Lab Assignment #1

Linear/Binary Search

Full Name:
BlazerId:

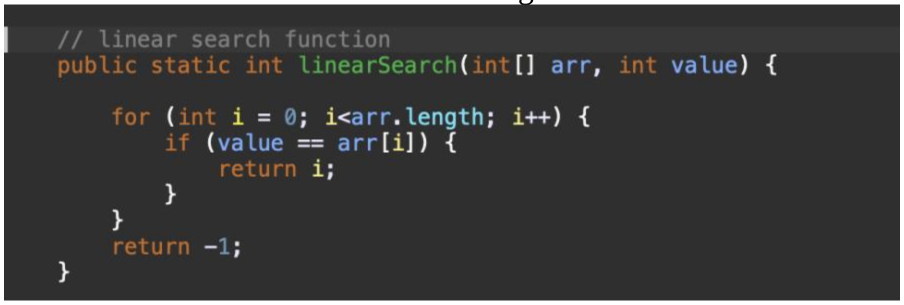
1. **Problem Specification**

The purpose of this project is to gain an understanding of two different searching algorithms, linear search and binary search. In order to better understand these methods, we implement the two algorithms and then use the algorithms to search for 1000 integers in sets of varying (increasing) size. We then evaluate / compare the runtimes for linear and binary search at varying dataset sizes.

2. **Program Design**

The code was written in a multitude of steps.

- Step 1 was to first implement the algorithms for linear search and binary search. The below screenshot is the search algorithm for linear search.

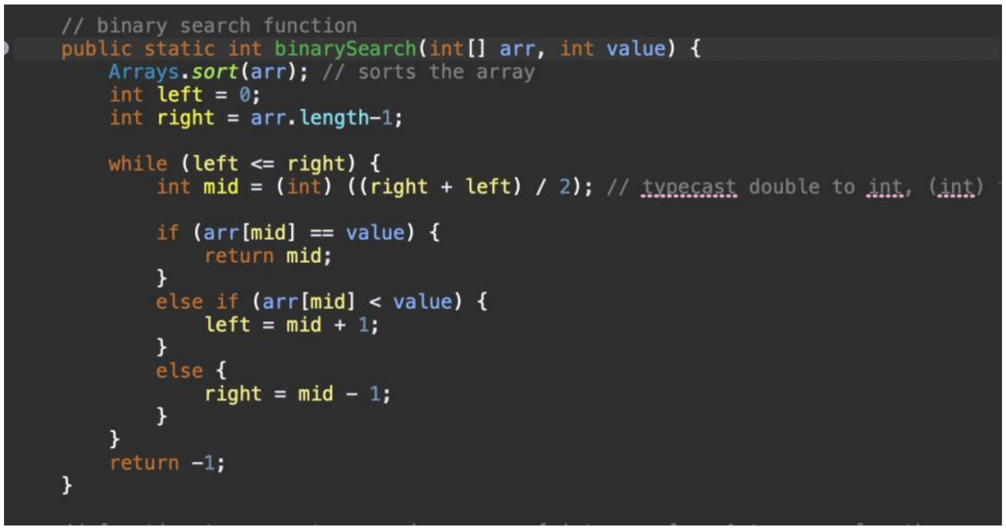


```
// linear search function
public static int linearSearch(int[] arr, int value) {

    for (int i = 0; i<arr.length; i++) {
        if (value == arr[i]) {
            return i;
        }
    }
    return -1;
}
```

- The code works by taking an integer value (value), and comparing it against the value of each value in an array of integers one by one. This is done by using a for loop beginning with index i = 0 all the way to i < array.length and comparing the two values (compares the target value against the values of the array from start to finish). If the values are equal, then the index at which they are equal is returned, if the values are not equal, the next iteration of the for loop occurs. -1 is returned if no equivalent value is found by the end of the for loop.

- Below is the screenshot of the binary search algorithm.



```
// binary search function
public static int binarySearch(int[] arr, int value) {
    Arrays.sort(arr); // sorts the array
    int left = 0;
    int right = arr.length-1;

    while (left <= right) {
        int mid = (int) ((right + left) / 2); // typecast double to int, (int)

        if (arr[mid] == value) {
            return mid;
        }
        else if (arr[mid] < value) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}
```

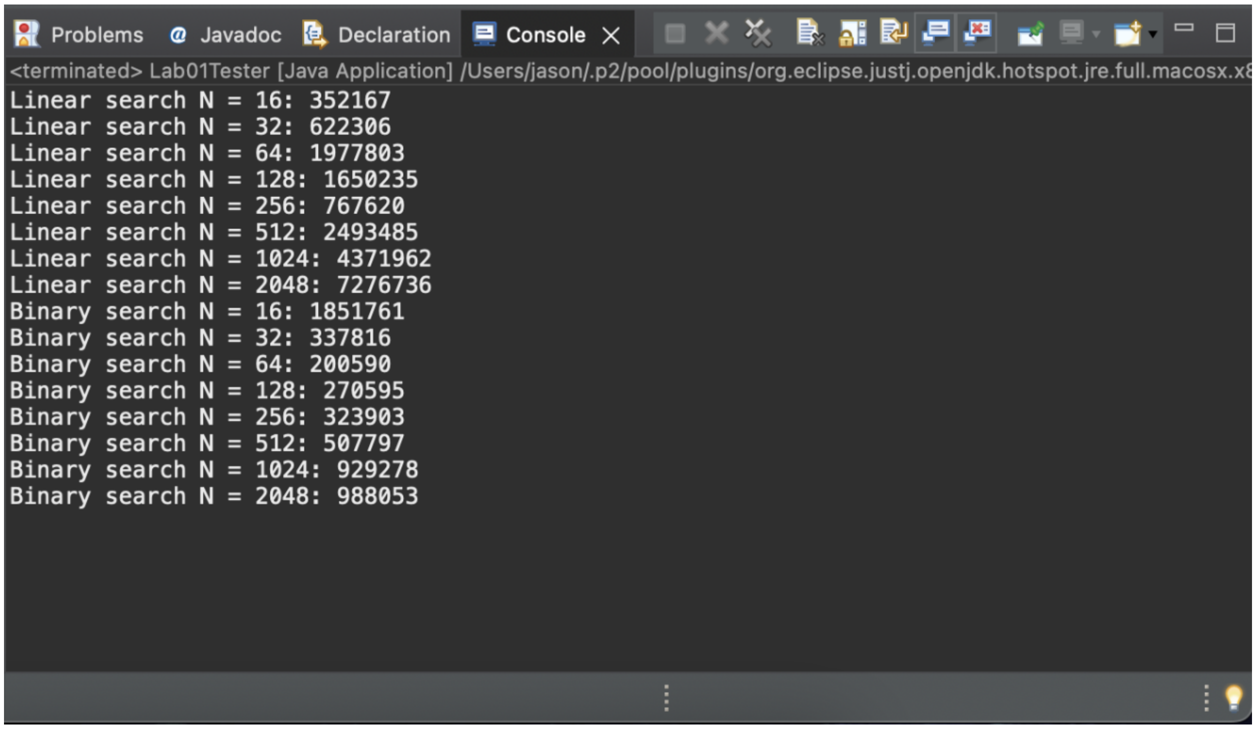
- For binary search, the algorithm works by first using the built-in java Arrays method to sort the array in ascending order. Then, a while loop is used to compare the target search key with the middle value of the array. While the left index is less than or equal to the right index (start and end of array), the loop continues. This is done so that the final iteration of the loop is on an array of size 1. If the target value is equal to the value of the midpoint of the array (rounded down, i.e. array of size 14 considers the 7th value as the middle number), the index of the midpoint is returned and the loop terminates. If the midpoint value is < the target value, it means the target value must be to the right of the midpoint, and so we change the value of the left index to be midpoint index + 1 so that we only search values to the right. If the midpoint value is > the target value, the target value must be to the left of the midpoint, and so we change the right index to be midpoint index – 1 so we only search values to the left of that midpoint. This continues until the midpoint value is either equal to target value or we return -1 because the target value is not in the array.

- Another function is used to generate a random array of size N with values ranging from 1 to N inclusive. This is done using Java's random library and the nextInt function which generates a random int from 1 to n (excludes upper bound so +1 is necessary in code). Loops through each index of the array and sets value at index = to randomly generated int. Function name: randomArray
- The main function declares arrays of different sizes and populates them using the randomArray function described above.

- A readfile function is used which takes a string of the filepath as input and returns a comma separated string of integers from the file which will be used as search keys.
- Using a try catch block, readfile is called and the string is saved as a message. This message is then turned into a string array (each value separated by a comma) and a for loop is then used to convert each string value to an int and store it into a new integer array.
- A for each loop is then used on the integer array of values from the text file, with the values being the keys (values we search for in the randomly generated arrays) used in the linear and binary search function calls. StartTime and endTime variables are then used to store nanoTime in order to calculate the time it takes to search (or sort + search) each of the keys in the randomly generated arrays. EndTime – startTime is used to calculate search time in nanoseconds, which is then printed out.

3. Output

Screenshot of runtime for linear and binary search for datasets of size 16, 32, 64, 128, 256, 512, 1024, and 2048 (2^{11})



```
<terminated> Lab01Tester [Java Application] /Users/jason/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x8
Linear search N = 16: 352167
Linear search N = 32: 622306
Linear search N = 64: 1977803
Linear search N = 128: 1650235
Linear search N = 256: 767620
Linear search N = 512: 2493485
Linear search N = 1024: 4371962
Linear search N = 2048: 7276736
Binary search N = 16: 1851761
Binary search N = 32: 337816
Binary search N = 64: 200590
Binary search N = 128: 270595
Binary search N = 256: 323903
Binary search N = 512: 507797
Binary search N = 1024: 929278
Binary search N = 2048: 988053
```

Table of data for linear and binary search runtimes

Size of dataset	Linear Search Time (nanoseconds)	Binary Search Time (nanoseconds)
16	352,167	1,851,761
32	622,306	337,816
64	1,977,803	200,590
128	1,650,235	270,595
256	767,620	323,903
512	2,493,485	507,797
1024	4,371,962	929,278
2048	7,276,735	988,053

4. Analysis and Conclusions

Time complexity of linear search is O(n) because it loops through the whole list 1 by 1. Time complexity of binary search is O(logn) because we compare target value once with the midpoint per iteration, and each iteration cuts the size of the list by half. Based on the time complexity of linear and binary search, we would expect binary search to be faster (on average) than linear search (though both searches may be extremely quick if the target value is towards the start of the array). The concluding data shows that binary search takes a lesser amount of time on average than linear search, even when sort time is taken to account, with the difference in speed becoming more apparent at larger datasets. Additionally, the results show that runtime for linear search on average increases at a significantly greater rate than for binary search at larger datasets.

These results accurately represent the O(n) and O(logn) time complexities of linear and binary search respectively, where O(n) should increase at a much faster rate than O(logn).

5. References

1. https://stackoverflow.com/questions/18838781/converting-string-array-to-an-integer-array
   a. Used for converting string to array of integers
2. https://www.youtube.com/watch?v=fDKIpRe8GW4&ab_channel=MichaelSambol
   a. Used for binary search algorithm and information
3. https://www.youtube.com/watch?v=246V51AWwZM&ab_channel=BroCode
   a. Linear search algorithm and information
4. https://stackoverflow.com/questions/3382954/measure-execution-time-for-a-java-method
   a. Used for syntax for getting runtime in java