2} 1$$

Using the summation formula:

$$\sum_{i=l}^{u} u - l + 1$$

We get:

$$MinDistance2(n) = \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1)$$

$$MinDistance2(n) = \sum_{i=0}^{n-2} (n - 1 - i)$$

$$MinDistance2(n) = \sum_{i=0}^{(n-1)-1} ((n-1) - i)$$

Using the summation formula:

$$\sum_{i=0}^{u-1} (u - i) = \frac{u(u+1)}{2}$$

We get:

$$= \frac{(n-1)\big((n-1)+1\big)}{2}$$

$$= \frac{n(n-1)}{2}$$

$$\therefore\ MinDistance2(n) = \frac{n^2 - n}{2}$$

Therefore, algorithm two also belongs to the efficiency class $\epsilon\theta(n^2)$

From the calculations above, it can be seen that both algorithms belong to the same efficiency class. Also, algorithm two is proven to be more time efficient according to the calculations. Therefore, with the known efficiency class for both algorithms, the theoretical basic operation count for both algorithms can now be obtained by using the formula $\mathrm{C_{ave}} \times \mathrm{n}^2$.

Implementation of the Algorithm and Experiments

- The MinDistance and the MinDistance algorithms and the pertinent observations and experiments were implemented using programming language C#(C Sharp). It is a free, universal and multi-paradigm programming language, with simple and user-friendly syntax (Wikipedia, n.d.).

- All of the experiments were implemented on an Asus laptop computer, running on the operating system Windows 10, with the system running at 1.99 GHz.
- We used the C# built-in **Random** class and its method **.Next()** to produce the test data needed for the experiments. Other concurrent software and applications were minimized during measuring the execution time
- The time it took for the algorithms to execute were measured using the built-in **Stopwatch** class. The execution time was measured in the units of milliseconds.
- The results of the experiments were exported from Virtual Studio to Microsoft Excel 2013 as a .CSV format file for further analysis and graphing by converting the datapoints into a Table. The Method **File.WriteAllText()** was used to convert the gathered data from C# to a CSV format. **File.WriteAllText()** takes the file path in the form of an integer.
- The observations and the results of the experiments were exported from Virtual Studio 2019 to Microsoft Excel in the form of .CSV files for graphing by converting the datapoints to a table.
- All of the figures were all prepared in Excel. Further knowledge required for exporting data from C# to a .CSV file were learnt by thorough research on StackOverflow(vc 74, 2011). The report was prepared using Microsoft Word 2013.

Results

In this section, the results gathered from the implementations of the algorithm undergo a functional correctness procedure and confirmed whether the implementation is correct and then the results are analyzed.

Functionality Correctness Results

The table given below demonstrates the functional correctness of the MinDistance & Mindistance2 algorithms. Implementation code is displayed in the appendix section. Only

TEST CASE	**TEST INSTANCE**	**Expected output for MinDistance**	**Expected output for MinDistance2**	**TEST RESULT**
Array of Consecutive Numbers	sorted_arr =[55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]	1	1	**Successful**
Array of Unsorted Numbers	sorted_arr= [20, 23, 57, 5, 131, 15, 292, 27, 13, 17, 88, 324, 123]	2	2	**Successful**
Array of Sorted Numbers	unsorted_arr = [5, 13, 15, 17, 20, 23, 27, 57, 88, 123, 131, 292, 324]	2	2	**Successful**
Reverse Sorted array	reverse_arr = [324, 292, 131, 123, 88, 57, 27, 23, 20, 17, 15, 13, 5]	2	2	**Successful**
Array of same digits	same_arr = [13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13]	0	0	**Successful**

Array of Partially Sorted Numbers	partial_arr = [13, 15, 17, 19, 27, 45, 11, 34, 31, 149]	2	2	**Successful**
Array of mixed integers	mixed_arr = [-99, 19, 17, 78, 2887, 45, -1, 34, -231, -1049]	2	2	**Successful**
Array of only negative integers	negative_arr = [-23, -99, -735, -44, -4, -1111, -88, -5324, -122, -12]	8	8	**Successful**
Array of Large and Small Integers	large_difference_arr = [1, 2100000000]	20999999999	2099999999	**Successful**
Array of only one element	small_arr = [1]	∞	∞	**Successful**

Table 1: Functionality testing of MinDistance & Mindistance2 algorithms.

Experimental Results for BOTH Algorithms and its Analysis

Below are the graphs of comparison in different aspects between the two algorithms.

The figure below shows the comparison of the number of basic operations for both algorithms.

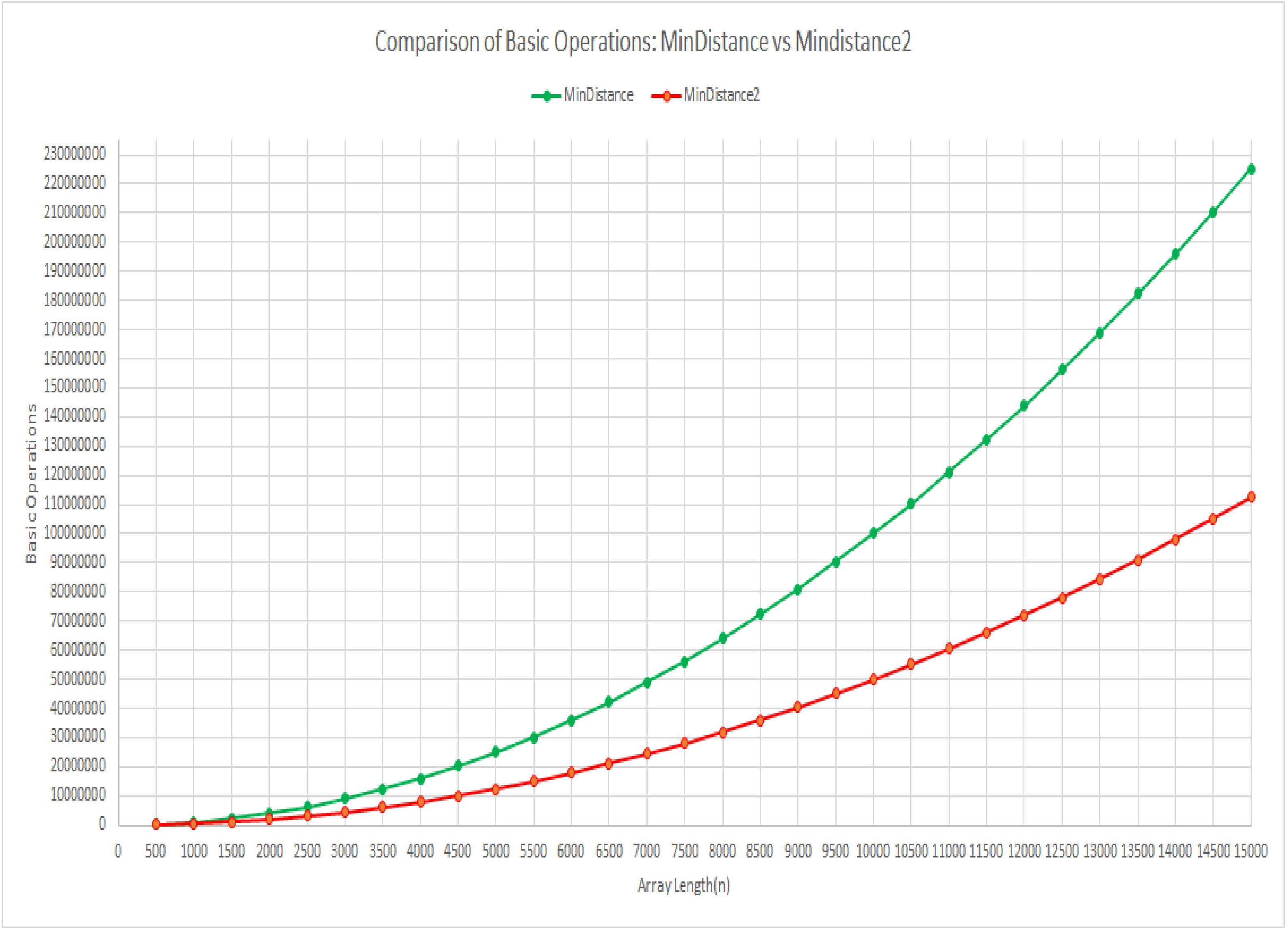


Figure 1

Comparison of Basic Operations: MinDistance1 & MinDistance2

After graphing and comparing the experimental results of the number of basic operations of the algorithms, it was observed that the graphs were plotted in an expected manner. Referring to figure 1, it is seen that the number of basic operations for the 1st algorithm is slightly above the 2200, 00,000 mark, while the number of basic operations for the 2nd one is slightly above the 1100, 00,000 mark. Through thorough analysis it can be concluded that the basic operations for the 1st algorithm is incremented by the multiplier of n^2 , while for the 2nd algorithm it is $n^2 - \frac{n}{2}$. So it is certainly clear from the graphs and the analysis that the MinDistance2, algorithm 2 is more efficient than MinDistance.

This graph shows the execution times of both algorithms against different input sizes.

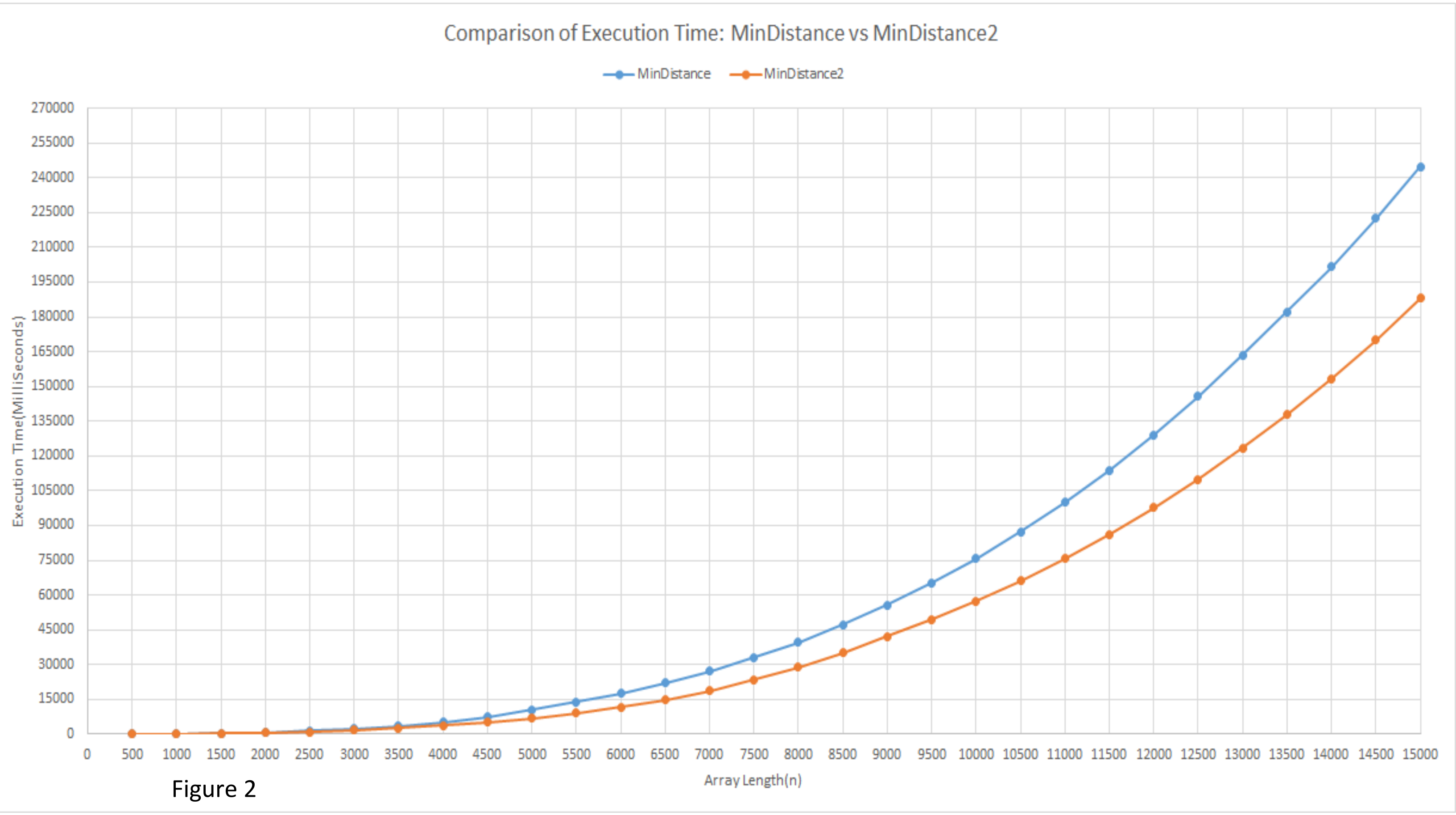


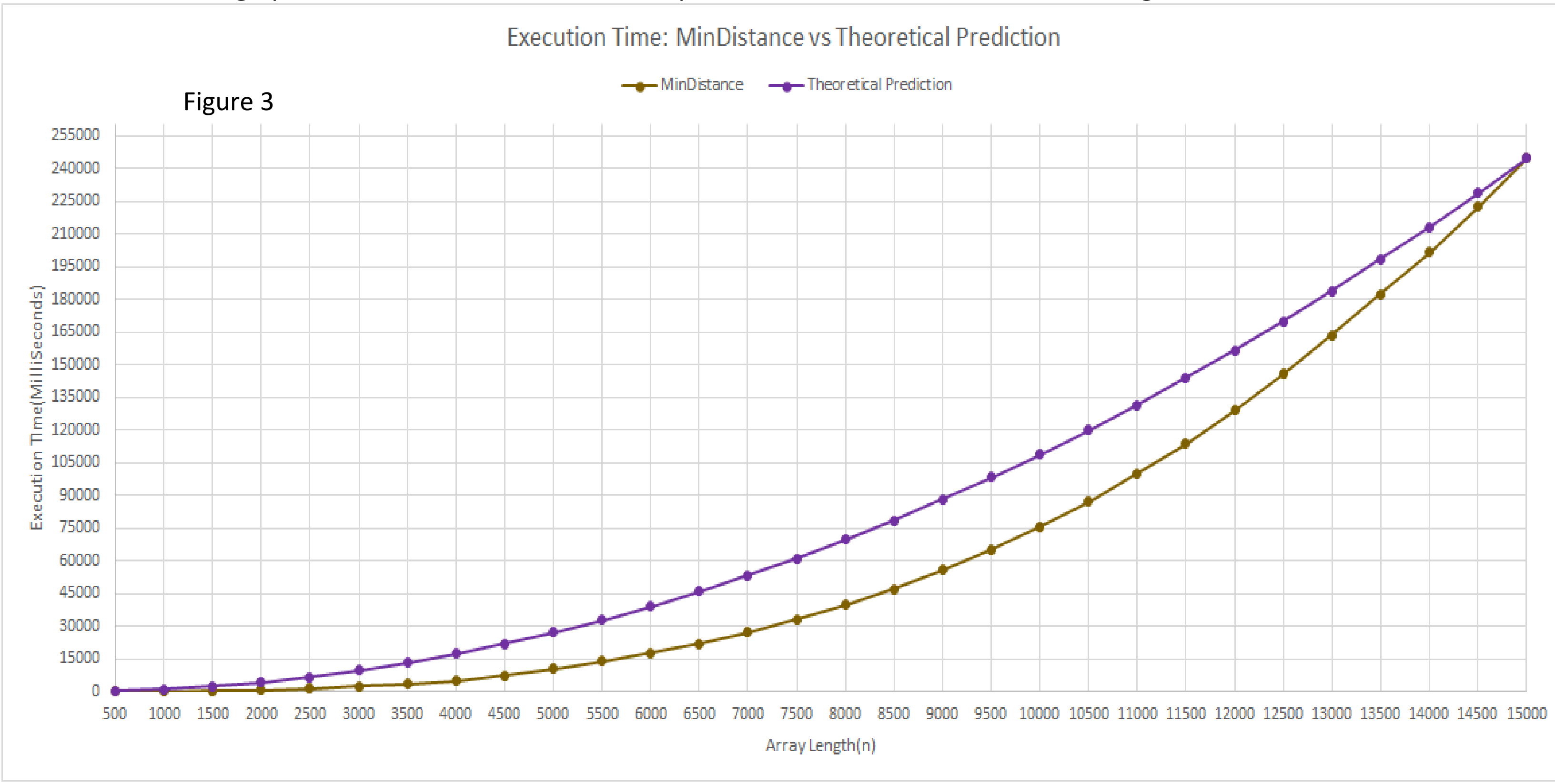
Figure 2

Comparison of Execution Time: MinDistance & MinDistance2

From figure 2, which is the graph for the comparison of execution time between MinDistance and MinDistance2, it is observed that the MinDistance2 takes lesser time to finish its operation on the same array length incrementing by 500, also used for MinDistance. While MinDistance finishes its operation at around 244000 milliseconds or 244 milliseconds, MinDistance2 finishes at around 188000 milliseconds or 188 seconds. Thus saying that algorithm 2 is more time efficient than algorithm 1 is justifiable.

This graph shows the theoretical and the experimental execution time for the first algorithm

Figure 3



Execution Time Analysis: MinDistance vs Theoretical Prediction

After analyzing and graphing the actual execution time against the theoretical prediction for MinDistance in figure 3, it is perceived that the actual time to execute matches average case execution time for this algorithm. Even though there were some deviation in the mid points of the graph for theoretical prediction, but the behavior matches with the actual execution time. The theoretical prediction was calculated using the formula $C_{ave} \times n^2$

This graph shows the theoretical and the experimental execution time for the second algorithm

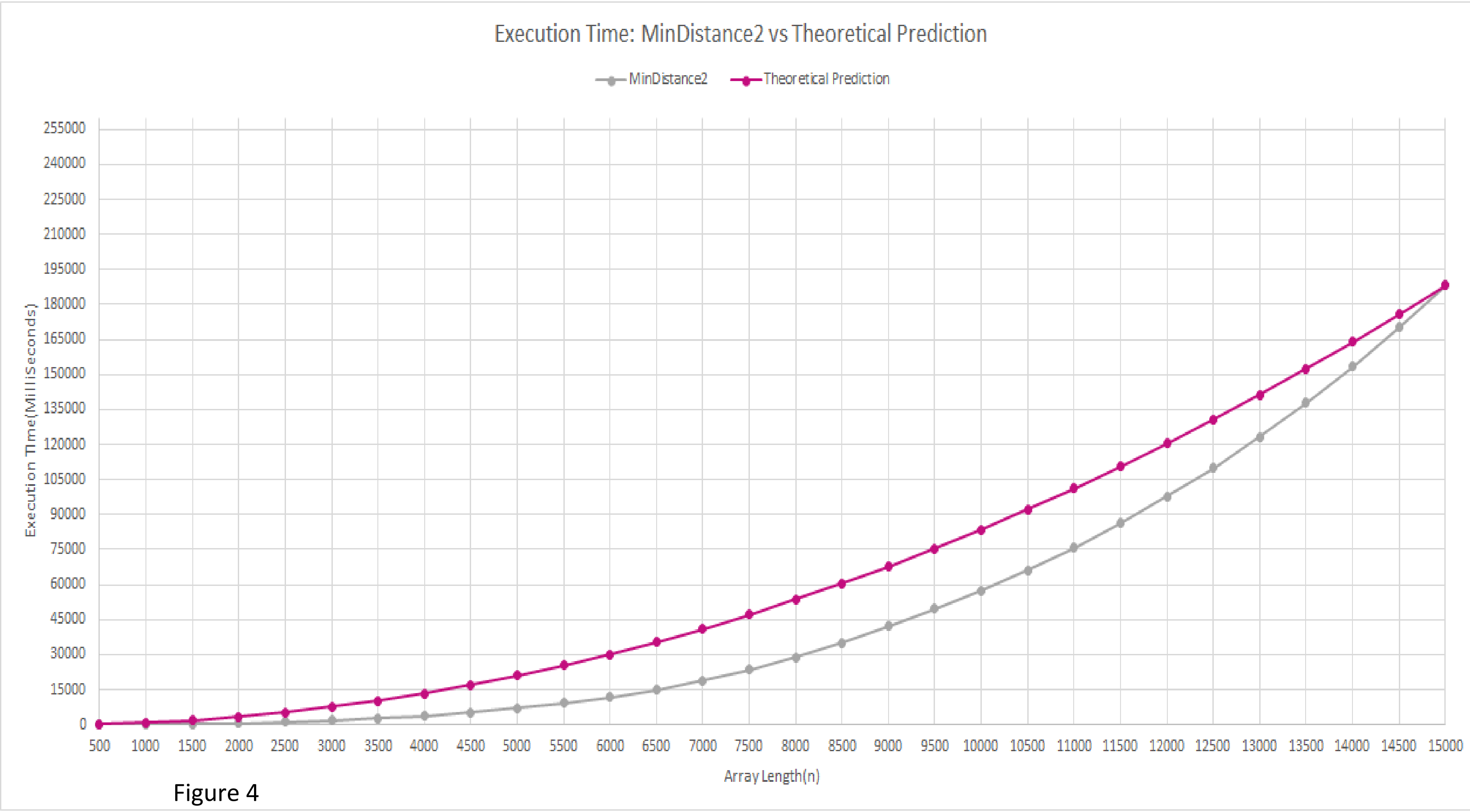


Figure 4

Execution Time Analysis: MinDistance2 vs Theoretical Prediction

Figure 4 represents the graph of execution time for MinDistance2 against its theoretical counterpart. As it was expected, the plot of the execution time for the 2nd algorithm matches with its theoretical counterpart. Just like the 1st algorithm, the theoretical prediction of execution time of the 2nd one is deviating slightly from the actual time. The theoretical prediction for execution time was formulated using the same formula used for 1st algorithm.

This graph shows the theoretical and the experimental basic operations count for the first algorithm.

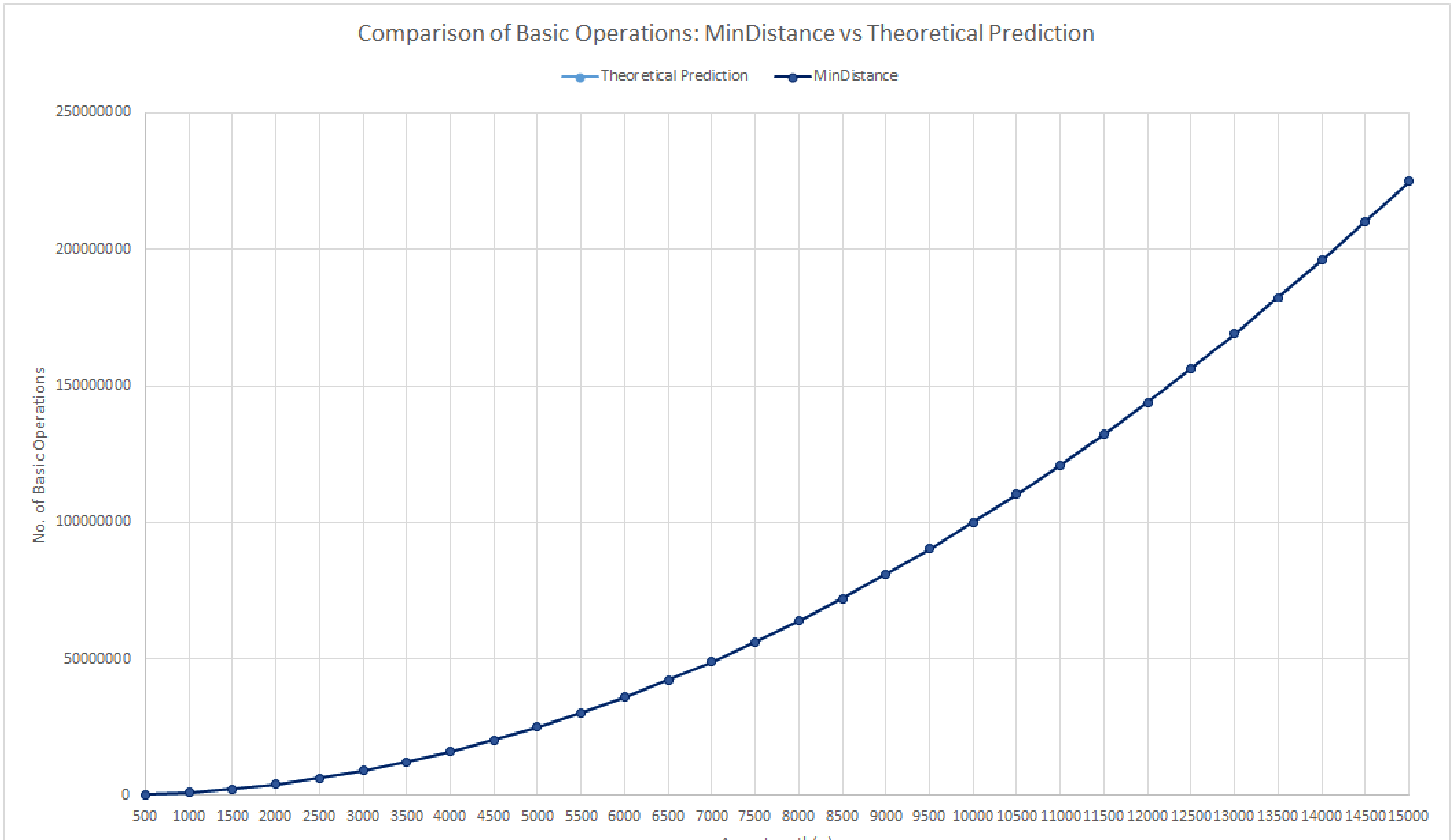


Figure 5

Basic Operations Analysis: MinDistance vs Theoretical Prediction

In Figure 5, the actual number of basic operations of the 1st algorithm is plotted against its theoretical counterpart. From the graph, it is not possible to distinguish the actual and theoretical basic operations, as their values are almost equal for each array length making, thus making them overlap each other. But it can be concluded by saying that the number of basic operations of the MinDistance matches its average case theoretical prediction.

This graph shows the theoretical and the experimental basic operations count for the first algorithm.

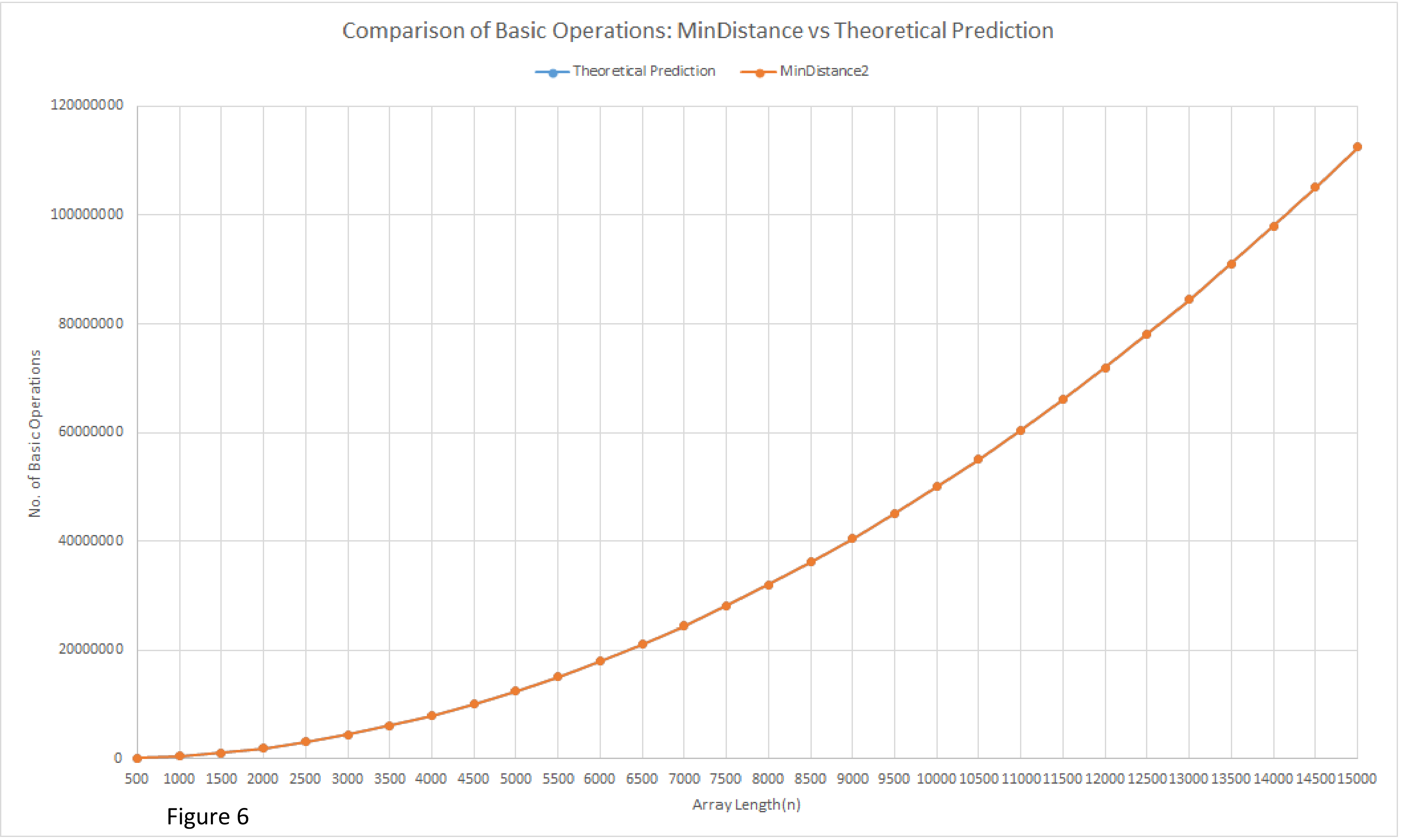


Figure 6

Basic Operations Analysis: MinDistance2 vs Theoretical Prediction

Figure 6 is the graph for the number of basic operations of MinDistance2 and its theoretical prediction. After analyzing, MinDistance2 was seen to be following the same path as MinDistance, because there were no difference in the graph of the actual and the theoretical prediction of basic operations. Thus, it looks like there is only one plot on the graph. The theoretical prediction for the number of basic operations was formulated using the formula$C_{ave} \times n^2$. And lastly it can be said that the real number of basic operations matches the theoretical prediction for MinDistance2

Conclusion

To conclude, after comparing both algorithms theoretically and experimentally using the obtained results, the improved version of the Minimum distance (MinDistance2) algorithm have been proven to be more efficient in both basic operations count and execution time meaning when compared to the original Minimum Distance algorithm (MinDistance), less operations were taken to complete the same task with a given input size and require less time to reach runtime completion overall.

References

This sections shows the references for the outside sources used.

1. C Sharp(Programming Language).(n.d.).
Retrieved on 6th April, 2019 from
https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

2. Vc 74. (2011). C# datatable to csv.
Retrieved on 6th April, 2019 from
https://stackoverflow.com/questions/4959722/c-sharp-datatable-to-csv

3. Microsoft .NET. (n.d.) Int32 Struct.
Retrieved on 7th April, 2019 from
https://docs.microsoft.com/en-us/dotnet/api/system.int32?view=netframework-4.7.2

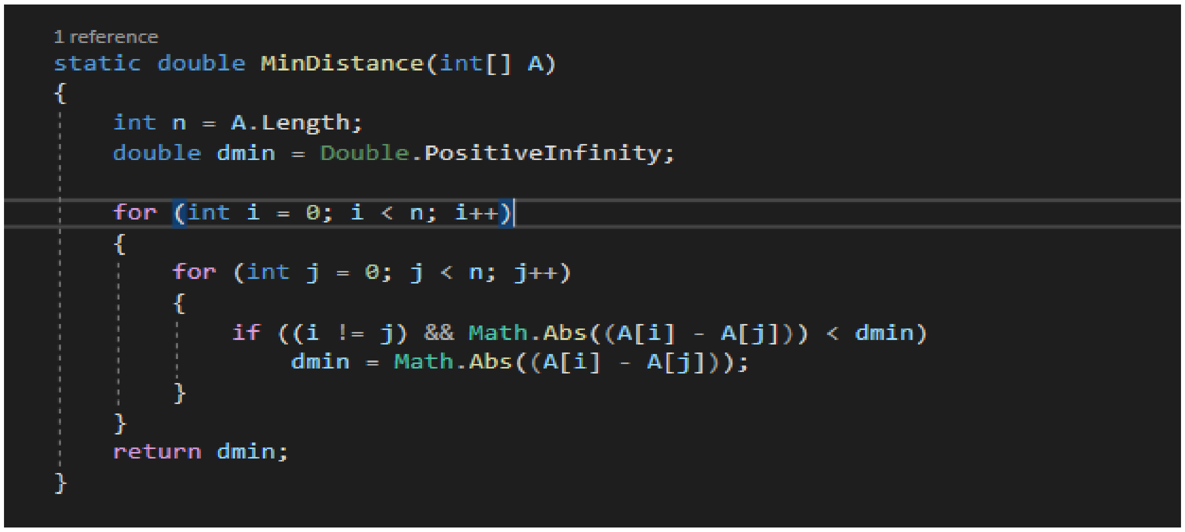
4. N9620699_CAB301_Assignment_1

5. N9637567_CAB301_Assignment_1

Appendices

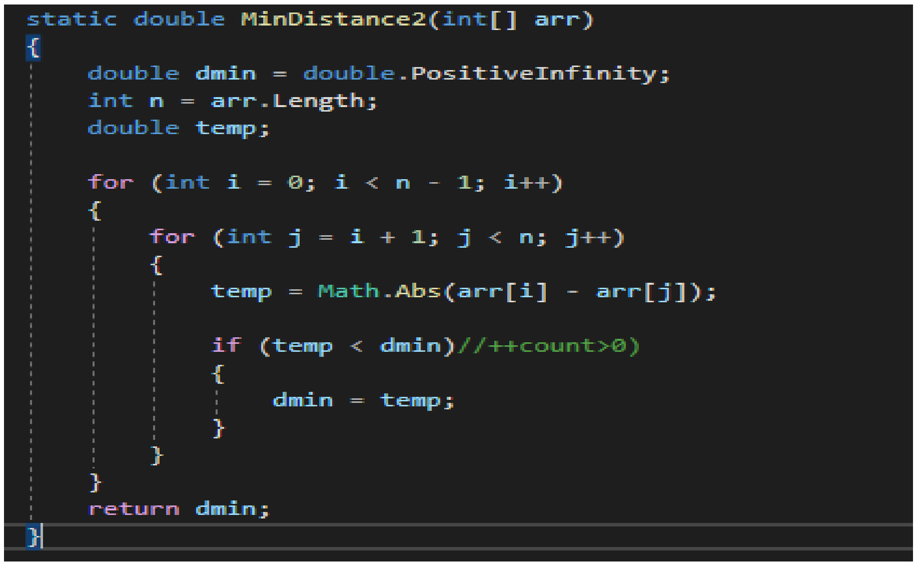
This section shows the source code for both algorithm implementations, including the basic operations count, the execution times and the functional testing.

Implementation of MinDistance Algorithm



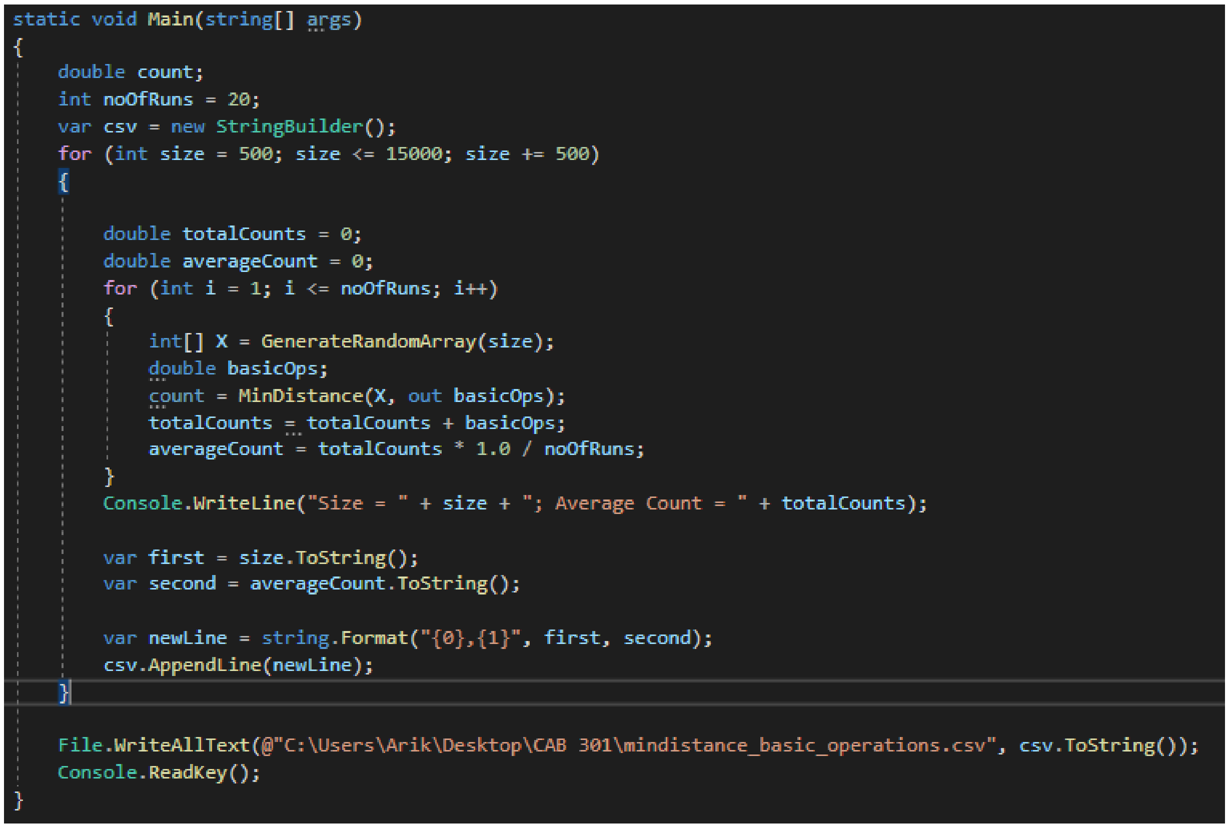
```
1 reference
static double MinDistance(int[] A)
{
    int n = A.Length;
    double dmin = Double.PositiveInfinity;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if ((i != j) && Math.Abs((A[i] - A[j])) < dmin)
                dmin = Math.Abs((A[i] - A[j]));
        }
    }
    return dmin;
}
```

Implementation of MinDistance2 Algorithm



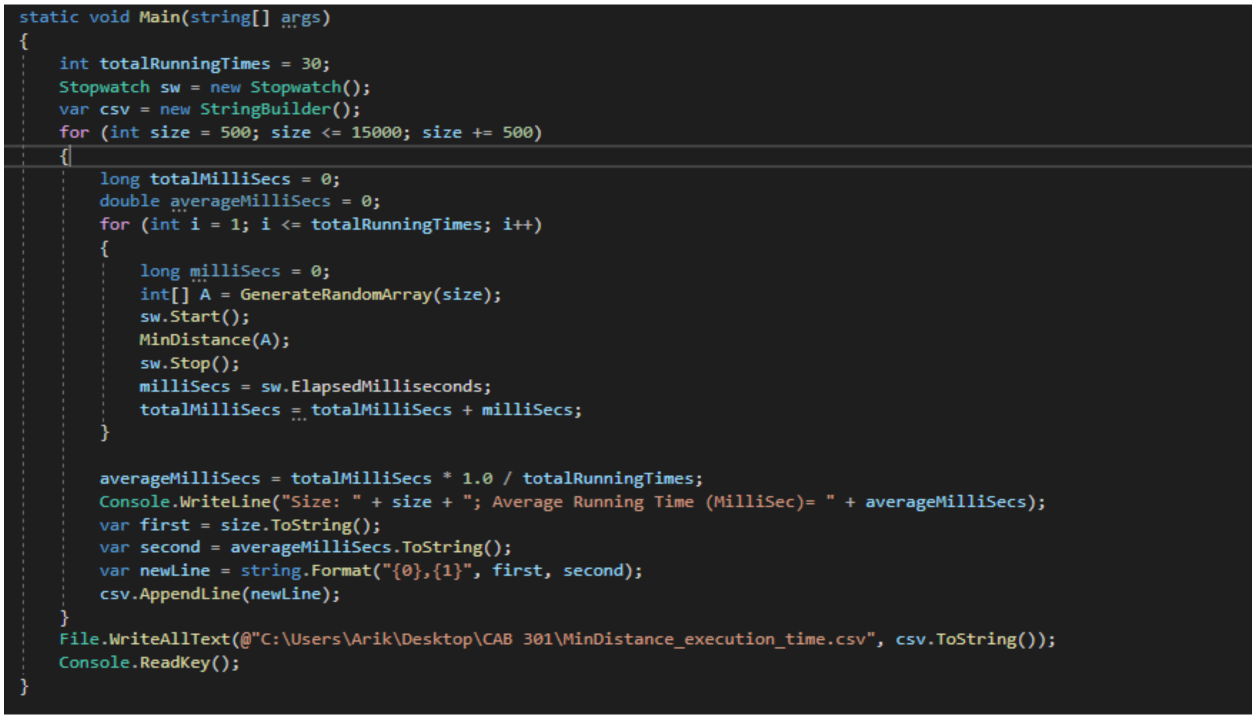
```
static double MinDistance2(int[] arr)
{
    double dmin = double.PositiveInfinity;
    int n = arr.Length;
    double temp;

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            temp = Math.Abs(arr[i] - arr[j]);

            if (temp < dmin)//++count>0)
            {
                dmin = temp;
            }
        }
    }
    return dmin;
}
```

Implementation of the method used for counting the basic operations



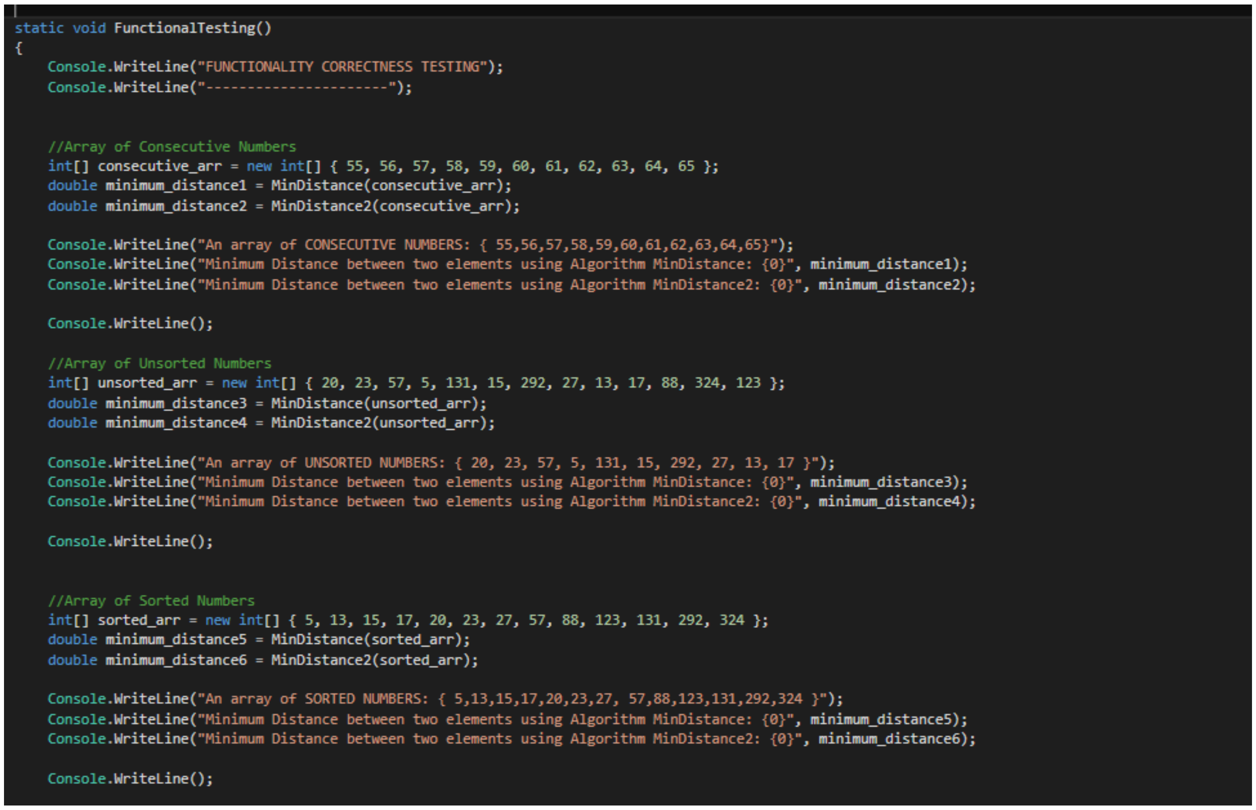
```
static void Main(string[] args)
{
    double count;
    int noOfRuns = 20;
    var csv = new StringBuilder();
    for (int size = 500; size <= 15000; size += 500)
    {

        double totalCounts = 0;
        double averageCount = 0;
        for (int i = 1; i <= noOfRuns; i++)
        {
            int[] X = GenerateRandomArray(size);
            double basicOps;
            count = MinDistance(X, out basicOps);
            totalCounts = totalCounts + basicOps;
            averageCount = totalCounts * 1.0 / noOfRuns;
        }
        Console.WriteLine("Size = " + size + "; Average Count = " + totalCounts);

        var first = size.ToString();
        var second = averageCount.ToString();

        var newLine = string.Format("{0},{1}", first, second);
        csv.AppendLine(newLine);
    }

    File.WriteAllText(@"C:\Users\Arik\Desktop\CAB 301\mindistance_basic_operations.csv", csv.ToString());
    Console.ReadKey();
}
```

Implementation of the method used for counting the execution time

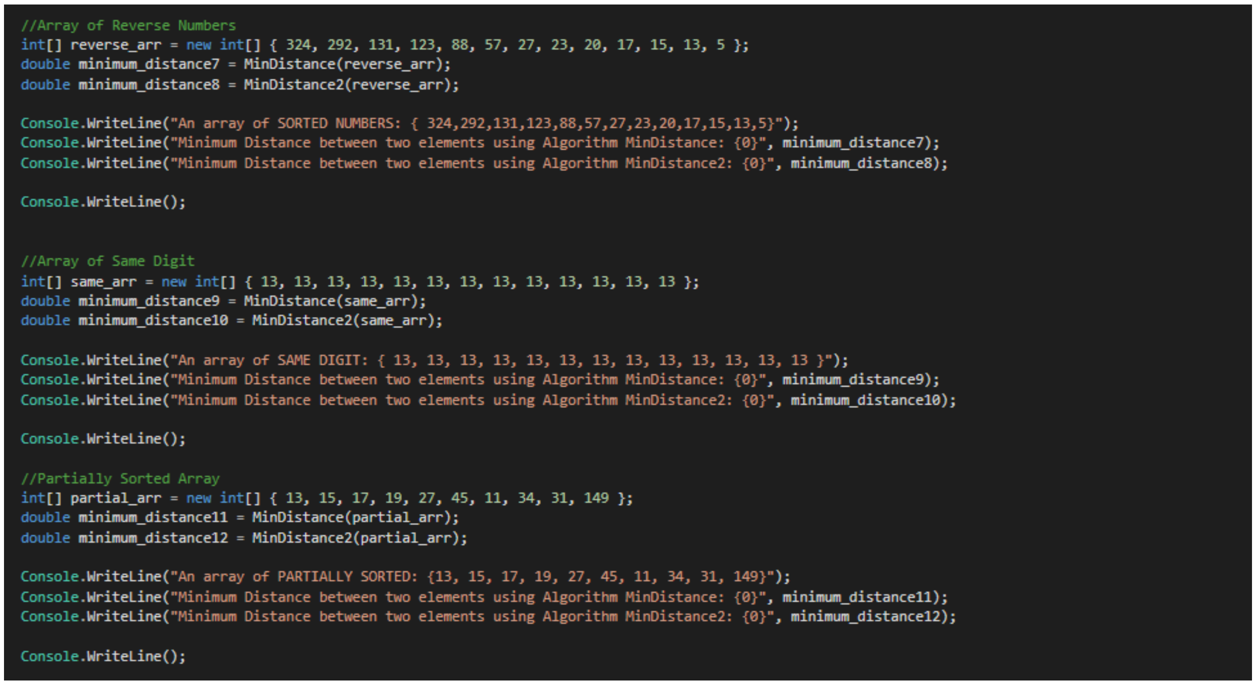


```
static void Main(string[] args)
{
    int totalRunningTimes = 30;
    Stopwatch sw = new Stopwatch();
    var csv = new StringBuilder();
    for (int size = 500; size <= 15000; size += 500)
    {
        long totalMilliSecs = 0;
        double averageMilliSecs = 0;
        for (int i = 1; i <= totalRunningTimes; i++)
        {
            long milliSecs = 0;
            int[] A = GenerateRandomArray(size);
            sw.Start();
            MinDistance(A);
            sw.Stop();
            milliSecs = sw.ElapsedMilliseconds;
            totalMilliSecs = totalMilliSecs + milliSecs;
        }

        averageMilliSecs = totalMilliSecs * 1.0 / totalRunningTimes;
        Console.WriteLine("Size: " + size + "; Average Running Time (MilliSec)= " + averageMilliSecs);
        var first = size.ToString();
        var second = averageMilliSecs.ToString();
        var newLine = string.Format("{0},{1}", first, second);
        csv.AppendLine(newLine);
    }
    File.WriteAllText(@"C:\Users\Arik\Desktop\CAB 301\MinDistance_execution_time.csv", csv.ToString());
    Console.ReadKey();
}
```

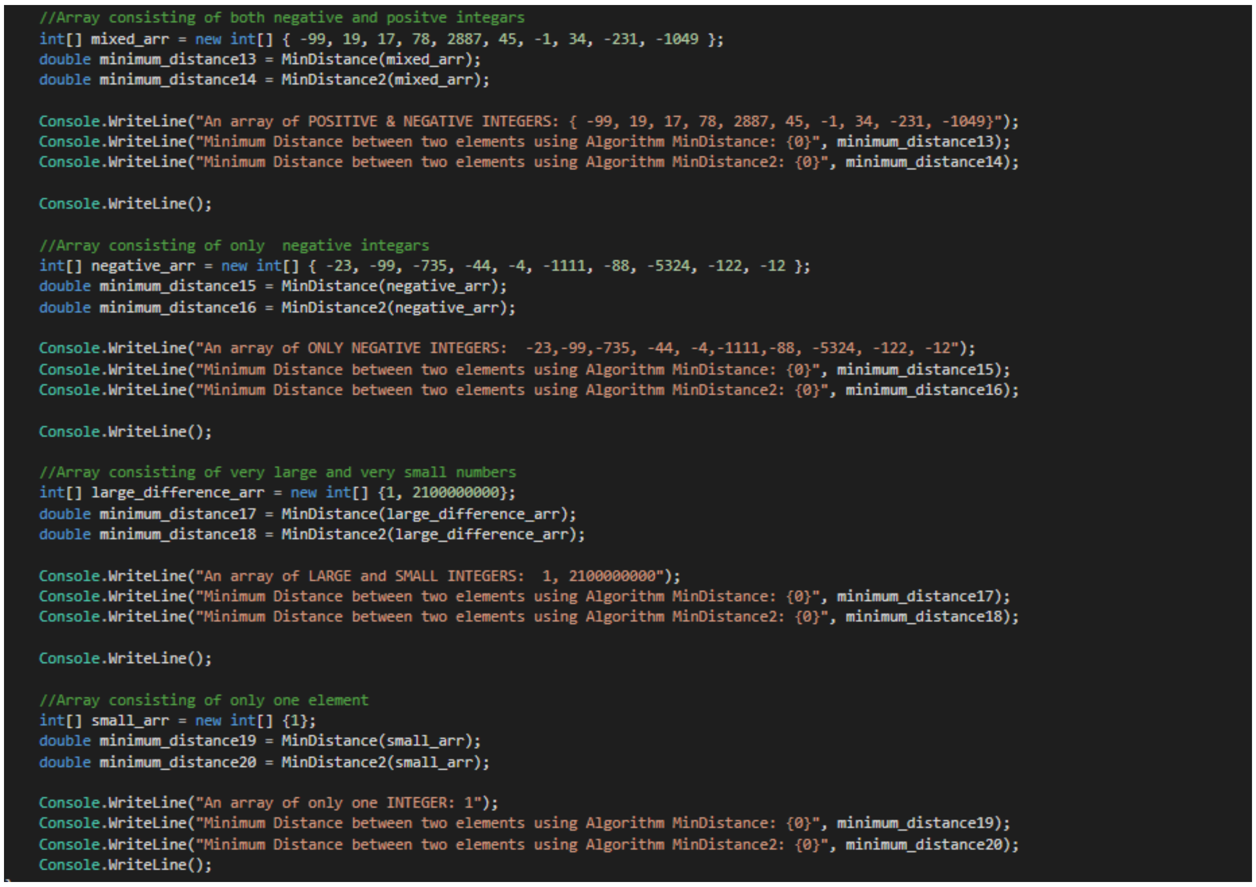
Implementation of functional testing, testing both algorithms



```
static void FunctionalTesting()
{
    Console.WriteLine("FUNCTIONALITY CORRECTNESS TESTING");
    Console.WriteLine("----------------------");


    //Array of Consecutive Numbers
    int[] consecutive_arr = new int[] { 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65 };
    double minimum_distance1 = MinDistance(consecutive_arr);
    double minimum_distance2 = MinDistance2(consecutive_arr);

    Console.WriteLine("An array of CONSECUTIVE NUMBERS: { 55,56,57,58,59,60,61,62,63,64,65}");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance1);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance2);

    Console.WriteLine();

    //Array of Unsorted Numbers
    int[] unsorted_arr = new int[] { 20, 23, 57, 5, 131, 15, 292, 27, 13, 17, 88, 324, 123 };
    double minimum_distance3 = MinDistance(unsorted_arr);
    double minimum_distance4 = MinDistance2(unsorted_arr);

    Console.WriteLine("An array of UNSORTED NUMBERS: { 20, 23, 57, 5, 131, 15, 292, 27, 13, 17 }");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance3);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance4);

    Console.WriteLine();


    //Array of Sorted Numbers
    int[] sorted_arr = new int[] { 5, 13, 15, 17, 20, 23, 27, 57, 88, 123, 131, 292, 324 };
    double minimum_distance5 = MinDistance(sorted_arr);
    double minimum_distance6 = MinDistance2(sorted_arr);

    Console.WriteLine("An array of SORTED NUMBERS: { 5,13,15,17,20,23,27, 57,88,123,131,292,324 }");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance5);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance6);

    Console.WriteLine();
```

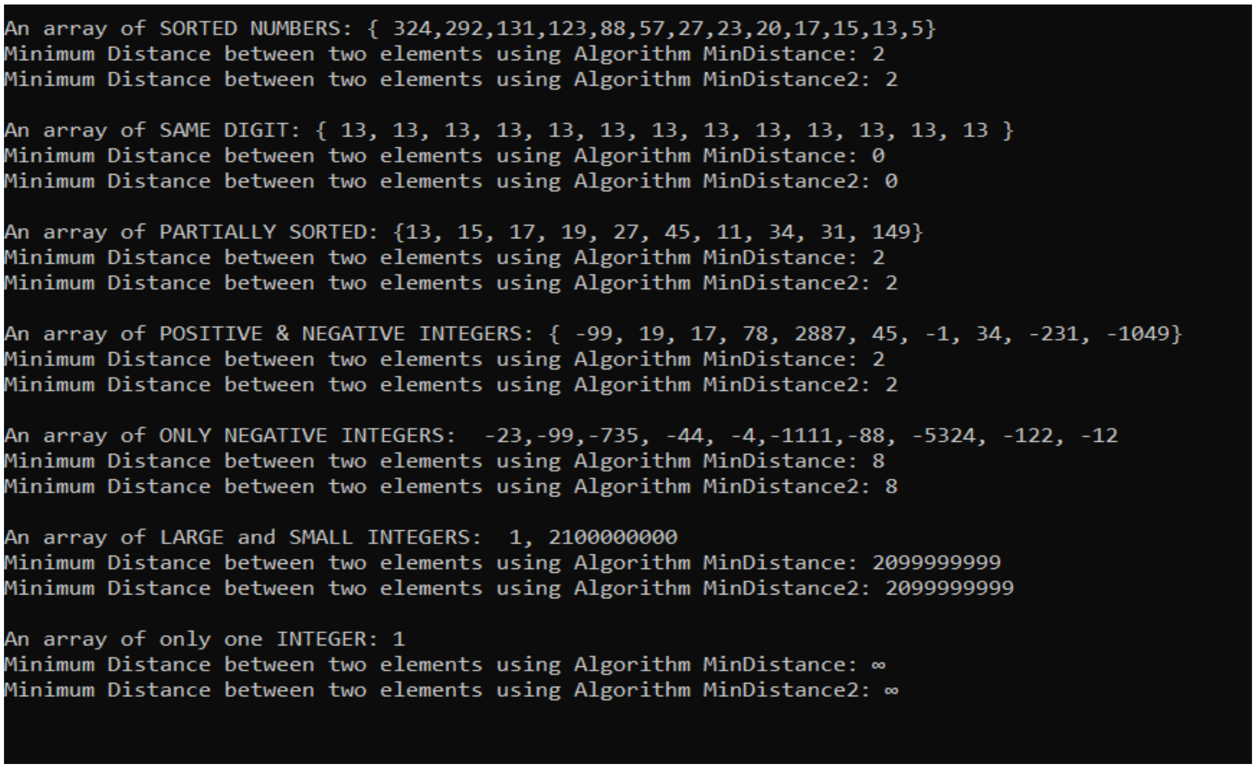


```
//Array of Reverse Numbers
int[] reverse_arr = new int[] { 324, 292, 131, 123, 88, 57, 27, 23, 20, 17, 15, 13, 5 };
double minimum_distance7 = MinDistance(reverse_arr);
double minimum_distance8 = MinDistance2(reverse_arr);

Console.WriteLine("An array of SORTED NUMBERS: { 324,292,131,123,88,57,27,23,20,17,15,13,5}");
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance7);
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance8);

Console.WriteLine();


//Array of Same Digit
int[] same_arr = new int[] { 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13 };
double minimum_distance9 = MinDistance(same_arr);
double minimum_distance10 = MinDistance2(same_arr);

Console.WriteLine("An array of SAME DIGIT: { 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13 }");
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance9);
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance10);

Console.WriteLine();

//Partially Sorted Array
int[] partial_arr = new int[] { 13, 15, 17, 19, 27, 45, 11, 34, 31, 149 };
double minimum_distance11 = MinDistance(partial_arr);
double minimum_distance12 = MinDistance2(partial_arr);

Console.WriteLine("An array of PARTIALLY SORTED: {13, 15, 17, 19, 27, 45, 11, 34, 31, 149}");
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance11);
Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance12);

Console.WriteLine();
```



```
    //Array consisting of both negative and positve integars
    int[] mixed_arr = new int[] { -99, 19, 17, 78, 2887, 45, -1, 34, -231, -1049 };
    double minimum_distance13 = MinDistance(mixed_arr);
    double minimum_distance14 = MinDistance2(mixed_arr);

    Console.WriteLine("An array of POSITIVE & NEGATIVE INTEGERS: { -99, 19, 17, 78, 2887, 45, -1, 34, -231, -1049}");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance13);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance14);

    Console.WriteLine();

    //Array consisting of only  negative integars
    int[] negative_arr = new int[] { -23, -99, -735, -44, -4, -1111, -88, -5324, -122, -12 };
    double minimum_distance15 = MinDistance(negative_arr);
    double minimum_distance16 = MinDistance2(negative_arr);

    Console.WriteLine("An array of ONLY NEGATIVE INTEGERS:  -23,-99,-735, -44, -4,-1111,-88, -5324, -122, -12");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance15);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance16);

    Console.WriteLine();

    //Array consisting of very large and very small numbers
    int[] large_difference_arr = new int[] {1, 2100000000};
    double minimum_distance17 = MinDistance(large_difference_arr);
    double minimum_distance18 = MinDistance2(large_difference_arr);

    Console.WriteLine("An array of LARGE and SMALL INTEGERS:  1, 2100000000");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance17);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance18);

    Console.WriteLine();

    //Array consisting of only one element
    int[] small_arr = new int[] {1};
    double minimum_distance19 = MinDistance(small_arr);
    double minimum_distance20 = MinDistance2(small_arr);

    Console.WriteLine("An array of only one INTEGER: 1");
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance: {0}", minimum_distance19);
    Console.WriteLine("Minimum Distance between two elements using Algorithm MinDistance2: {0}", minimum_distance20);
    Console.WriteLine();
```

Output for functionality correctness testing



```
An array of SORTED NUMBERS: { 324,292,131,123,88,57,27,23,20,17,15,13,5}
Minimum Distance between two elements using Algorithm MinDistance: 2
Minimum Distance between two elements using Algorithm MinDistance2: 2

An array of SAME DIGIT: { 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13 }
Minimum Distance between two elements using Algorithm MinDistance: 0
Minimum Distance between two elements using Algorithm MinDistance2: 0

An array of PARTIALLY SORTED: {13, 15, 17, 19, 27, 45, 11, 34, 31, 149}
Minimum Distance between two elements using Algorithm MinDistance: 2
Minimum Distance between two elements using Algorithm MinDistance2: 2

An array of POSITIVE & NEGATIVE INTEGERS: { -99, 19, 17, 78, 2887, 45, -1, 34, -231, -1049}
Minimum Distance between two elements using Algorithm MinDistance: 2
Minimum Distance between two elements using Algorithm MinDistance2: 2

An array of ONLY NEGATIVE INTEGERS:  -23,-99,-735, -44, -4,-1111,-88, -5324, -122, -12
Minimum Distance between two elements using Algorithm MinDistance: 8
Minimum Distance between two elements using Algorithm MinDistance2: 8

An array of LARGE and SMALL INTEGERS:  1, 2100000000
Minimum Distance between two elements using Algorithm MinDistance: 2099999999
Minimum Distance between two elements using Algorithm MinDistance2: 2099999999

An array of only one INTEGER: 1
Minimum Distance between two elements using Algorithm MinDistance: ∞
Minimum Distance between two elements using Algorithm MinDistance2: ∞
```