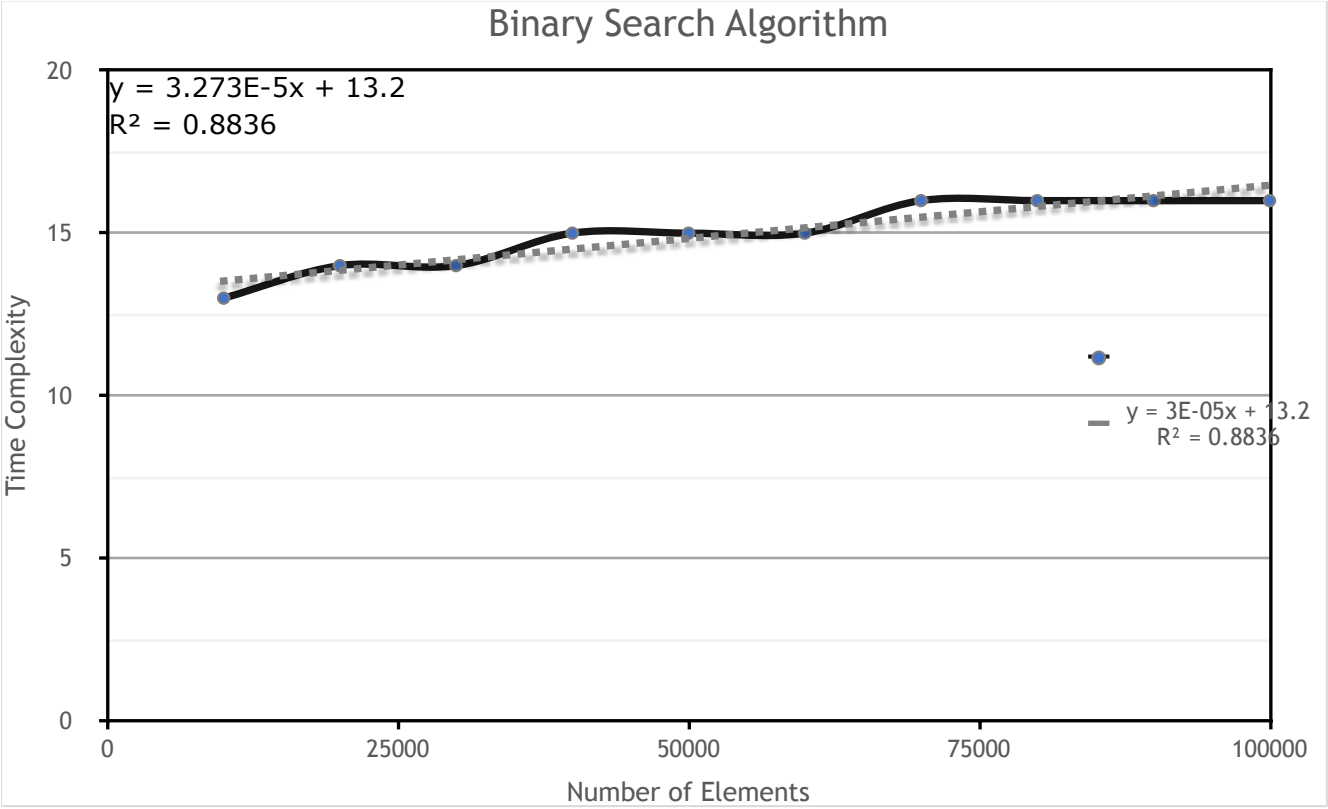
04 - Array Searching

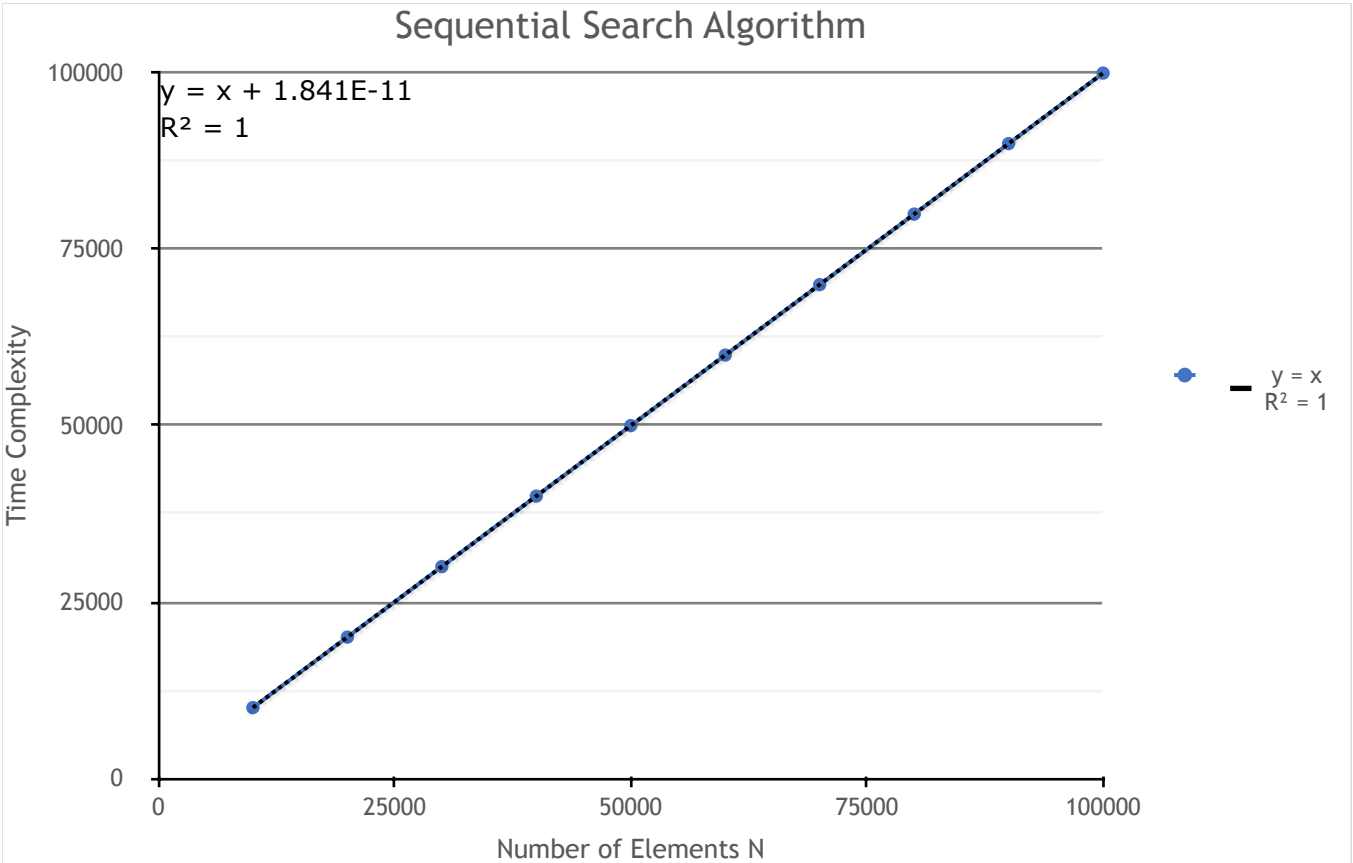
Post-Lab Exercises

Exercise 1

Using the program lab0403.cpp, find the worst case number of iterations for both the binary and sequential search algorithms for the range N = 10000 to N = 100000. Now using the data, apply regression analysis to find the equation for the curve for both the algorithms.

Worst case analysis of Binary and Sequential Search Algorithms		
Number of Elements	**Binary Search**	**Sequential Search**
10000	13	10000
20000	14	20000
30000	14	30000
40000	15	40000
50000	15	50000
60000	15	60000
70000	16	70000
80000	16	80000
90000	16	90000
100000	16	100000


Binary Search Algorithm
y = 3.273E-5x + 13.2
R² = 0.8836
Time Complexity
Number of Elements
0
5
10
15
20
0
25000
50000
75000
100000
y = 3E-05x + 13.2
R² = 0.8836


Sequential Search Algorithm
y = x + 1.841E-11
R² = 1
Time Complexity
Number of Elements N
0
25000
50000
75000
100000
y = x
R² = 1

Exercise 2

There is a third array searching algorithm called the 'Interpolation Search'. It is very similar to the binary search, but uses a different mechanism (actually a statistical equation) to find the new midpoint of the an array.

Implement an interpolation search programme and discuss its merits and demerits. Also, debate on its computational complexity.

You are allowed to take help from online resources. However, references of such resources need to be cited.

Interpolation Search - Test Programme

The following programme (CodeWhoop 2017) demonstrates the benefits of using an interpolation search over a binary search to find a specific value in an array of sorted, uniformly distributed integers.

```
#include <iostream>
using namespace std;

//FUNCTION PROTOTYPES
void printArray(int* myArray, int nElems);
int binarySearch(int* myArray, int arraySize, int searchKey);
int interpolationSearch(int* myArray, int arraySize, int searchKey);


//MAIN
int main()
{
    cout << "This programme demonstrates
                interpolation search." << endl;
    int intArray[] = {10, 12, 13, 16, 18, 19, 20,
                      21, 22, 23, 24, 33, 35, 42, 47};
    int nElems = sizeof(intArray)/sizeof(intArray[0]);
    int value = 18;     //value to search for


    //ECHOING ARRAY
    cout << "The array being searched is as follows" << endl;
    printArray(intArray, nElems);


    //BINARY SEARCH
    cout << "\n\nPerforming binary search for "
           << value << "." << endl;
    int indexOfValue = binarySearch(intArray, nElems, value);
    if (indexOfValue != nElems)
        cout << value << " found at index " << indexOfValue << endl;
    else
        cout << value << " not found in array." << endl;
```

```
    //INTERPOLATION SEARCH
    cout << "\n\nPerforming interpolation search for " << value <<
endl;
    indexOfValue = interpolationSearch(intArray, nElems, value);
    if (indexOfValue != -1)
        cout << value << " found at index " << indexOfValue << "." <<
endl;
    else
        cout << value << " not found in array." << endl;

    cout << "\nEND OF PROGRAMME" << endl;
    return 0;
}//end main


//FUNCTION DEFINITIONS
void printArray(int* myArray, int nElems)
{
    for (int i = 0; i < nElems; i++)
    {
        cout << myArray[i] << " ";
        if ((i + 1)% 8 == 0)
            cout << endl;
    }
    cout << endl;
}//end printArray


int binarySearch(int* myArray, int arraySize, int searchKey)
{
    int lowerBound = 0;
    int upperBound = arraySize - 1;
    int currentIndex;
    int currentIteration = 0;

    while (true)
    {
        cout << "Binary search iteration: " << ++currentIteration <<
endl;

        currentIndex = (lowerBound + upperBound)/2;
        if (myArray[currentIndex] == searchKey)
            return currentIndex;
        else if (lowerBound > upperBound)
            return arraySize;
        else
        {
            if (myArray[currentIndex] < searchKey)
                lowerBound = currentIndex + 1;
            else
                upperBound = currentIndex - 1;
        }//end bound modification else block
    }//end while loop
}//end binarySearch()
```

```
int interpolationSearch(int* myArray, int arraySize, int searchKey)
{
    int start = 0;              //first index of array being searched
    int end = arraySize - 1;    //last index of array being searched
    int currentIndex;           //index being examined by loop
    int searchIteration = 0;    //track how many iterations it takes to
find element

    while (start < end && searchKey >= myArray[start]
           && searchKey <= myArray[end])
    {
        cout << "Interpolation search iteration: "
                << ++searchIteration << endl;
        currentIndex =
        start + ((double)(end - start)/(myArray[end] - myArray[start]))
           * (searchKey - myArray[start]);

        //if element found at currentIndex, return this index
        if (myArray[currentIndex] == searchKey)
            return currentIndex;

        //if element not found, update subarray indexes
        if (myArray[currentIndex] < searchKey)
            start = currentIndex + 1;
        else
            end = currentIndex - 1;
    }//end while loop
    return -1; //exits while loop if element not found -> return -1
}//end interpolationSearch()
```

Interpolation Search - Output

```
This programme demonstrates interpolation search.
The array being searched is as follows
10 12 13 16 18 19 20 21
22 23 24 33 35 42 47


Performing binary search for 18.
Binary search iteration: 1
Binary search iteration: 2
Binary search iteration: 3
Binary search iteration: 4
18 found at index 4


Performing interpolation search for 18
Interpolation search iteration: 1
Interpolation search iteration: 2
18 found at index 4.

END OF PROGRAMME
Program ended with exit code: 0
```

Interpolation Search vs Binary Search

An improved version of the binary search algorithm, the interpolation search algorithm can be used to find a specific element in an ordered array (or similar ordered data structure) of uniformly distributed elements.

Its fundamental principle is the same as binary search algorithm's: to find a specific value by repeatedly dividing a data structure into smaller and smaller substructures so as to iteratively reduce the range of elements which need to be searched. (www.tutorialspoint.com, nd)

The efficiencies of both algorithms depend on choosing which element of the structure to examine. Binary search will always examine the midpoint of the array or data structure and vary the upper and lower bound indexes repeatedly. This success of this approach relies on the `searchKey` being present at the middle index of the subarray or part of the data structure being examined. If this is not the case, successive iterations are necessary to divide the data structure into smaller and smaller parts.

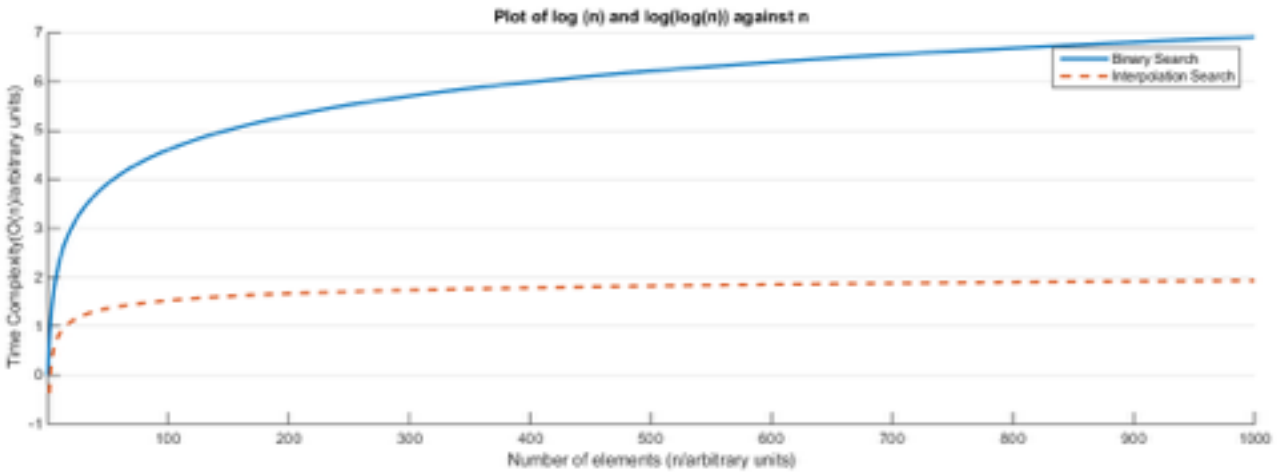
This is the part where the interpolation search algorithm is an improvement over the binary search algorithm. Instead of always using the midpoint of a data structure as its point of examination, the algorithm uses linear interpolation to calculate the search space which may contain the search key. In the `interpolationSearch()` function described above, the current index is updated via linear interpolation. This involves examining the endpoints of the array and determining the amount by which the value stored in an array increases (or decreases) with each index - this ratio of value to index is the gradient (JoshTheEngineer, 2015).

The algorithm uses this gradient to move the currentIndex variable to point to the index which it expects the searchKey to lie on.

This is an improvement over binary search because it uses a statistically proven method to accurately predict the index of the search key, instead of simply dividing the data structure and hoping the search key will lie at the midpoint.

Computational Complexity

On average, interpolation search for a data set with n elements has a time complexity of log (log (n)), which is a significant improvement over the average binary search time complexity of log (n).



Worst case time complexity of the interpolation search algorithm is O(n).

Merits and Demerits

Merits
- Better average time complexity than binary search.
- Same space complexity.

Demerits
- Requires data set to be sorted.
- Does not work well with non-uniform distributions i.e. the probability of choosing an element from the data structure must be the same for all elements. (Khan Academy, 2015)
- Does not work when data set is not linear e.g. when elements show an exponential increase or decrease.

References

Sachdev, A. (n.d.). *Interpolation Search - GeeksforGeeks.* [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/interpolation-search/ [Accessed 31 Jan. 2018].

www.tutorialspoint.com. (n.d.). *Data Structures and Algorithms Interpolation Search*. [online] Available at: https://www.tutorialspoint.com/data_structures_algorithms/interpolation_search_algorithm.htm [Accessed 31 Jan. 2018].

Khan Academy. (2012). *Discrete uniform distribution*. [online] Available at: https://www.youtube.com/watch?v=cyIEhL92wiw [Accessed 31 Jan. 2018].

JoshTheEngineer. (2015). *Explained: Linear Interpolation* [Math]. [online] Available at: https://www.youtube.com/watch?v=Cvc-XalN_kk [Accessed 31 Jan. 2018].

CodeWhoop. (2017). *Interpolation Search Algorithm using C++*. [online] Available at: https://www.youtube.com/watch?v=-MPTAD4z0gY [Accessed 31 Jan. 2018].