**Week 6 - Assignment: Evaluate Searching Algorithms**

Patrick Swafford

TIM-6110: Programming Languages and Algorithms

Jeevan D’Souza

05 Feb 2023

Table of Contents

Program Structure ........................................................................................................................... 3

Integer LinearSearch ....................................................................................................................... 6

Integer BinarySearch ...................................................................................................................... 7

String LinearSearch ........................................................................................................................ 8

String BinarySearch ........................................................................................................................ 9

Program Structure

This program is structured such that an input file, in this case a.in, a command argument, and a search token is passed into the program via the command line. The program reads the lines of the input file into a string array, then compares the instruction against strings named after the four searching algorithms being discussed: linear and binary search of both an integer and a string array. The entire program is wrapped in a try catch block to ensure proper command formatting.

public static void Main(String [] args) {

try {

// variables in readable format

String file = args[0];

String instruction = args[1];

String token = args[2];

// Read in file contents

String [] fin = File.ReadAllLines(file);

// Demonstrate Linear Search with an integer array

if (String.Compare(instruction, "linear\_int") == 0) {

// Convert string array into integer array

int [] arr = Array.ConvertAll(fin, int.Parse);

// convert token to an int

int key = int.Parse(token);

// Write to output the command

Console.WriteLine("Linear Search with an Integer array");

// Write a message telling the user the program is searching

// for the key

Console.WriteLine("Now searching for {0}", key);

// Obtain the index or -1 if key not found

int index = linearSearch(arr, key);

// Output results

if (index < 0) {

Console.WriteLine("The value {0} was not found", key);

return;

}

Console.WriteLine("The Value {0} was found at index: {1}",

key, index);

// Demonstrate Binary Search with an integer array

} else if (String.Compare(instruction, "binary\_int") == 0) {

// Convert string array into integer array

int [] arr = Array.ConvertAll(fin, int.Parse);

// Convert token to an int

int key = int.Parse(token);

// Binary search requires the array to be sorted

Array.Sort(arr);

// Write to output the command

Console.WriteLine("Binary Search with an Integer array");

// Write a message telling the user the program is searching

// for the key

Console.WriteLine("Now searching for {0}", key);

// Obtain the index or -1 if key not found

int index = binarySearch(arr, key);

// Output results

if (index < 0) {

Console.WriteLine("The value {0} was not found", key);

return;

}

Console.WriteLine("The Value {0} was found at index: {1}",

key, index);

// Demonstrate linear search with a string array

} else if (String.Compare(instruction, "linear\_string") == 0) {

// Write to output the command

Console.WriteLine("Linear Search with a String array");

// Write a message telling the user the program is

// searching for the key

Console.WriteLine("Now searching for {0}", token);

// Obtain the index or -1 if key not found

int index = linearSearch(fin, token);

// Output results

if (index < 0) {

                        Console.WriteLine("The value {0} was not found", token);

                        return;

}

Console.WriteLine("The Value {0} was found at index: {1}",

token, index);

// Demonstrate binary search with a string array

} else if (String.Compare(instruction, "binary\_string") == 0) {

// Sort the array lexicographically for a string search

Array.Sort(fin);

// Write to output the command

Console.WriteLine("Binary Search of a String array");

// Write a message telling the user the program is searching

// for the key

Console.WriteLine("Now searching for {0}", token);

// Obtain the index or -1 if key not found

int index = binarySearch(fin, token);

// Output results

if (index < 0) {

Console.WriteLine("The value {0} was not found", token);

return;

}

Console.WriteLine("The Value {0} was found at index: {1}",

token, index);

// Throw an error if invalid input

} else {

throw new FormatException();

}

// Exceptions: Self explanatory

} catch (IndexOutOfRangeException) {

Console.Error.WriteLine("Error: Index Out Of Range");

} catch (FileNotFoundException) {

Console.Error.WriteLine("Error: File not found");

} catch (FormatException) {

Console.Error.WriteLine("Error: Format Exception");

}

return;

Integer LinearSearch

The Big O notation for this linear search algorithm is O(n). This is because the algorithm has one loop that iterates n times in the worst case, where n is the length of the array. The loop iterates through the length of the array, and either returns the index of the key if the key is an element of the array, or -1 otherwise.

public static int linearSearch(int [] arr, int key) {

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

if (arr[i] == key) {

return i;

}

}

return -1;

}

Text

Description automatically generated

Integer BinarySearch

The Big O notation for this BinarySearch algorithm is O(log n). This is because even in the worst case scenario, where the key is not in the array, the algorithm will only need log n iterations.

public static int binarySearch(int [] arr, int key) {

return binarySearch(arr, key, 0, arr.Length - 1);

}

private static int binarySearch(int [] arr, int key, int low, int high) {

if (high >= low) {

                int med = (low + high) / 2;

if (key == arr[med])

return med;

if (key < arr[med])

return binarySearch(arr, key, low, med - 1);

if (key > arr[med])

return binarySearch(arr, key, low + 1, high);

}

return -1;

        }

Text

Description automatically generated

String LinearSearch

The Big O notation for this String LinearSearch algorithm, like its integer counterpart, is also O(n), for the same reasons. This algorithm, in the worst case, will search the entire contents of the array before it returns -1 indicating that the key was not found. A key difference is that unlike with the integer counterpart, a non-numeric input key does not throw an error.

Text

Description automatically generated

Text

Description automatically generated

String BinarySearch

The Big O notation for this String BinarySearch algorithm, like its integer counterpart, is also O(log n), for the same reasons. This algorithm, in the worst case, will only make log n iterations before it returns -1 indicating the key was not found in the array. A key difference is that unlike with the integer counterpart, a non-numeric input key does not throw an error. Also note that the returned index of the key “21” is different. This is not a bug. The array had to be sorted lexicographically in order for the string variant of binary search to work properly.

public static int binarySearch(String [] arr, String key) {

return binarySearch(arr, key, 0, arr.Length - 1);

}

private static int binarySearch(String [] arr, String key, int low, int high) {

if (high >= low) {

int med = (low + high) / 2;

                //Console.WriteLine("{0} {1}", arr[med], key);

if (String.Compare(arr[med], key) == 0)

return med;

if (String.Compare(arr[med], key) > 0)

return binarySearch(arr, key, low, med - 1);

if (String.Compare(arr[med], key) < 0)

return binarySearch(arr, key, low + 1, high);

}

return -1;

}

Text

Description automatically generated