Spatial and Temporal Analysis

Emmanuel Ameh

Conducting Spatial and Temporal Analysis

In writing and implementing code solutions for the assignment, I decided on using the C#

programming language. C# is a high-level programming language. I performed spatial and

temporal analysis paying close attention to algorithmic approach and underlying data structure to

implement the most effective and efficient algorithms. I downloaded and installed Microsoft

Visual Studio (.NET environment). I created the programs described and evaluated their

algorithms and data structures.

**C# Code for Random number assignment to an Array size of 10,000**

using System;

namespace Assignment\_1

{

class Program

{

static void Main(string[] args)

{

DateTime currentDate = DateTime.Now;

long StartTime = currentDate.Ticks;

//Algorithm statements for program that assigns 10,000 random numbers to an array,

{

int[] randomNum = new int[10000];

Random RandomNumber = new Random();

for (int i = 0; i < 10000; i++)

{

randomNum[i] = RandomNumber.Next(1, 10000);

}

foreach (int j in randomNum)

{

Console.WriteLine("The next of 10,000 random numbers is:{0}", j);

}

Console.Read();

}

long EndTime = DateTime.Now.Ticks;

long elapsedTicks = DateTime.Now.Ticks - currentDate.Ticks;

TimeSpan elapsedSpan = new TimeSpan(elapsedTicks);

// TimeSpan interval = EndTime - StartTime;

// Console.WriteLine(" {0:NO} ticks", elapsedTicks);

Console.WriteLine(" The number of ticks required to complete the process without a sorting algorithm is:{0}", elapsedTicks);

Console.WriteLine(" The number of seconds required to complete the process without a sorting algorithm is:{0}", elapsedTicks/10000000, "seconds");

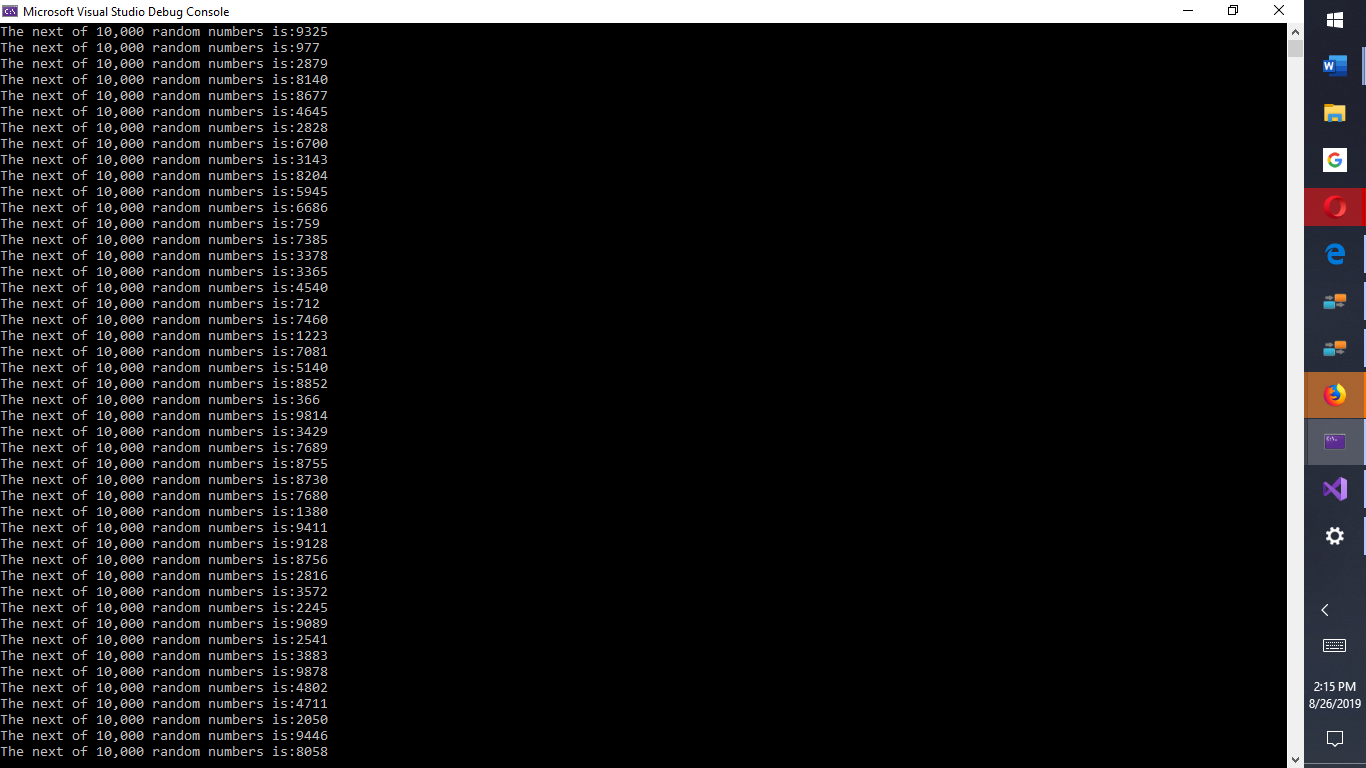
Console.WriteLine("Hello World!");

}

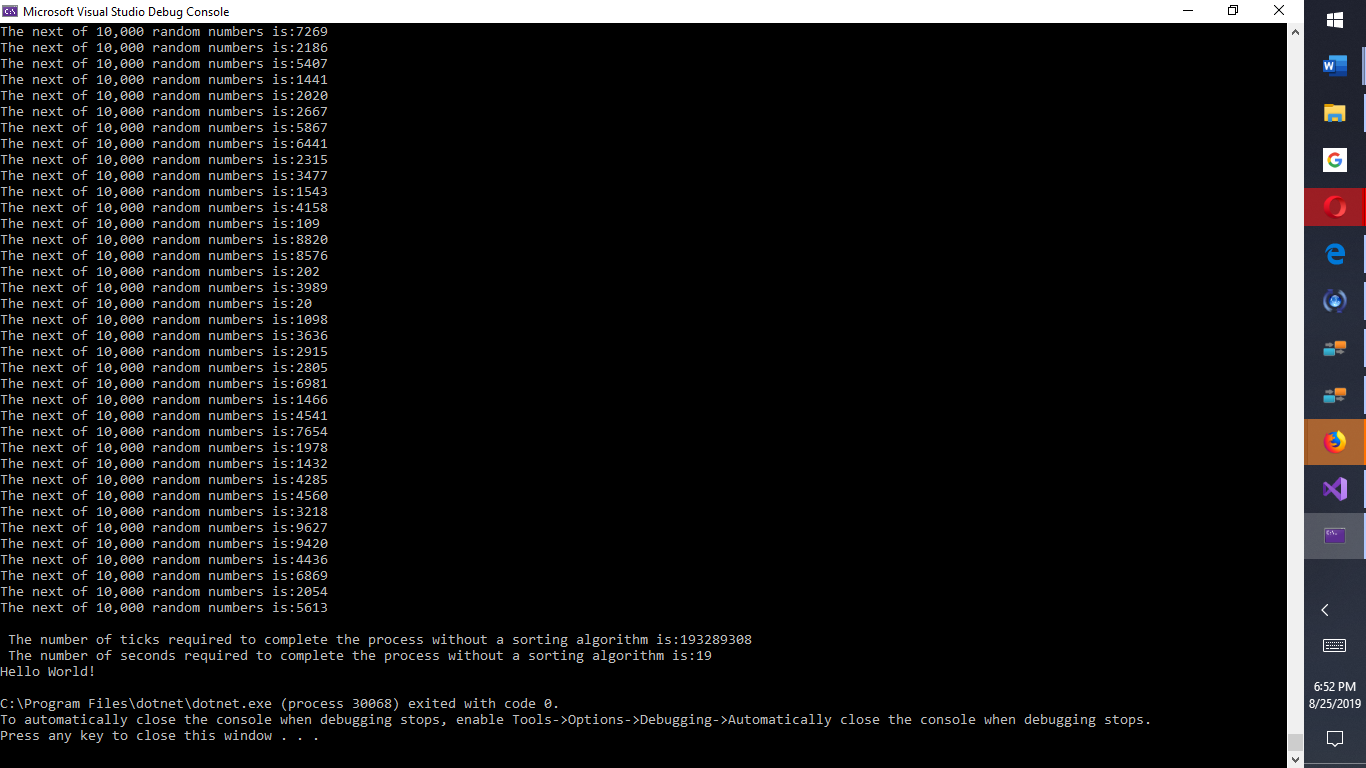
}

}

**Debugger Console Output**



*Figure 1.* 10,000 random number generation output.



*Figure 2.* 10,000 random number generation output, including temporal analysis.

**C# Code for Bubble Sort Algorithm**

using System;

namespace BubbleSortAlgorithm

{

class Program

{

private const string Format = "The next of 10,000 random numbers is:{0}";

#pragma warning disable IDE0060 // Remove unused parameter

public static void Main()

#pragma warning restore IDE0060 // Remove unused parameter

{

DateTime currentDate = DateTime.Now;

long StartTime = currentDate.Ticks;

{

int[] randomNum = new int[10000];

Random RandomNumber = new Random();

for (int i = 0; i < 10000; i++)

{

randomNum[i] = RandomNumber.Next(1, 10000);

}

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

{

int j = randomNum[i];

Console.WriteLine(

Format, j);

}

Console.Read();

long EndTime = DateTime.Now.Ticks;

long elapsedTicks = DateTime.Now.Ticks - currentDate.Ticks;

TimeSpan elapsedSpan = new TimeSpan(elapsedTicks);

// TimeSpan interval = EndTime - StartTime;

// Console.WriteLine(" {0:NO} ticks", elapsedTicks);

Console.WriteLine(" The number of ticks required to complete the process without a sorting algorithm is:{0}", elapsedTicks);

Console.WriteLine(" The number of seconds required to complete the process without a sorting algorithm is:{0}", elapsedTicks / 10000000, "seconds");

// Console.WriteLine(" {0, -35} {1,20:NO}", "Ticks:", interval.Ticks);

Console.WriteLine("Hello World!");

}

}

#pragma warning disable IDE0060 // Remove unused parameter

static void **BubbleSort**(int[] arr, int n)

#pragma warning restore IDE0060 // Remove unused parameter

{

if (arr is null)

{

throw new ArgumentNullException(nameof(arr));

}

//BubbleSort(randomNum, RandomNumber);

int i, j, temp;

bool swapped;

for (i = 0; i < 9999; i++)

{

swapped = false;

for (j = 0; j < 10000 - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

// swap arr[j] and arr[j+1]

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

// IF no two elements were swapped by inner loop, then break

if (swapped == false)

break;

}

}

private static void BubbleSort(int[] randomNum, Random RandomNumber)

{

for (int i = 0; i < 10000; i++)

{

randomNum[i] = RandomNumber.Next(1, 10000);

}

}

//Function to print an array

static void PrintArray(int[] arr, int n=10\_000)

{

int i;

for (i = 0; i < 10\_000; i++)

Console.Write(arr[i] + " ");

Console.WriteLine();

}

// Driver method

public static void Main(int[] j)

{

int[] arr = j;

int n = arr.Length;

BubbleSort(arr, n);

Console.WriteLine("Sorted array:");

PrintArray(arr, 10000);

// private static void Main(string[] args)

// static void BubbleSort(int[] arr, int n)

//{

// DateTime currentDate = DateTime.Now;

//long StartTime = currentDate.Ticks;

//}

}

}

}

**Steps 1 and 2 repeated for quick sort and insertion sort**

C# **Code for Sorting Algorithms**

using System;

namespace Assignment1.\_2

{

public class Solution

{

static void Main(string[] args)

{

{

{

//to create a Random class object

Random rand = new Random();

int[] arr1 = new int[10000];

int[] arr2 = new int[10000];

int[] arr3 = new int[10000];

///int index = 0, i;

/// fill the array with random integers

/// while (index < 10000)

{

// to create a random number

/// int x = rand.Next();

// check to see if the element is already present in the array

/// for (i = 0; i < index; i++)

for (int i = 0; i < 10000; i++)

{

arr1[i] = rand.Next(1, 10000);

arr2[i] = rand.Next(1, 10000);

arr3[i] = rand.Next(1, 10000);

}

foreach (int j in arr1)

{

Console.WriteLine("The next of 10,000 random numbers for quick sort is:{0}", j);

}

Console.Read();

foreach (int j in arr2)

{

Console.WriteLine("The next of 10,000 random numbers for bubble sort is:{0}", j);

}

Console.Read();

foreach (int j in arr3)

{

Console.WriteLine("The next of 10,000 random numbers for insertion sort is:{0}", j);

}

Console.Read();

TimeSpan ts = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc));

// get the starting time

long start = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

// sort array using selection sort

// get the ending time

long end = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

long total\_time = end - start;

{

Console.WriteLine("Time Taken by Quick Sort:{0}", total\_time, "mins");

}

Console.Read();

// get the starting time

start = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

// sort array using quick sort

selection\_sort\_array(arr1, 10000);

bubble\_sort\_array(arr2, 10000);

// get the ending time

end = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

total\_time = end - start;

Console.WriteLine("Time Taken by Bubble Sort:{0}", total\_time, "mins");

Console.Read();

// get the starting time

start = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

// sort array using selection sort

insertion\_sort\_array(arr3, 10000);

// get the ending time

end = (long)ts.TotalMilliseconds; //System.DateTime.Now.ToUniversalTime();

total\_time = end - start;

{

Console.WriteLine("Time Taken by Insertion Sort :{0}", total\_time, "mins");

}

Console.Read();

}

#pragma warning disable CS8321 // Local function is declared but never used

static void quick\_sort\_array(int[] arr1, int n)

#pragma warning restore CS8321 // Local function is declared but never used

{

int i, j, min, x;

// traverse the array

for (i = 0; i < n - 1; i++)

{

// store the index of the smallest element

min = i;

// traverse the array from the element after i till the end

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

{

// if the current element is smaller than the min element

if (arr1[j] < arr1[min])

min = j;

}

// Swap first element and min

x = arr1[min];

arr1[min] = arr1[i];

arr1[i] = x;

}

for (int i1 = 0; i1 < arr1.Length; i1++)

{

\_ = arr1[i1];

Console.WriteLine("Quick sort array is:{0}", i1);

}

Console.Read();

}

#pragma warning disable CS8321 // Local function is declared but never used

static void bubble\_sort\_array(int[] arr2, int n)

#pragma warning restore CS8321 // Local function is declared but never used

{

int i, j, x;

// loop from first till the second last element

for (i = 0; i < n - 1; i++)

{

for (j = 0; j < n - i - 1; ++j)

{

// if the next element is smaller than the current element

if (arr2[j + 1] < arr2[j])

{

// swap the j and j + 1 th element of a

x = arr2[j];

arr2[j] = arr2[j + 1];

arr2[j + 1] = x;

}

}

}

for (int i2 = 0; i2 < arr2.Length; i2++)

{

\_ = arr2[i2];

Console.WriteLine("Bubble sorted array is:{0}", i2);

}

Console.Read();

}

#pragma warning disable CS8321 // Local function is declared but never used

static void insertion\_sort\_array(int[] arr3, int n)

#pragma warning restore CS8321 // Local function is declared but never used

{

int i, j, temp;

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

{

// store the current element in temp

temp = arr3[i];

// set j to i - 1

j = i - 1;

// all elements greater than temp are

// shifted 1 position right

while (j >= 0 && arr3[j] > temp)

{

// shifted 1 position right

arr3[j + 1] = arr3[j];

j--;

}

arr3[j + 1] = temp;

}

for (int i3 = 0; i3 < arr3.Length; i3++)

{

\_ = arr3[i3];

Console.WriteLine("Insertion sorted array is:{0}", i3);

}

Console.Read();

}

}

}

}

}

}

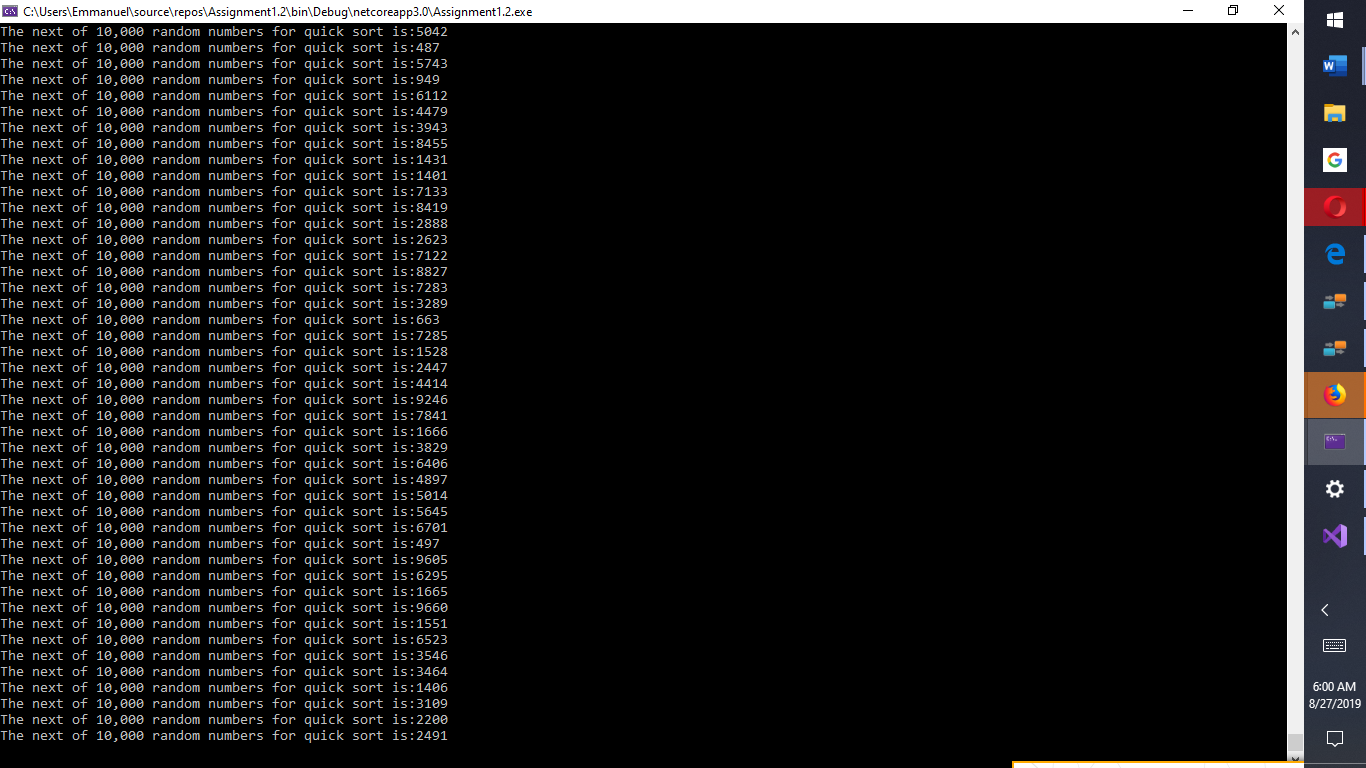
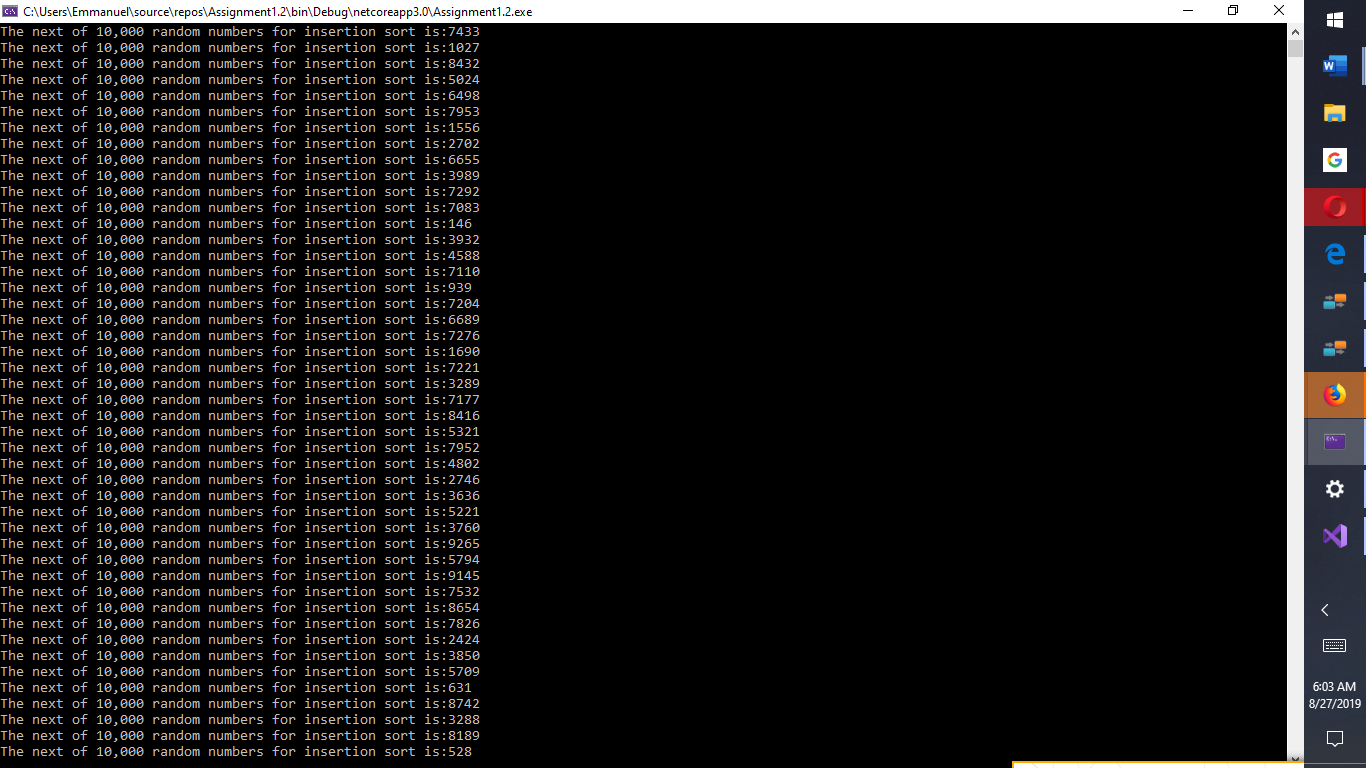


Fig 3. 10,000 random numbers for quick sorting



*Fig 4*. 10,000 random numbers for insertion sorting

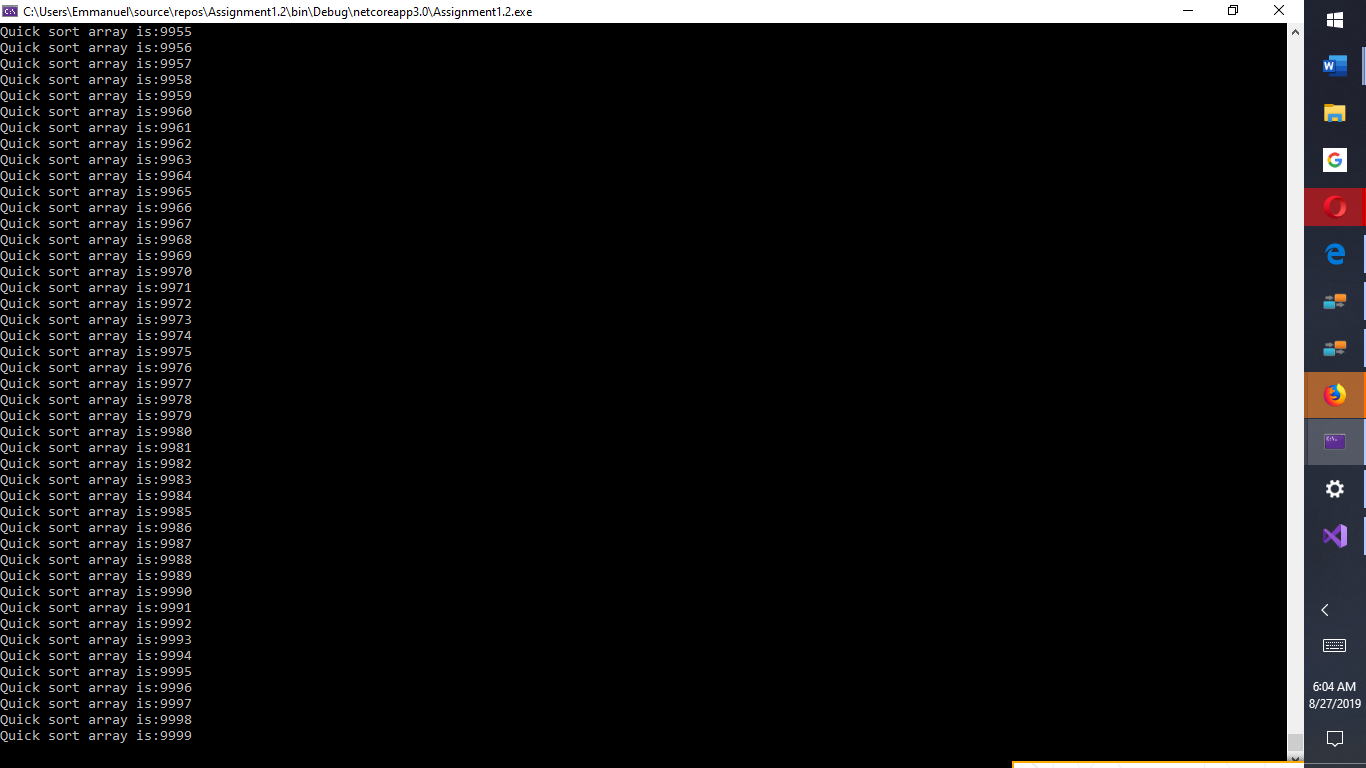
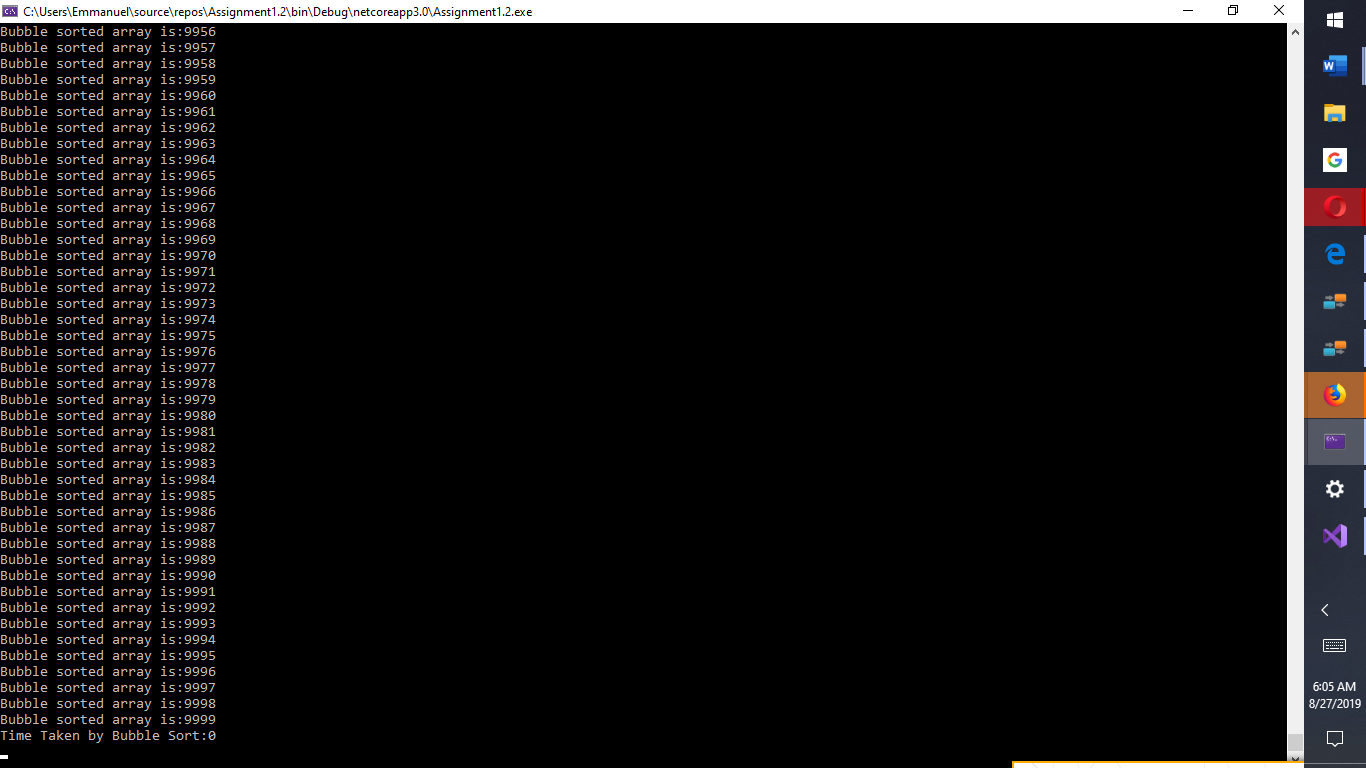
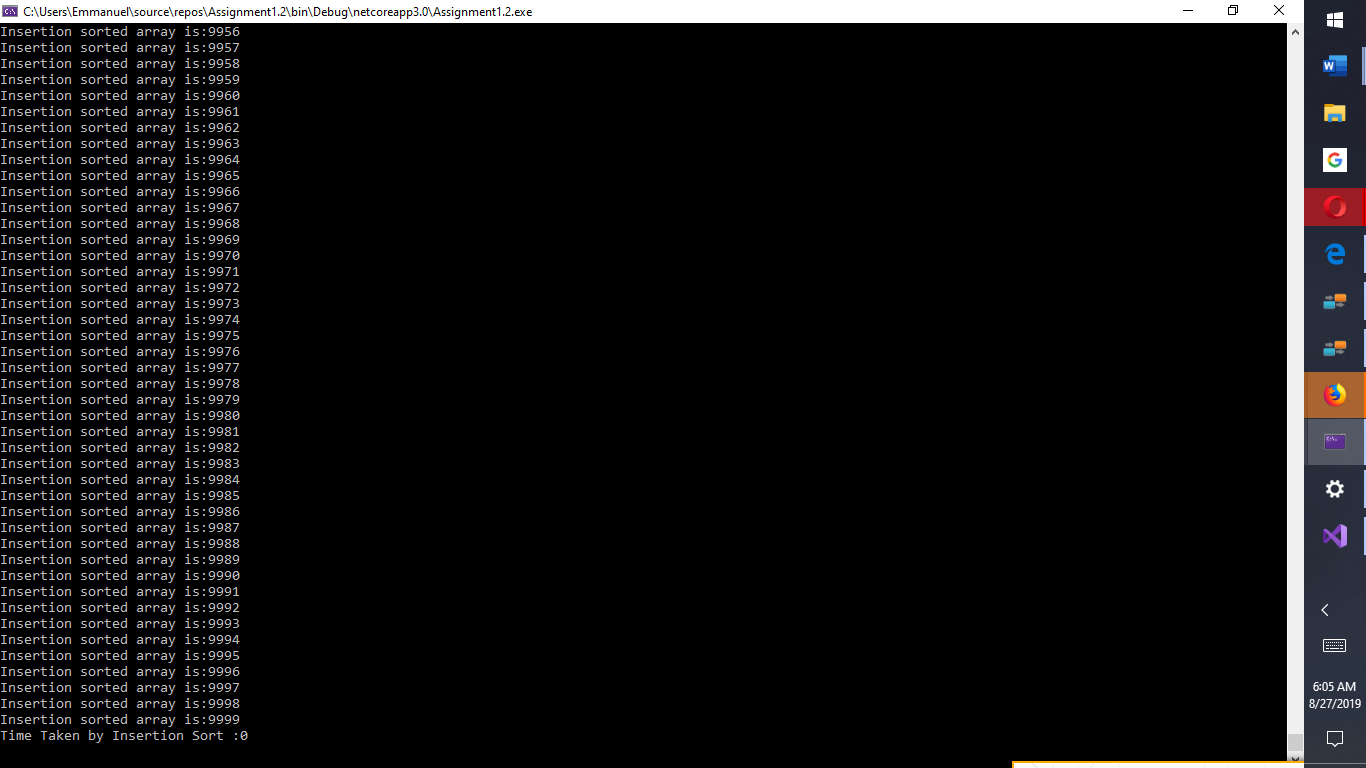


Fig. 5. Quick sorted array of 10,000 random numbers



*Fig 6*. Bubble Sorted array of 10,000 random numbers



*Fig 7*. Insertion Sorted array of 10,000 random numbers



*Fig 8*. Insertion Sorted array of 10,000 random numbers

**Steps 1 and 2 repeated for array of size 50**

The following lines of code were modified to implement algorithm for array size of 50.

to create a Random class object

Random rand = new Random();

int[] arr1 = new int[50];

int[] arr2 = new int[50];

int[] arr3 = new int[50];

///int index = 0, i;

/// fill the array with random integers

/// while (index < 50)

{

// to create a random number

/// int x = rand.Next();

// check to see if the element is already present in the array

/// for (i = 0; i < index; i++)

for (int i = 0; i < 50; i++)

{

arr1[i] = rand.Next(1, 50);

arr2[i] = rand.Next(1, 50);

arr3[i] = rand.Next(1, 50);

}

foreach (int j in arr1)

{

Console.WriteLine("The next of 50 random numbers for quick sort is:{0}", j);

}

Console.Read();

foreach (int j in arr2)

{

Console.WriteLine("The next of 50 random numbers for bubble sort is:{0}", j);

}

Console.Read();

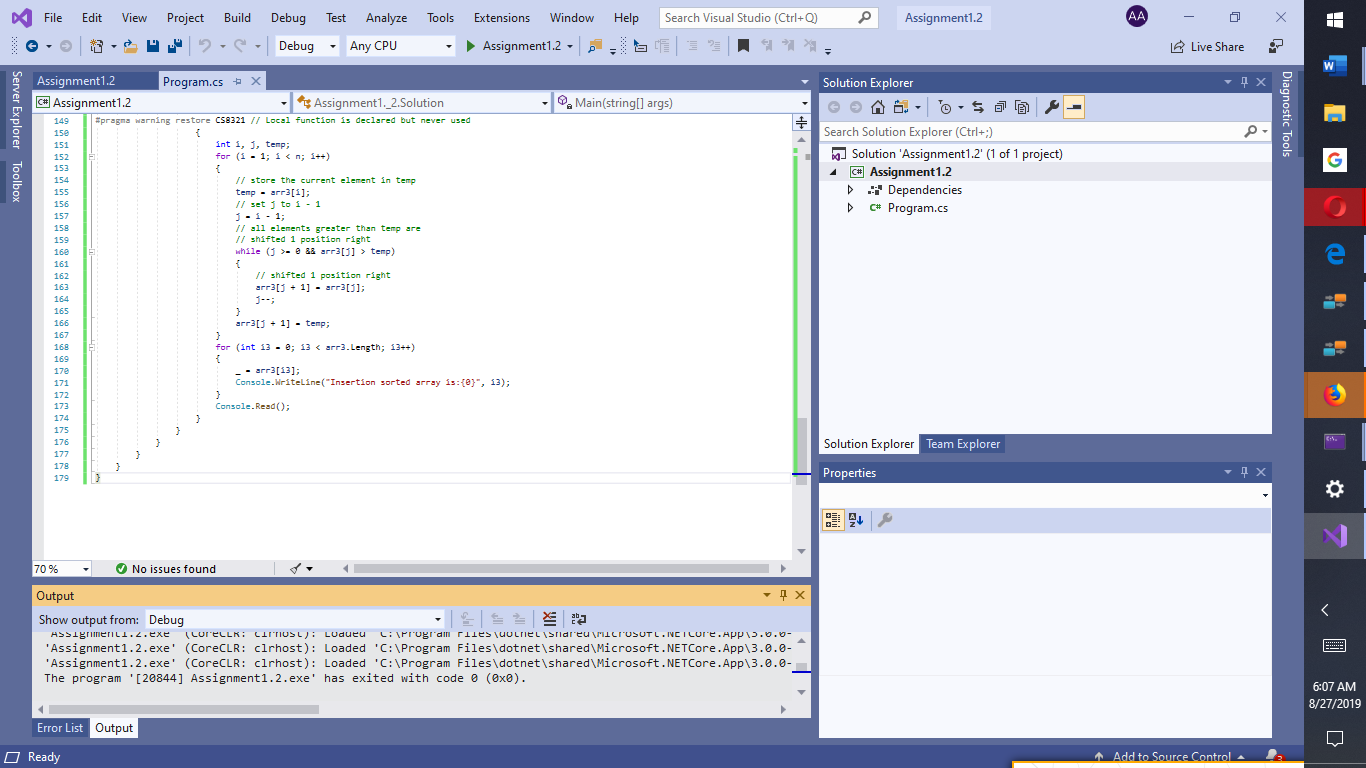
foreach (int j in arr3)

{

Console.WriteLine("The next of 50 random numbers for insertion sort is:{0}", j);

}

Console.Read();



*Fig 9*. Visual Studio Solution Explorer for C# programming

**Part 2 Spatial Analysis**

**10,000-element array to count each number’s occurrences with temporal and spatial analysis.**

**C# Code**

using System;

using System.Collections.Generic;

using System.Linq;

namespace SpatialAnalysis2

{

class Program

{

private const long maxGarbage = 10000;

static void Main(string[] args)

{

Program myProgram = new Program( );

//Method to obtain Total Memory Usage: Memory = System.GC.GetTotalMemory(false);

DateTime currentDate = DateTime.Now;

long StartTime = currentDate.Ticks;

//Algorithm statements for program that assigns 10,000 random numbers to an array, here

{

int[] randomNum = new int[10000];

Random RandomNumber = new Random();

for (int i = 0; i < 10000; i++)

{

randomNum[i] = RandomNumber.Next(1, 10000);

}

foreach (int j in randomNum)

{

Console.WriteLine("The next of 10,000 random numbers is:{0}", j);

}

Console.Read();

foreach (int j in randomNum)

{

randomNum[j]++;

}

//to access the counts

for (int i = 0; (i < 10000); i++)

{

Console.WriteLine((""

+ (i + " occurs "

+ (randomNum[i] + " times"))));

}

Console.Read();

for (int x = 0; (randomNum[x] > 500); x++)

{

Console.WriteLine(("" + (x + " occurs " + (randomNum[x] + " times"))));

}

Console.Read();

long EndTime = DateTime.Now.Ticks;

long elapsedTicks = DateTime.Now.Ticks - currentDate.Ticks;

TimeSpan elapsedSpan = new TimeSpan(elapsedTicks);

// TimeSpan interval = EndTime - StartTime;

// Console.WriteLine(" {0:NO} ticks", elapsedTicks);

Console.WriteLine(" The number of ticks required for 10000 element array to count each element's occurrence:{0}", elapsedTicks);

Console.WriteLine(" The number of seconds required to complete the process without a sorting algorithm is:{0}", elapsedTicks / 10000000, "seconds");

//To obtain Total Memory Usage

Console.WriteLine("Total Memory Usage to complete the process:{0}", GC.GetTotalMemory(false));

Console.Read();

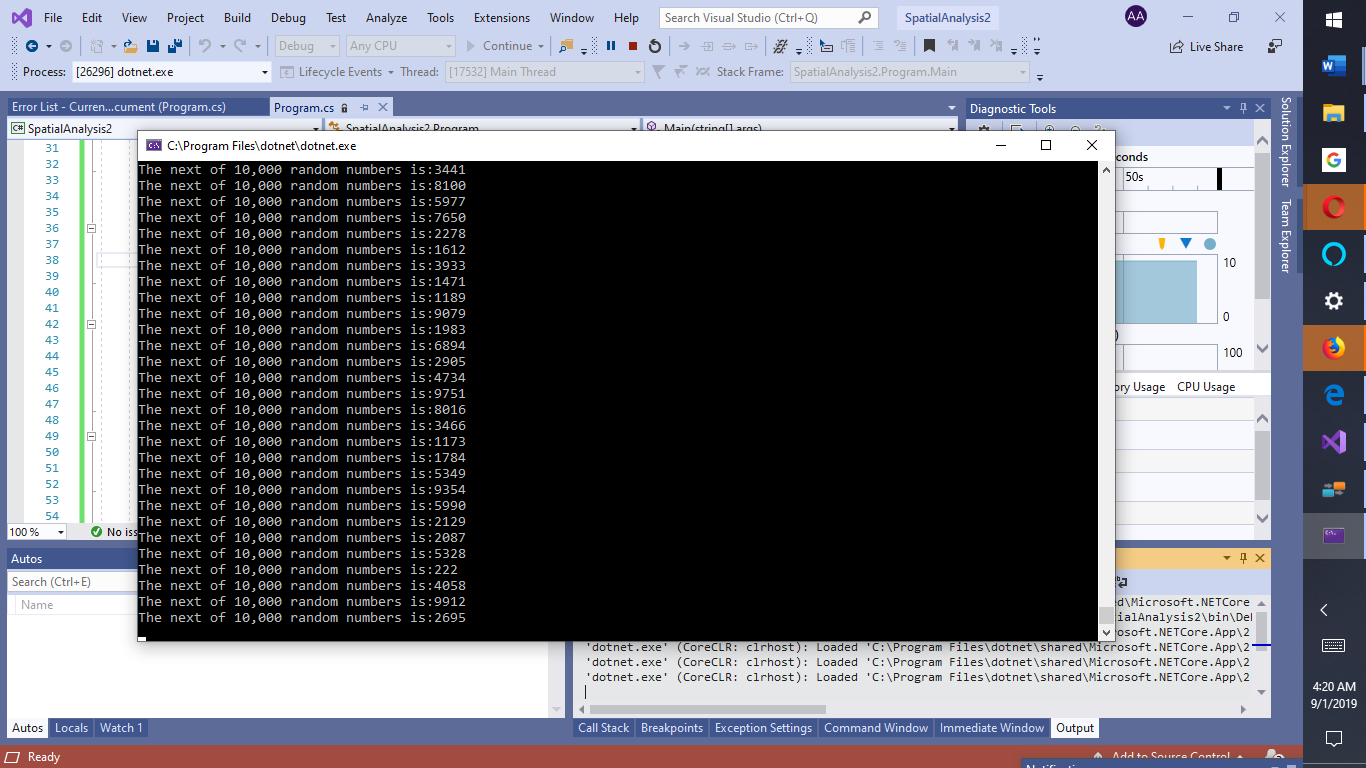
}

}

}

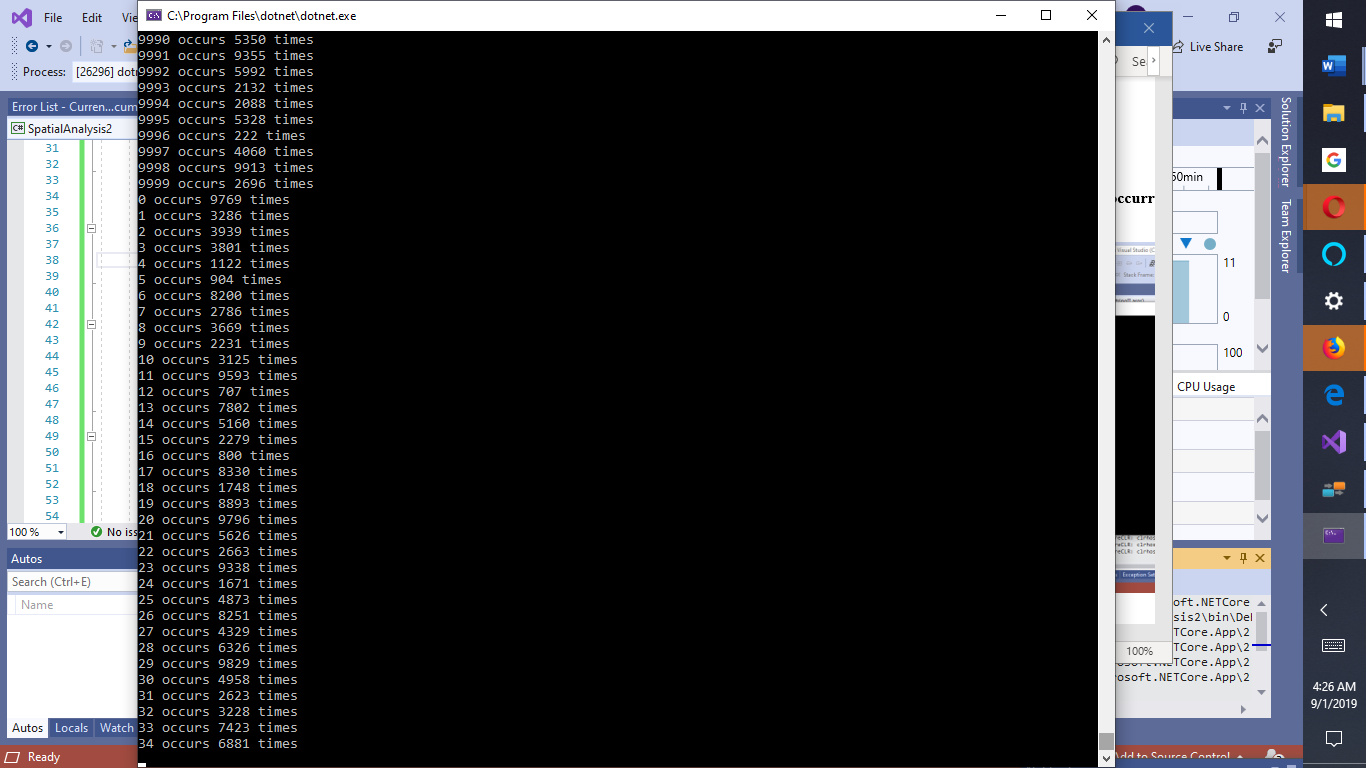
}

**Console Output for Array of 10,000 to determine frequencies/occurrences**



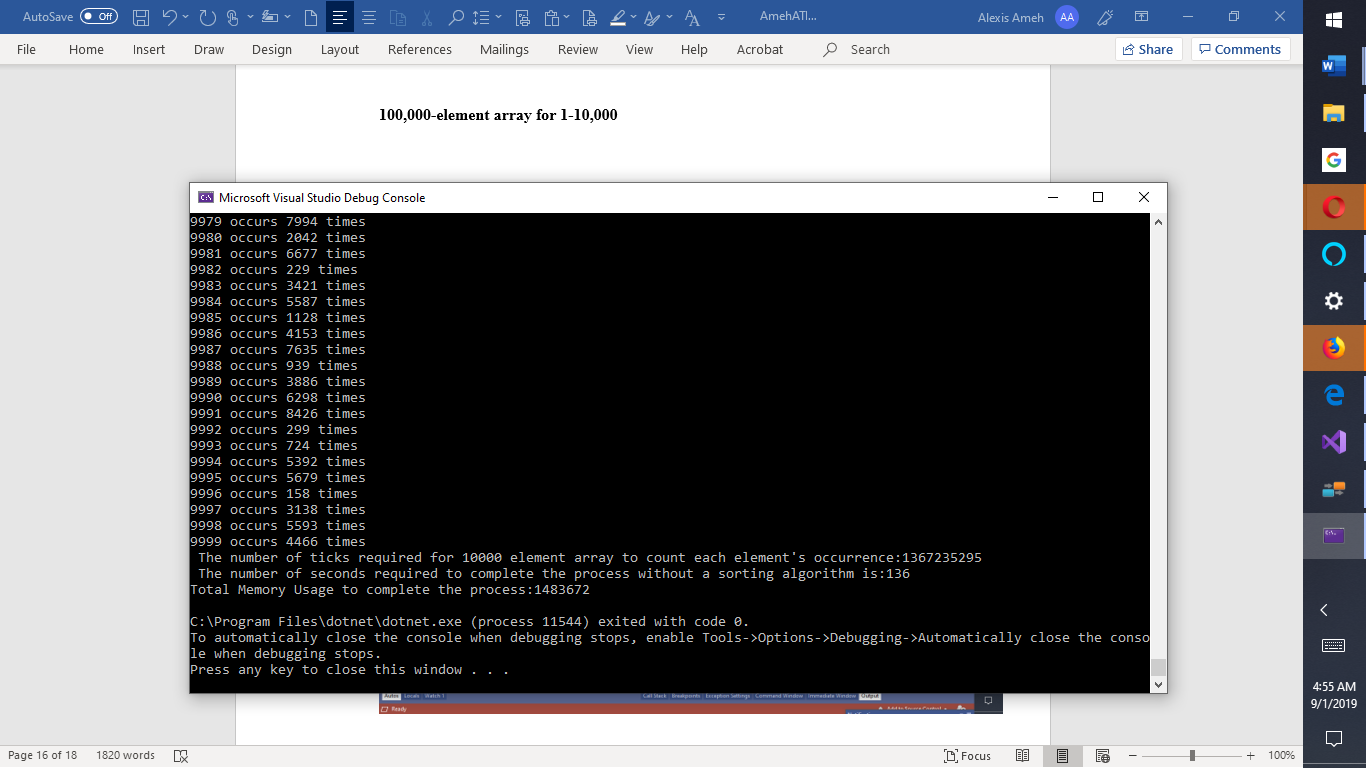
*Fig 10*. Array of 10,000 to determine frequencies/occurrences

**Console Output for Frequencies for Array of element 10,000**



*Fig 11.* Number of occurrences of 10,000 array elements

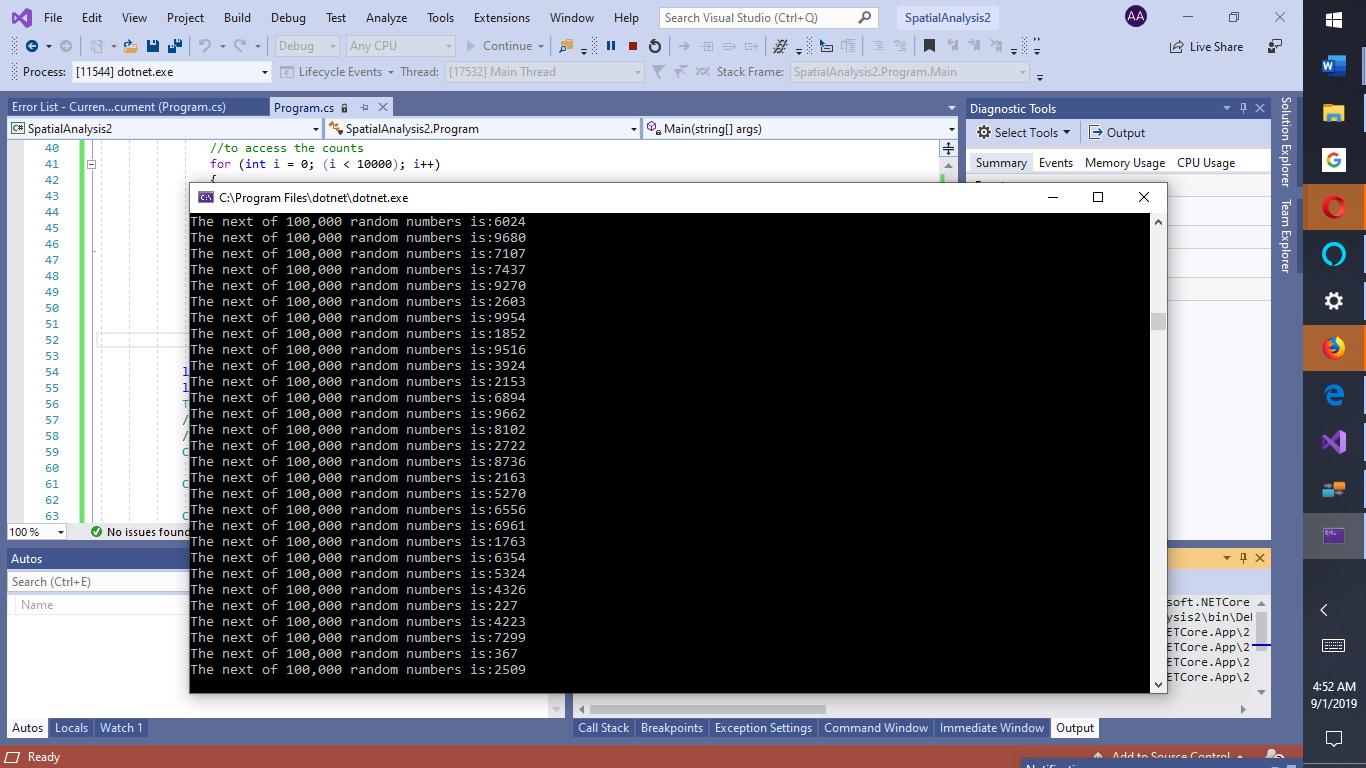
**Console Output for Array of 10,000 to determine frequencies/occurrences**



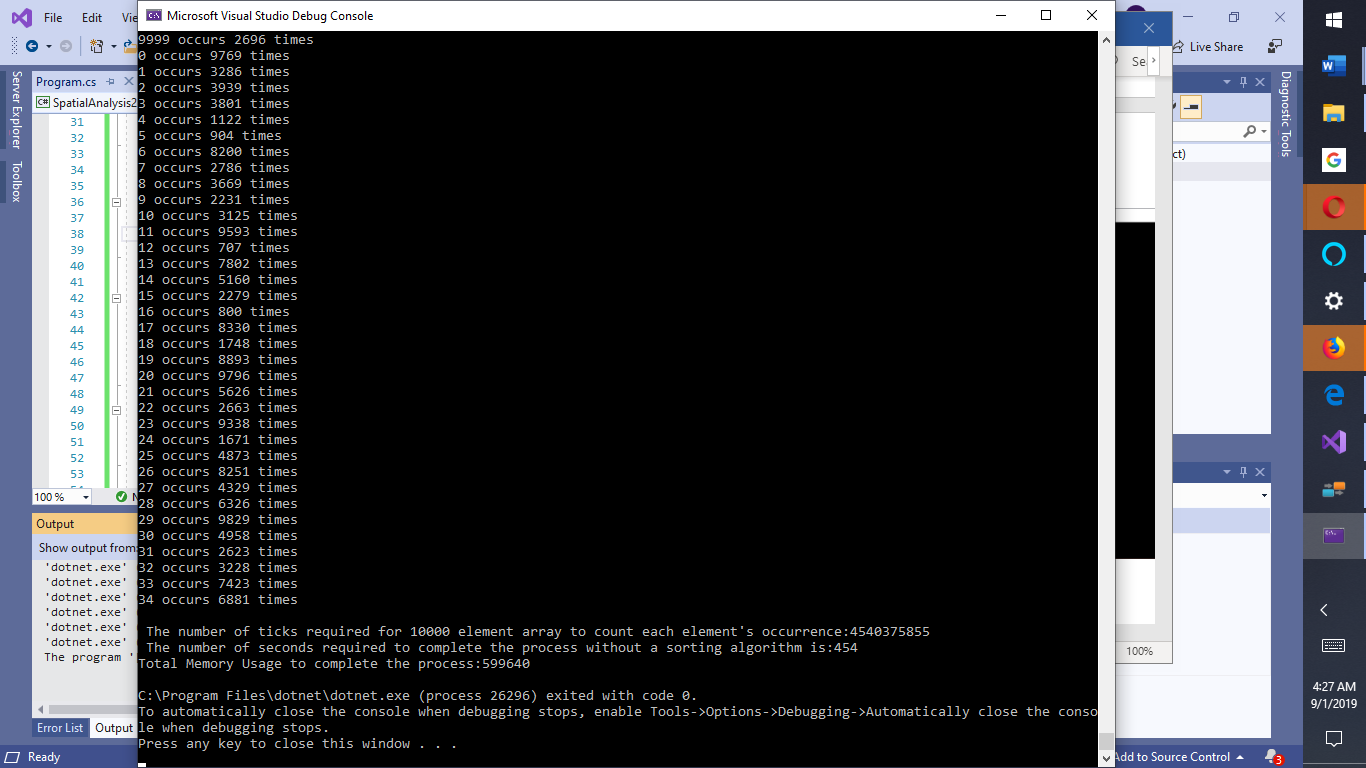
*Fig 12.* Console Output for Array of 10,000 to determine frequencies/occurrences with memory

1483672 and temporal analysis of 136 seconds

**Console Output for 100,000-element array for range 1-10,000**



*Fig 13.*100,000-element array for range 1-10,000



*Fig 14.*Number of occurrences of 100,000 array-elements in range 1-10000 with spatial

and temporal analysis **with results 454 secs and memory 500640.**

**Heap Sort Code for Spatial Analysis**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Spatial\_Analysis

{

class Program

{

#pragma warning disable IDE0060 // Remove unused parameter

static void Main(string[] args)

#pragma warning restore IDE0060 // Remove unused parameter

{

if (args is null)

{

throw new ArgumentNullException(nameof(args));

}

}

// if (args is null)

//{

// throw new ArgumentNullException(nameof(args));

//}

//}

// using System.Collections.Generic;

public virtual IList<int> TopKFrequent(int[] nums, int k)

{

//count the frequency for each element

Dictionary<int, int> map = new Dictionary<int, int>();

foreach (int num in nums)

{

map[num] = map.GetValueOrDefault(num, 0) + 1;

}

// create a min heap

PriorityQueue<KeyValuePair<int, int>> queue = NewMethod();

//maintain a heap of size k = 10.

foreach (KeyValuePair<int, int> entry in map.SetOfKeyValuePairs())

{

queue.offer(entry);

if (queue.size() > k)

{

queue.poll();

}

}

//get all elements from the heap

IList<int> result = new List<int>();

while (queue.size() > 0)

{

//result.Add(queue.poll().Key);

}

//reverse the order

result.Reverse();

return result;

//}

{

}

}

private PriorityQueue<KeyValuePair<int, int>> NewMethod()

{

throw new NotImplementedException();

}

//private static PriorityQueue<KeyValuePair<int, int>> NewMethod()

//{

// int IComparer.compare(object e, object e.Value);

// return new PriorityQueue<KeyValuePair<int, int>>(System.Collections.IComparer.compare(e => e.Value));

}

}

//Helper class added by Java to C# Converter:

//---------------------------------------------------------------------------------------------------------

// Copyright © 2007 - 2019 Tangible Software Solutions, Inc.

// This class can be used by anyone provided that the copyright notice remains intact.

//

// This class is used to replace calls to some Java HashMap or Hashtable methods.

//---------------------------------------------------------------------------------------------------------

//using System.Collections.Generic;

namespace Spatial\_Analysis

{

internal static class HashMapHelper

{

public static HashSet<KeyValuePair<TKey, TValue>> SetOfKeyValuePairs<TKey, TValue>(this IDictionary<TKey, TValue> dictionary)

{

HashSet<KeyValuePair<TKey, TValue>> entries = new HashSet<KeyValuePair<TKey, TValue>>();

foreach (KeyValuePair<TKey, TValue> keyValuePair in dictionary)

{

entries.Add(keyValuePair);

}

return entries;

}

public static TValue GetValueOrNull<TKey, TValue>(this IDictionary<TKey, TValue> dictionary, TKey key)

{

dictionary.TryGetValue(key, out TValue ret);

return ret;

}

}

}

**Explanations for Sorting Algorithms with reduced array size**

quick\_sort\_array(arr1, 50);

bubble\_sort\_array(arr2, 50);

insertion\_sort\_array(arr3, 50);

When repeated for an array of 50 numbers, all the sorting algorithms were quicker to

process/implement. This is because, the algorithm has less data in the array to traverse. Quick

Sort on the average is better than bubble sort but is not as effective as insertion sort. So,

insertion sort is the best and bubble sort is the worst sorting algorithm among the three sorting

algorithms. Insertion sort is faster for small n for instance n = 50, because Quick Sort has extra

overhead from the recursive function calls. Insertion sort is also more stable than Quick sort and

requires less memory. In heap sort, time complexity is O(n\*log(k)). Heap is often used to

reduce time complexity from n\*log(n) to n\*log(k) as depicted in code above.