Binary Search

An Article for Assignment 6 on LaTeX
CS251: Computer Laboratory

Jayant Agrawal
(14282)

INSTRUCTOR
Prof. Arnab Bhattacharya

Department of Comuter Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
KANPUR 208016, INDIA

Feb 2016

Contents

List of Figures

List of Tables

1 Abstract

Searching and Sorting are the two most basic routines that are involved in almost every Software . As the name suggests, searching involves finding a key in a given list. The trivial solution is to iterate over all the elements of the list and report whether the key is found or not. The Time Complexity involved in this operation is of the order of the size of the list. In this article, we see how to solve this problem in a much more efficient way. The solution is known as **Binary Search** and the time complexity involved is of the order of $log(n)$, if the given input is arranged nicely, where 'n' is the size of the given list or set.

2 Algorithm

Though, this approach has a wide variety of applications, it is most commonly used to find a given value within a sorted sequence. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.[CB]

The main idea of binary search is to keep track of reasonable guesses. Lets say that A and B are playing a game known as the *guessing game*, where A selects a number between 1 and 100. Now, B's job is to find out the number that A selected. Lets say B comes up with two guesses, 25 and 81. A says that his number is higher than the first guess and lower than the second guess.

Figure 1: Red Area : Reasonable guesses

Then, it is obvious that the selected number lies between 26 and 80. Figure 1 shows the remaining area of reasonable guesses. We can safely ignore the black region. In the next turn, it is reasonable to guess a number such that it divides the set of reasonable guesses into two ranges of the same size. If the guess is not correct, then based on A's response whether the guess is too high or low, we can again discard half the remaining choices. For example, in this case the halfway point would be $(26 + 80)/2$, or 53. If A says that it is too high, then we can eliminate 53 to 80 as shown in Figure 3.

Figure 2: Remaining set of guesses

In this way we can keep halving the size or our domain and arrive at the correct guess. We can formulate our algorithm as follows. Let the variable $left$ be the correct minimum reasonable guess, and $right$ be the correct maximum reasonable guess. The input to the problem is the number n, the highest possible number that the opponent can think of. We assume that the lowest possible number that can be chosen is 1. The algorithm is as follows. [CB]

- Let $left = 1$ and $right = n$.
- Guess the average of $right$ and $left$, rounded down (so that it is an integer).
- If you guesses the number, *stop* . You found it!.
- If the guess was too low, set $left$ to be one larger than the guess.
- If the guess was too high, set $right$ to be one smaller than the guess.
- Go back to step 2.

Following is the pseudo code for the above algorithm.

2.1 Pseudo-Code

Consider the simple problem of finding a key within an array. The following pseudo-code returns the index of the key, if it is found. Let the size of the array 'A' be 'n'. [Wik]

Algorithm 1 Binary Search Algorithm

```
1: procedure BinSearch
2:     left ← 0
3:     right ← n − 1
4:     while right ≤ left do
5:         mid ← (left + right)/2
6:         if A[mid] == key then
7:             return mid
8:         else if A[mid] < key then                ▷ Discarding left half
9:             left ← mid + 1
10:        else                                     ▷ Discarding right half
11:            right ← mid − 1
12:        end if
13:    end while
14:    return NOT_FOUND
15: end procedure
```

2.2 Complexity

Consider Algorithm 1, it is easily observable that each iteration the set of remaining guesses reduces exactly by half untill there is only one value left. At each iteration the time taken is of the order of constant time. So, the time complexity of our algorithm depends only on the number of total iterations. We start with the total size of n and end up at the size of one irrespective of whether we find the key or not, after halving the size of the set at each iteration. Thus, the total no of iterations are of the order of $log_2(n)$.

n	$log_2(n)$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
1048576	20
2097152	21

Table 1: No of Iterations for different values of n

Table 1 shows the no of iterations it takes for different values of n. Therefore, assuming the RAM model of computaion, the time complexty of the algorihtm $T_0(n)$ is given by:

$$T_0(n) = O(log_2(n)) \tag{1}$$

3 Pre-Processing Time

In the above description, we have always assumed that the given input is sorted. If the given is sorted, then pre-processing time is of the order of constant time. Binary Search should always be applied in this case.

Now, if the input is in a random order, binary search algorithm can not be applied without pre-processing. For the algorithm to be applicable it is necessary that the input be sorted. So, the pre-processing time is equivalent to the time taken to sort the input. The fastest sorting algorithm that exists, takes $O(n * log(n))$ time, where n is the size of the given input.

4 Comparison with linear search

Table 1 can also be shown as in Figure 3.

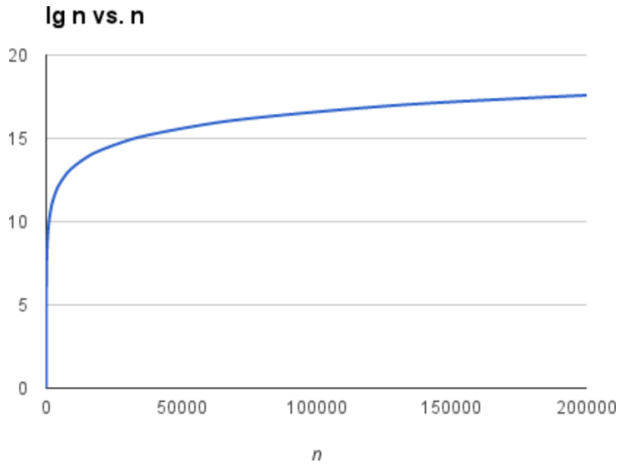

Figure 3: $log(n)$ vs n

Linear Search is a technique where the key is searched by going over the entire set of inputs and checking if the element is the key or not. Obviously, the time taken for this algorithm in the worst case is of the order of the size of the input i.e. n $[O(n)]$.

As mentioned in Section 3, if the given input is sorted, binary search should be applied irrespective of the input size and no of queries. But if the given input is not sorted the deciding between binary search and linear search depends upon the application conditions.

Let the number of querries be k. Then, the total time taken for the Binary Search algorithm $(T_1(n))$ is given by :

$$T_1(n) = O(n * log(n) + k * log(n)) \tag{2}$$

The total time taken for Linear Search Algorithm $(T_2(n))$ is given by :

$$T_2(n) = O(k * n) \tag{3}$$

In the case of a single query, Equations 1 ,2 and 3 can be visualised as in Figure 4.

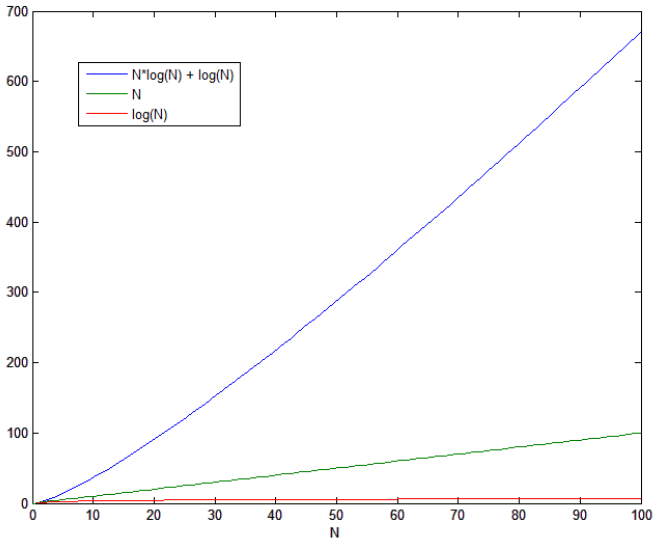

Figure 4: For k = 1, Single Query

4.1 No of Queries- Large

For large k, from Equations 2 and Equations 3, we can see that, we should apply binary search algorithm.

4.2 No of Queries- Low

In this case, the approach to be applied depends on the input size n. For small n, the time taken for both the approaches is similar.

Note: In some applications, we need to insert some elements in the input set and then search for the key. In the case of binary search algorithm, inserting takes extra time whereas in linear search inserting is easy. Similarly for deletion. Thus, the approach to be applied depends on these parameters as well.

4.3 Summary

Table 2 summarizes the comparison between binary and linear search.

Sr. No.	Comparison	Linear Search	Binary Search
1	Time Complexity	$O(n)$	$O(log_2(n))$
2	Best Case time	$O(1)$ [First element]	$O(1)$ [Center element]
3	Prerequisite [Array]	None	Sorted
4	Worst Case[n=1000]	1000 comparisons	10 comparisons
5	Data Structure	Array, Linked List	Array
6	Insert Operation	Easy	Requires processing

Table 2: Comparison between Linear search and Binary Search[Kum15]

5 Conclusion

Binary search is a more efficient searching technique than linear search but insertion of an element is not efficient as it requires arranged elements in specific order. Further it is also possible to apply binary search on linked list by making necessary modifications to original binary search algorithm. The selectiom of searching algorithm can be based on the data structure on to which it is applied and which operations are required more. Balance is required between search and maintanance of a data structure.[Kum15]

References

[CB] Thomas Cormen and David Balkcom. Binary search. [Accessed; 27 Feb 2016].

[Kum15] Parmar Kumbharana. Comparing linear search and binary search algorithms to search an element from a linear list implemented through static array, dynamic array and linked list. *International Journal of Computer Applications*, 121(3):13–17, 2015.

[Wik] Wikipedia. Binary search algorithm. [Accessed; 27 Feb 2016].

Index