Objective

This lab is to study how different sizes of input can impact execution time with different searching algorithms. Two specific methods of searching are examined: Linear Search and Binary Search.

Methodologies

A *StringData* class is provided to generate a pre-sorted array of strings as search base. Three key words – "not_here", "mzzzz", "aaaaa" – are to be searched in the search base, with Linear Search method and Binary Search method respectively. During the code run under the driver program *Analyzer*, execution time is recorded from the start to the end of code execution to calculate an execution time for finding the different keywords in search base with each method. Codes in *Analyzer.java* is attached in Appendix with the implementation of Binary and Linear search methods.

Results

Table 1: Execution time for searching different keywords with different searching methods.

Keywords	Index	Execution time in nano seconds (Binary)	Execution time in nano seconds (Linear)
"not_here"	-1	44088	63559516
"mzzzz"	5940687	1844	96124501
"aaaaa"	0	18112	87760908

Table 1 shows the index of each keyword in the search base as well as the execution time in nano seconds of the searching with Binary and Linear search method respectively.

Responses to questions

1. Why is a search for "not_here" the worst-case for linear search and binary search?

   In the pre-sorted search base, the keyword doesn't exist. For linear search, it has to go through all the elements in the search base and compare with the keyword, and the search base is fairly large. For binary search, even each execution of the run eliminates half of the search base, the algorithm still has to search through each half of the search base until it runs out, and again the search base is fairly large.

2. Why is a search for "mzzzz" the average-case for linear search?

   In the pre-sorted search base, the keyword exists at the index which is not very extreme. It certainly doesn't exist at the beginning which can be searched fairly quick. The keyword doesn't exist at the very last which the algorithms have to go through lots of elements to find it. It sits around the middle where linear search runs about halfway.

3. Why is a search for “aaaaa” the best-case for linear search?

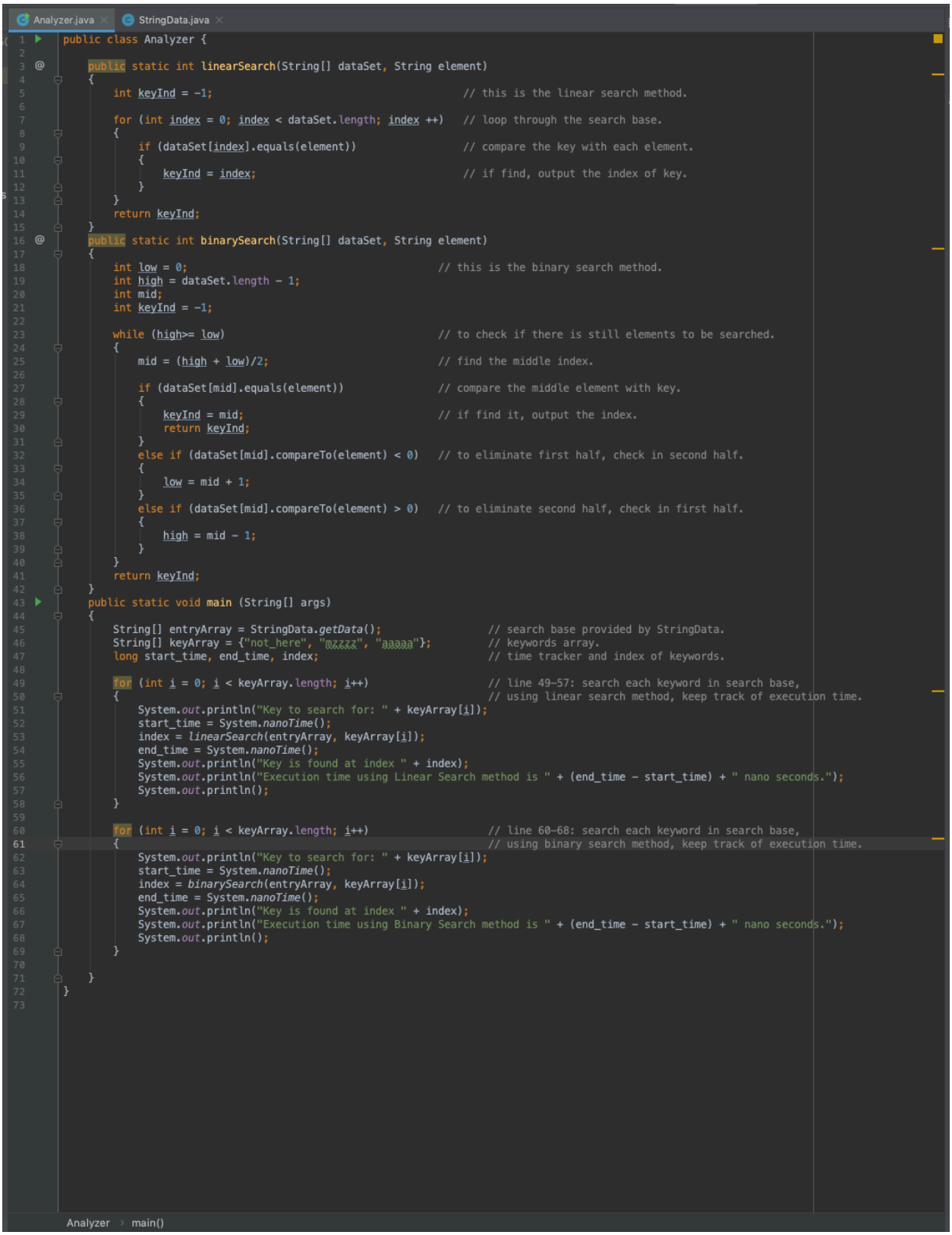
   In the pre0sorted search base, the keyword is right at the start where the linear search begins from the start. Once the algorithm finds it, it would stop and require no more runs. Unlike linear search, binary search will begin at the middle and even though the keyword is right at the start, binary algorithm still has to search the middle position every step of the run until it finds, which a lot of unnecessary work is done.

4. How do the results you saw compare to the Big-O complexity for these algorithms?

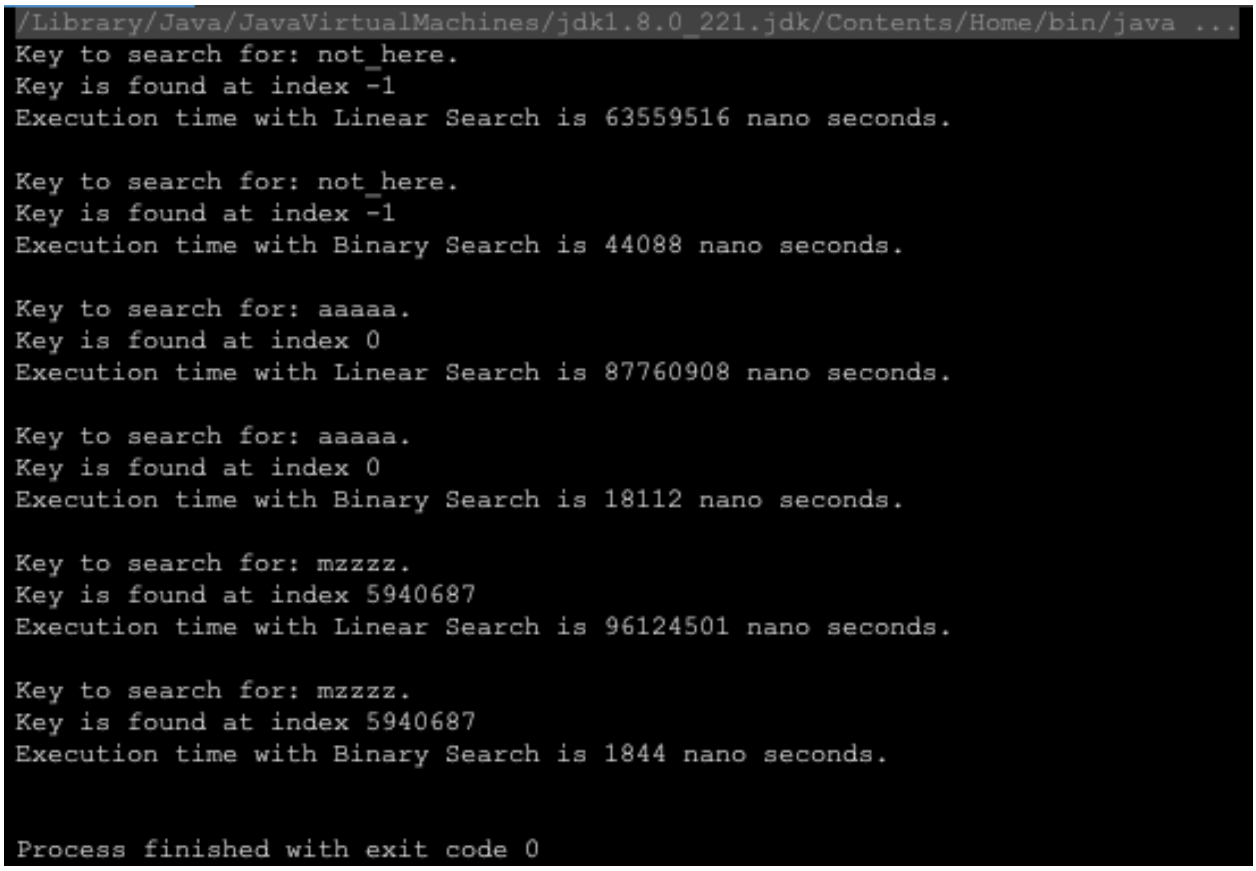
   For linear search algorithm, the results match the Big O complexity of the algorithm which is O(N), N is the number of elements or iterations the algorithm needs to run. The size of search base and execution time has correlations in the order of magnitude of size and execution time. For binary search algorithm, it exhibits an O(log N) complexity. The keyword “mzzzz” is found at the first iteration. The keyword “not_here” is never found after going through the entire search base which has the longest execution time. Compare to the execution time for finding keyword “aaaaa”, they share correlation with logarithmic functions which the execution time of finding keyword “aaaaa” remains the same order of magnitude and not drastically diverge.

5. Why do the binary search results appear so similar, while the linear search results are so divergent?
   Because binary search exhibits O(log N) complexity, as the size N increases, the result function should grow, but converge which the increased size should not cause drastic differences. Linear search, on the other hand, exhibits linear complexity O(N) where the result function would not converge, instead, it will increase as the size N increases linearly, to infinity theoretically speaking. So at the end it is really the growth rate function dominates the behavior of the function mathematically. If the basic growth rate function diverges, the result function only will increase as the size N increases. If the basic growth rate function converges, the result function will increase but eventually plateau.

Appendix



```java
public class Analyzer {

    public static int linearSearch(String[] dataSet, String element)
    {
        int keyInd = -1;                                    // this is the linear search method.

        for (int index = 0; index < dataSet.length; index ++)   // loop through the search base.
        {
            if (dataSet[index].equals(element))             // compare the key with each element.
            {
                keyInd = index;                             // if find, output the index of key.
            }
        }
        return keyInd;
    }
    public static int binarySearch(String[] dataSet, String element)
    {
        int low = 0;                                        // this is the binary search method.
        int high = dataSet.length - 1;
        int mid;
        int keyInd = -1;

        while (high>= low)                                  // to check if there is still elements to be searched.
        {
            mid = (high + low)/2;                           // find the middle index.

            if (dataSet[mid].equals(element))               // compare the middle element with key.
            {
                keyInd = mid;                               // if find it, output the index.
                return keyInd;
            }
            else if (dataSet[mid].compareTo(element) < 0)   // to eliminate first half, check in second half.
            {
                low = mid + 1;
            }
            else if (dataSet[mid].compareTo(element) > 0)   // to eliminate second half, check in first half.
            {
                high = mid - 1;
            }
        }
        return keyInd;
    }
    public static void main (String[] args)
    {
        String[] entryArray = StringData.getData();                 // search base provided by StringData.
        String[] keyArray = {"not_here", "mzzzz", "aaaaa"};         // keywords array.
        long start_time, end_time, index;                           // time tracker and index of keywords.

        for (int i = 0; i < keyArray.length; i++)                   // line 49-57: search each keyword in search base,
        {                                                           // using linear search method, keep track of execution time.
            System.out.println("Key to search for: " + keyArray[i]);
            start_time = System.nanoTime();
            index = linearSearch(entryArray, keyArray[i]);
            end_time = System.nanoTime();
            System.out.println("Key is found at index " + index);
            System.out.println("Execution time using Linear Search method is " + (end_time - start_time) + " nano seconds.");
            System.out.println();
        }

        for (int i = 0; i < keyArray.length; i++)                   // line 60-68: search each keyword in search base,
        {                                                           // using binary search method, keep track of execution time.
            System.out.println("Key to search for: " + keyArray[i]);
            start_time = System.nanoTime();
            index = binarySearch(entryArray, keyArray[i]);
            end_time = System.nanoTime();
            System.out.println("Key is found at index " + index);
            System.out.println("Execution time using Binary Search method is " + (end_time - start_time) + " nano seconds.");
            System.out.println();
        }

    }
}
```



Reference

"Introduction to Java programming", Y.Daniel Liang, 10th Edition. Ch 22