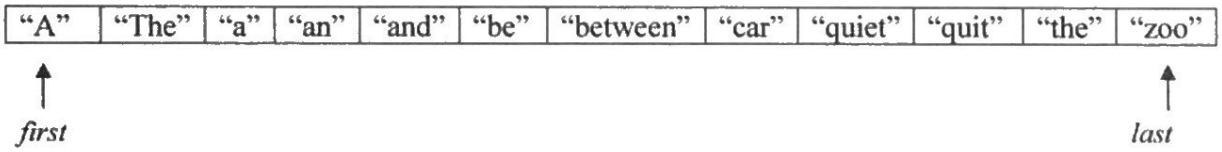
Tanmai Kalidindi

Search: Linear and Binary

To experience different methods of searching you will read in the words from the Declaration of Independence from a file, sort them, prompt the user to enter a word, and then perform both a linear search and a binary search to look for the word. Each of these methods returns either the location of the word or -1 if the word was not present. Each method also counts and stores the number of comparisons made during its search. Predict the search that will need fewer comparisons.

A **linear search** is usually implemented iteratively, meaning that it looks at each word in turn, starting from the beginning of the list.

A **binary search** can be implemented either iteratively or recursively, but we will do it recursively. Each recursive call calculates the middle index and compares that value to the target. If they are equal, return the index and you're done. If the value is less than the target, then search the upper half. Else, search the lower half. Recur until either a) the index finds the target, or b) *first* is greater than *last*. Try it out below. Search the array below for "quiet". Then search the array for "if".



Assignment

Open up `Searches_Driver.java`. Notice that each method *returns a value*, but does not print it, which is standard practice on the APCS A exam. In the driver, after reading 1325 words from `declaration.txt` into an array, we sort the array by calling `Arrays.sort`. Notice how the driver uses the return values from `Searches.linear` and `Searches.binary` to display the relevant messages.

The method headers in this lab are:

```
public static int linear(Comparable[] array, Comparable target)
public static int binary(Comparable[] array, Comparable target)
private static int helper(Comparable[] array, Comparable target, int first,
                                                                 int last)
```

Sample Run

```
Enter a word:    Liberty
                 Linear Search found it at location 133 in 134 comparisons.
                 Binary Search found it at location 133 in 10 comparisons.

Enter a word:    oppression
                 oppression was not found by the linear search.
                 oppression was not found by the binary search.

Enter a word:    He
                 Linear Search found it at location 86 in 87 comparisons.
                 Binary Search found it at location 101 in 6 comparisons.
```

Big-O and Linear Search

5	2	8	10	2	4	0	2	1	8	5	3	5	9

1. Given an array, what is the Big-O of the best case in the Linear Search? O(1)
2. What is the Big-O of the average case in the Linear Search? $O(\frac{n+1}{2})$
3. What is the Big-O of the worst case in the Linear Search? O(n)

Big-O and Binary Search

4	6	8	10	12	14	20	22	31	38	45

4. Given a sorted array, what is the Big-O of the best case in the Binary Search? O(1)
5. What is the Big-O of the average case in the Binary Search? O(log(n))
6. What is the Big-O of the worst case in the Binary Search? O(log(n))

The Big-O efficiency of the binary search (in the average case) is **O(log n)**. If *n* were doubled, then the work to search an item is, on average, increased by 1. Thinking about it a different way, the binary search discards half the data at each check. (There is a precondition: the data set must be sorted—assume it is).

7. In the Binary Search, the depth of recursion *k* is related to *n* elements by the equation: $n = 2^k$. Solve for *k*. *k* =
8. $\log_2(8) =$ 3 $\log_2(32) =$ 5 $\log_2(100) \approx$ 6.5 $\log_2(260) \approx$ 8
9. In general, given n elements, how many comparisons, on average, does the **binary search** require? $\log_2(n)$
10. Calculate the average case Big-O values for binary searches of

100 elements	6.644
3000 elements	11.551
1,000,000 elements	19.932

11. In practical, not theoretical, terms, which is faster for small sets, linear search or binary search?

linear

12. In practical, not theoretical, terms, which is faster for large sets, linear search or binary search?

binary