Algorithms & Complexity: Assessment I

Samuel J. Orman-Chan (25659005)

April 28, 2022

Contents

Created Using LaTeX.

1 Design

In the beginning, there was nothing.

Then there was code.

The code was full of logic errors and was only capable of returning the 10$^{\text{th}}$ (with an offset of +1) values of the unsorted arrays. I am unable to comprehend why and therefore the algorithms are implemented but only linear search partially works but is unable to return the indexes. It was however, able to provide a count of the number of times an number was found in the array extracted from the text file. The program does however contain code to provide a QuickSort, a Bubble Sort & a Linear Search. Despite this, none, except the Linear Search are nearly functional.

The Bubble Sort Code was based on the code found here(Unknown, 2014, p.1). However it did not succeed in returning the sorted data. This was most likely a logic error owing to my poor understanding of the OOP paradigm. Despite this, the code for the Bubble Sort algorithm itself does look as if it should work properly, meaning that the issue lies in the support code to make the algorithm usable. The user interface is based around keyboard inputs, with the user having to press RETURN after selecting the file they wish to sort & search. However in the program, the file paths are hard-coded and the program assumes that the files will always remain at the same paths. This may lead to crashes if the file names or paths are changed however handling these kind of errors I feel is outside the scope of the program.

On the QuickSort side of things, I cannot tell if it works at all. This is as the code was not successfully returning the array sorted. Instead it just output the inputted array, apparently unchanged. My QuickSort was based laregly on pseudocode found on (*Quicksort*, 2022b, p.1). This meant that likely, I made many errors in the understanding and subsequent implementation of the code. In part as I used the 1$^{\text{st}}$ value as my partition.

The process of implementation was difficult and the outputs were evidently wrong. However, the file reading seemed to work fine, with all three files being properly read into their respective arrays without issue. This was achieved using a small helper function to convert the data provided by the “ReadAllLines” function into an integer array. This functioned via the use of the “Array.ConvertAll” and the “Int32.Parse” functions built into C#. This component was the only functional part of the whole program that I could see.

The outputting of every 10$^{\text{th}}$ value was achieved with a for loop with a step of 10. This part worked however, it was not outputting the sorted array, as noted previously.

2 Algorithm Choices and Results

The Algorithms were chosen primarily based on ease of implementation. This is as I struggled with the C# language and as such, felt that simpler algorithms would make life a little easier. I chose to implement two sorting algorithms, QuickSort & Bubble Sort. However owing to problems with the program, I am unable to provide data on how many steps each algorithm takes to process an array of data and subsequently sort it. Therefore any information regarding these algorithms will be based entirely off descriptions found in other works as to their respective efficiencies. A consequence of this issue is that tables cannot be provided as the program would only return 1 for the number of steps taken, a glaring logic error. I speculate this may be down to issue with namespaces however I am not entirely certain.

What I can say is that the program is non-functional, and as a result, time and space efficiency are difficult to find. I can however comment that the Linear search did seem rather slow to run but as I cannot provide proper time data, I cannot corroborate this and this is entirely subjective. Despite this or rather owing to this issue, I can provide big O notation information from other sources. According to (Dorantes, 2022, p.1), the QuickSort algorithm should have a time complexity of $\Omega(\log_n)$ at best and at worse a time complexity of $O(n^2)$. The Bubble Sort on the other hand, would be expected to have at worst time complexity of $O(n^2)$ and at best a complexity of $\Omega(n)$. Meaning that, the QuickSort would in theory be typically a more time efficient algorithm to use.

The Linear Search would be expected to have a best case time complexity of $\Omega(1)$ and at worst $O(n)$. On average, a linear search would be assumed to an average efficiency of $\Theta(\frac{n+1}{2})$.(*Linear search*, 2022a, p.1).

Part of the intention with the Linear Search was to have a quick to implement example of a Searching Algorithm as well as one that could work with unsorted or duplicate data. Therefore resulting in the selection of Linear Search as a search algorithm. Despite these benefits, Linear Search is a very inefficient algorithm by most standards as referred to above with its $O(n)$ time complexity.

3 Appendix A — The Code.

```
using System.Linq;

class Program
{


    public static void Main()
    {


        string[] shares1_256 = System.IO.File.ReadAllLines(@"./Share_1_256.txt");
        string[] shares2_256 = System.IO.File.ReadAllLines(@"./Share_2_256.txt");
        string[] shares3_256 = System.IO.File.ReadAllLines(@"./Share_3_256.txt");
        int[] arry1 = Recast2Ints(shares1_256);
        int[] arry2 = Recast2Ints(shares2_256);
        int[] arry3 = Recast2Ints(shares3_256);
        Console.WriteLine("The first values of each unsorted array:");
        //Console.WriteLine(arry1[255] + " " + arry2[255] + " " + arry2[255] + " ");
        Console.Write("Select the array you want to search: 1, 2 or 3 ONLY.\n> ");
        string strUserChoice = Console.ReadLine();
        int intUserChoice = 0;
        try
        {
             intUserChoice = System.Convert.ToInt32(strUserChoice);
        }
        catch (ArgumentNullException)
        {
            ;

        }

        int[] arryChoice = new int[258];

        if (intUserChoice == 1)
        {
            arryChoice = arry1;
        }
        else if (intUserChoice == 2)
        {
            arryChoice = arry2;
        }
```

```
else if (intUserChoice == 3)
{
    arryChoice = arry3;
}
else
{
    ;
}

Console.WriteLine("For BubbleSort press B, for a
QuickSort that doesn’t work in the slightest, press Q.");

ConsoleKeyInfo m = Console.ReadKey();
int[] sortedD = arryChoice;
if (m.Key == ConsoleKey.B)
{
    sortedD = BubbleSort(arryChoice);
}
else if (m.Key == ConsoleKey.Q)
{
    sortedD = QuickSort(arryChoice);
}
else { Environment.Exit(0); }

for (int D = 0; D <= 256; D = D + 10)
{
    Console.WriteLine(sortedD[D]);
}
int[] revSortedD = new int[256];
for (int E = sortedD.Length - 1; E >= 0; E--)
{
    revSortedD[E] = sortedD[E];
}
Console.WriteLine("In descending order:");
for (int F = 0; F <= 256; F = F + 10)
{
    Console.WriteLine(revSortedD[F]);
}
//Console.WriteLine(arryChoice);
Console.Write("Enter your number to search for using a Linear Search.> ");
int intUserFind = System.Convert.ToInt32(Console.ReadLine());
int findWlLinear = LinearSearch(intUserFind, arryChoice);
//Console.Beep();
```

```
    Console.WriteLine();
    //Console.ReadKey();
}

public static int[] Recast2Ints(string[] sharesUSortedStr)
{
    int[] result = Array.ConvertAll(sharesUSortedStr, (s) => Int32.Parse(s));

    //List<int> list = sharesUSortedStr.ConvertAll<int>(s => Int32.Parse(s));

    //for (int i = 0; i < sharesUSortedStr.Length; i++)
    //{
    //    result[i] = Int32.Parse(sharesUSortedStr[i]);
    //}

    return result;
}

public static int LinearSearch(int searchDatum, int[] arraySearched)
{
    int stepsTaken = 0;
    int[] posLocates = new int[258];


    foreach (int i in arraySearched)
    {
        //Console.WriteLine(i);

        if (searchDatum == i)
        {
            Console.WriteLine("Found in List.");
            posLocates[stepsTaken] = 1;
        }
        else
        {
            //Console.WriteLine("Not Found.");
            posLocates[stepsTaken] = 0;
        }
        stepsTaken = stepsTaken + 1;

        //Console.WriteLine(stepsTaken);
    }
    //foreach (int i in posLocates)
    //{
    //    Console.WriteLine(i);
    //}

    int foundPoss = posLocates.Where(p => p == 1).Count();
    Console.WriteLine("Found in " + foundPoss + " location(s).");

    bool posFinders = false;
    int[] listOfPos;

    while (posFinders == false)
    {

    }

    return 1;
}

public static int[] BubbleSort(int[] sharesData)
{
    int stepsTaKen = 0;
    int arryLen = sharesData.Length;
    for (int iter2 = 0; iter2 < arryLen -1; iter2++)
    {
        for (int iter3 = 0; iter3 < arryLen - iter2 - 1; iter2++)
        {
            if (sharesData[iter3] > sharesData[iter3 + 1])
            {
                stepsTaKen++;
                int tempVar = sharesData[iter3];
                sharesData[iter3] = sharesData[iter3 + 1];
                sharesData[iter3 + 1] = tempVar;
            }
        }
    }
    Console.WriteLine("This took " + stepsTaKen + " to complete.");
    foreach (int k in sharesData)
    {
        Console.WriteLine(k);
    }
    return sharesData;
    }

    public static int[] QuickSort(int[] sharesData)
    {
        int stepsTaken = 0;
        int sortPoint1 = 0;
        int iter = 0;
        int sortPoint2 = sharesData.Length -1;
        bool sorted = false;
        int tempVar;
        int pivot = sharesData[0];
        while (sorted == false)
        {
            stepsTaken++;
            pivot = sharesData[iter];

            if (sharesData[sortPoint1] > sharesData[sortPoint2])
            {
                tempVar = sharesData[sortPoint2];
                sharesData[sortPoint2] = sharesData[sortPoint1];
                sharesData[sortPoint1] = tempVar;
            }
            else { sorted = true; }
            sortPoint1++;
            sortPoint2--;
            sharesData.Append(stepsTaken);
            return sharesData;
            iter++;
        }
        foreach (int k in sharesData)
        {
            Console.WriteLine(k);
        }

        return sharesData;


    }
}
```

4 References

References

Dorantes, C.A., 2022. *Quicksort algorithm* [Cesar's Tech Insights] [Online]. Available from: `https://medium.com/cesars-tech-insights/quicksort-17c5d24e7e5f` [Accessed April 27, 2022].

Wikipedia, 2022a. *Linear search* [Online]. Page Version ID: 1071025653. Available from: `https://en.wikipedia.org/w/index.php?title=Linear_search&oldid=1071025653` [Accessed April 27, 2022].

Wikipedia, 2022b. *Quicksort* [Online]. Page Version ID: 1080346337. Available from: `https://en.wikipedia.org/w/index.php?title=Quicksort&oldid=1080346337` [Accessed April 27, 2022].

Unknown, 2014. *Bubble sort* [GeeksforGeeks] [Online]. Available from: `https://www.geeksforgeeks.org/bubble-sort/` [Accessed April 27, 2022].