Implementation of Ternary Search Algorithm

Akshay Gupta IIT2017505, Snigdha Dobhal IIT2017506, Naman Deept IIT2017507

Dept. of Information Technology
Indian Institute of Information Technology Allahabad

April 7, 2019

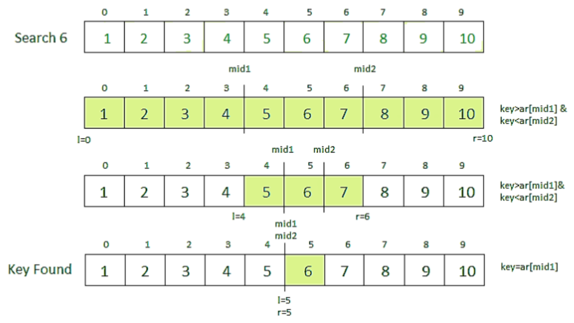
1 Abstract

This paper discusses ternary search algorithm which uses divide and conquer approach to solve/search a given number/element/object out of sorted sequence/enumeration of the number/elements/objects of that kind. The given algorithm in the paper uses recursion as well as iteration as the implementation of the algorithm.

Keywords: *Ternary*, *Search*, *Divide*, *Conquer*, *Recursion*, *Iteration* .

2 Introduction

A ternary search algorithm is a technique in computer science for finding the minimum or maximum of a uni-modal function. A ternary search determines either that the minimum or maximum cannot be in the first third of the domain or that it cannot be in the last third of the domain, then repeats on the remaining two thirds.
Ternary search is a divide and conquer algorithm that can be used to find an element in an array. It is similar to binary search where we divide the array into two parts. In this algorithm, we divide the given array into three parts and determine which has the key (searched element). We can divide the array into three parts by taking mid1 and mid2 which can be calculated as shown below. Initially, l and r will be equal to 0 and n-1 respectively, where n is the length of the array.



3 Proposed Method

Input :

3.1 Procedure

1. First, we compare the key with the element at mid1. If found equal, we return mid1.
2. If not, then we compare the key with the element at mid2. If found equal, we return mid2.
3. If not, then we check whether the key is less than the element at mid1. If yes, then recur to the first part.
4. If not, then we check whether the key is greater than the element at mid2. If yes, then recur to the third part.
5. If not, then we recur to the second (middle) part.

3.2 Algorithm

3.2.1 Recursive Approach:

We divide the array into three parts and use divide and conquer strategy to find our desired key using this method.

```
procedure TERNARY($array, left, right, key$)
    $gap \leftarrow right - left$
    if $gap \geq 0$ then
        $mid_l \leftarrow left + \frac{gap}{3}$
        $mid_r \leftarrow right - \frac{gap}{3}$
        if $key = array[mid_l]$ then
            return $mid_l$
        end if
        if $key = array[mid_r]$ then
            return $mid_r$
        end if
        if $key < array[mid_l]$ then
            $Ternary(array, left, mid_l + 1, key)$
        end if
        if $key > array[mid_r]$ then
            $Ternary(array, mid_r + 1, right, key)$
        end if
        if $arr[mid_l] < key < arr[mid_r]$ then
            $Ternary(array, mid_l + 1, mid_r - 1, key)$
        end if
    end if
    if $gap < 0$ then
        return NOT FOUND
    end if
end procedure
```

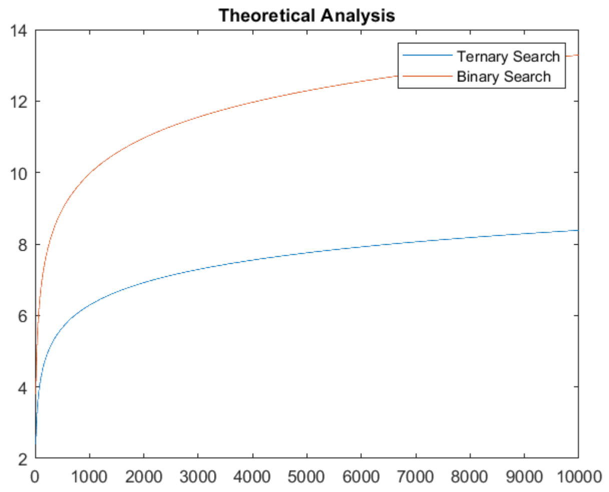
3.2.2 Iterative Approach:

As the problem can also be solved without using recursion ,the iterative approach should also be discussed.

```
procedure $ternarySearch(array, l, r, key)$
    while $r \geq l$ do
        $mid1 \leftarrow l + (r - 1)/3$
        $mid2 \leftarrow r + (r - 1)/3$
        if $array[mid1]$ == key then
            return FOUND
        end if
        if $array[mid2] == key$ then
            return FOUND
        end if
        if $key < array[mid1]$ then
            $r \leftarrow mid1 - 1$
        end if
        if $key > array[mid2]$ then
            $l \leftarrow mid2 + 1$
        end if
        else
            $l \leftarrow mid1 + 1$
            $r \leftarrow mid2 - 1$
    end while
    return NOT FOUND
end procedure
```

4 Time Complexity Analysis

Figure: Theoretical analysis depicting the time comparison of ternary search vs binary search .



In our ternary search algorithm, we partition the list (or the array) into 2 parts with one with n/3 size and other being 2n/3 size where n is the size of the sorted sequence of the elements .

T(n) = T($\frac{2n}{3}$) + $4c$;

Applying Master theorem we get

$\implies T(n) = \frac{4logn}{log3}$+c$_1$

$\implies T(n) \propto(\log_3 n)$

So , from the above recurrence relation , it can be concluded that the time complexity of the ternary search algorithm is proportional to order of ($log_3 n$) . But , in the best case scenario when element to be searched is found at either of the trisection of the division is made during the ternary search implementation , it will find the element in it's first attempt and hence the best case time complexity of the ternary search algorithm is $\Omega(1)$.

While the worst case scenario happens one it iterates over all the elements and takes time proportional to the one earlier solved and hence the worst case time complexity of the ternary search algorithm is $\mathbf{O}(log_3 n)$.

5 Experimental Analysis

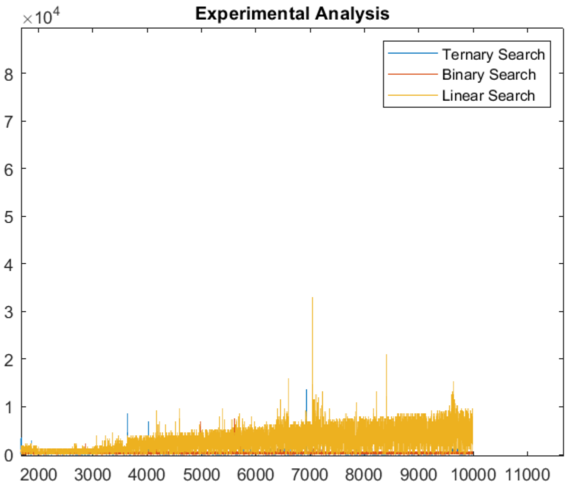


Figure:Experimental Analysis time taken by ternary search , binary search and linear search algorithm vs size of the sorted sequence .

6 Conclusion

Ternary search Algorithm was found to be way faster algorithm to search the given element out of a sorted sequence of that kind of elements in comparison to the binary search algorithm and the linear search algorithm . The difference between ternary search algorithm and binary search algorithm is significant for large size of sequence of elements . Better methods like quaternary search algorithm could be applied to get more efficient result for searching while staying stable and reliable.

7 References

1. Ternary Search *https://www.geeksforgeeks.org/ternary−search/* retrieved on 7th April 2019
2. Ternary Search *https://www.hackerearth.com/practice/algorit search/tutorial/* retrieved on 7th April 2019
3. Ternary Search *https://en.wikipedia.org/wiki/Ternary_search* retrieved on 7th April 2019
4. Introduction to Algorithms by Cormen