Algorithm Implementation

The first algorithm I chose to implement in C# was insertion sort. Insertion sort is a sorting algorithm that builds a sorted array one step at a time. It goes through each element and checks to see if it should be moved to the left (if it is a smaller value). By doing this, a sorted array gets built over time. Here is the code that I wrote for insertion sort:

```
using System;

namespace algorithm1{

  class sortingAlgorithm{

    static void insertionSort(int[] nums, int size){

      for(int i = 1; i < size; i++){
        int value = nums[i];
        int j = i - 1;

        while(j >= 0 && nums[j] > value){
          nums[j+1] = nums[j];
          j--;
        }

        nums[j+1] = value;
      }
    }
```

This code will take a given array of integers and sort it from least to greatest. It starts with a for loop, which will iterate through each element in the given array. The integer "value" is used to hold the value of the current location in the array, and this is the element that is being checked. The integer "j" is used to access the location (or multiple locations) left of the current location in the array. After this, the algorithm goes into the while loop. The while loop checks to see if "j" is still in the appropriate range and if the value at location j is greater than the value being checked. If this is true, it executes the content of the while loops until it gets to the start of the array or an element that is less than the current element is found. Once it exits the while loop, the value being checked gets placed in the j+1 location in the array.

For example, say we are checking the 4th element for a given array of {4, 7, 14, 5, 9}. Following the code, we would check to see if the 3rd element is greater than the 4th element in the array. Since 14 is greater than 5, the value of 14 gets moved to the 4th location in the array, and we continue to check the current element. Now we check to see if the 2nd element is greater than the current element, which is still the value 5. Since 7 is greater than 5, the value of 7 gets moved to the 3rd location in the array, and we continue to check the current element. Now we check to see if the 1st element is greater than the current element, which is still the value 5. Since 4 is not greater than 5, the value of 5 gets placed in the 2nd location in the array. The end of the for loop is reached, and the algorithm would move to begin checking for the 5th element. After the pass, our array would look like this: {4, 5, 7, 14, 9}.

The second algorithm I chose to implement in C# was binary search. Binary search is a searching algorithm which finds a specified value in a sorted array. It does this recursively by dividing the search interval in half at each step. C# is language that is well suited for a search algorithm like binary search, and it is also good for others like breadth-first search and depth-first search. Here is the code that I wrote for the binary search algorithm:

```
using System;

namespace algo2{

  class searchAlgos{

    public static int binarySearch(int[] nums, int target, int left, int right){
      if(left <= right){
        int middle = left + (right - left) / 2;

        if(nums[middle] == target){
          return middle;
        }
        else if(nums[middle] < target){
          return binarySearch(nums, target, middle+1, right);
        }
        else{
          return binarySearch(nums, target, left, middle-1);
        }
      }
      return -1;
    }

```

The algorithm takes a sorted array, target number, left pointer location, and right pointer location when calling it. It starts by checking if the array is still being searched. If the left pointer value is greater than the right pointer value, then the target number wasn't found and the algorithm returns -1. If the target is still being looked for, it gets the middle locations value (between the left and right pointer) and checks to see if it is the target number. If this happens to be the target number, then the algorithm returns the location of this element in the given array. If it's not the target value, the algorithm checks to see if the target number would be further right or further left in the array. Once this is found out, it recursively calls itself until the target number is found or it has been deemed that the list does not contain the target number.

The third algorithm I chose to implement in C# was a simple LINQ program. In this program, a LINQ environment is set up to use over data. Bill Wagner's example from Microsoft Learn, Work with Language-Integrated Query (LINQ) (2021) helped structure my example. The two created classes for the data are buildings at Xavier University and the room numbers they have. The Main method generates each room with its building and room number and prints the result. The compiler translates LINQ statements written in query syntax into the appropriate method call syntax. Here is my code for this algorithm, which appears on the next page:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace linq{

  class linqExample{

    // All possible room numbers
    static IEnumerable<string> roomNumbers(){
      yield return "101";
      yield return "102";
      yield return "103";
      yield return "201";
      yield return "202";
      yield return "203";
      yield return "301";
    }

    // All possible buildings
    static IEnumerable<string> buildings(){
      yield return "Alter Hall";
      yield return "Smith Hall";
      yield return "Health United Building";
    }

    static void Main(string[] args){
      // Generate all possible rooms
      var sampleRooms = from b in buildings()
                        from n in roomNumbers()
                        select new {building = b, roomNumber = n};

      // Print each room that has been generated
      foreach(var room in sampleRooms){
        Console.WriteLine(room);
      }
```

First, the algorithm needs to identify that it is using LINQ by specifying it at the top, by saying “using System.linq”. The data classes use “yield return” to produce the sequence as they run. This is helpful when querying the data. The sampleRooms variable holds the query result from the two classes, which is then printed out by using the foreach loop. This is a very simple algorithm that uses LINQ. There can be many more specifications depending on the contents of the data sets, and programmers can also manipulate the order of the results depending on what they prefer.