A Novel Ternary Search Algorithm

Nitin Arora
Dept. of Computer Science & Engineering
WIT, Dehradun, Uttarakhand, India

Mamta Martolia Arora
Dept. of Computer Science & Engineering
WIT, Dehradun, Uttarakhand, India

Esha Arora
Dept. of Computer Science & Engineering
DIET, Rishikesh, Uttarakhand, India

ABSTRACT
Searching is an algorithm that search a particular element in a given list of elements. Sorting Technique is frequently used in a large variety of important applications to search a particular element. Several Searching Algorithms of different time and space complexity are exist and used. This paper provides a novel searching algorithm Ternary search which is based on dividing the given elements into three parts. We also compare the Ternary search algorithm with Linear Search and Binary Search. MATLAB is used for implementation and Analysis of CPU time taken for all the three searching algorithms used. Linear search can be used with any random array but for binary search and ternary search sorted array is required. Result shows that ternary search algorithm requires less time for search any particular element.

Keywords
Binary Search, Linear Search, Algorithm, Data Structures

1. INTRODUCTION
Searching is an algorithm that search a particular element in a given list of elements. Sorting Technique is frequently used in a large variety of important applications to search a particular element. Several Searching Algorithms of different time and space complexity are exist and used. This paper discussed and compare the previously exist searching algorithms.

This paper provides a novel searching algorithm Ternary search which is based on dividing the given elements into three parts. In this paper the comparison of the Ternary search algorithm with Linear Search and Binary Search is also has been discussed.

The remainder of this paper is organized as follows. Section 2 introduces the Related Work; Section 3 describes our modified ternary search algorithm; Conclusion and future scope in Section 4; given acknowledgments in Section 5 and all the used references are given in section 6.

2. RELATED WORK
A well defined set of instructions for solving a particular kind of problem. Algorithms exist for systematically solving many types of problems like sorting, searching etc.

2.1 Linear Search

```
// Search for a matching String val in the array vals.
// If found, return index. If not found, return -1.
int LinearSearch(int Array[ ], int val) {
// Loop over all items in the array
for (int i=0; i<vals.length; i++) {
// Compare items
int rslt = val.compareTo( vals[i] );
if ( rslt == 0 ) {              // Found it
return i;                   // Return index
}
}
return -1;     // If we get this far, val was not found.
```

Systematically enumerate all possible values and compare to value being sought. For an array, iterate from the beginning to the end, and test each item in the array.

2.1.1 Complexity Analysis of Linear Search
We can have three cases to analyze an algorithm:

Worst Case Analysis (Usually Done)
In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be Θ(n).

Average Case Analysis (Sometimes done)
In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity

$$Average\ Case\ Time = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1) * (n+2)/2)}{(n+1)}$$

$$= \theta(n)$$

Best Case Analysis (Bogus)
In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the

first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be $\theta(1)$

2.2 Binary Search

Quickly find an item (val) in a sorted list.

Procedure:

1. Init min and max variables to lowest and highest index
2. Repeat while min ≤#max,
   a. Compare item at the middle index with that being sought (val)
   b. If item at middle, equals val, return middle
   c. If val comes before middle, then reset max to middle-1
   d. If val comes after middle, reset min to middle+1
3. If min >#max, val not found

```
// Search for a matching val String in the String array vals
// If found, return index. If not found, return -1
// Use binary search.
 int bSearch(String val, String[] vals) {
  int min = 0;
  int max = vals.length-1;
  int mid;
  int rslt;
  while (min <= max) {
            mid = int( (max + min)/2 );        // Compute next index
            rslt = val.compareTo( vals[mid] );  // Compare values
                  if ( rslt == 0 ) {              // Found it
                  return mid;                    // Return index
                  } else if ( rslt < 0 ) {         // val is before
vals[mid]
                  max = mid - 1;                 // Reset max to item
before mid
                  } else {                      // val is after vals[mid]
                  min = mid + 1;                 // Reset min to item
after mid
    }
   }
   // If we get this far, val was not found.
   return -1;
}
```

Complexity Analysis of Binary Search

After 1st iteration, N/2 items remain ($N/2^1$)

After 2nd iteration, N/4 items remain ($N/2^2$)

After 3rd iteration, N/8 items remain ($N/2^3$)

Search stops when items to search ($N/2^K$) → 1

i.e. $N = 2^K$, $\log_2(N) = K$

Worst case: Number of iterations is $\log_2(N)$

It is said that Binary Search is a logarithmic algorithm and executes in O(logN) time.

3. OUR MODIFIED TERNARY SEARCH ALGORITHM

In Ternary Search Algorithm we divide the given array into three equal parts. The first part contains the elements from index 0 to index n/3, second part contains the elements from index $(\frac{n}{3} + 1)$ to $\frac{2n}{3}$ and the third part contains the elements from index $(\frac{2n}{3} + 1)$ to n. The element be to search can lie in either of the three parts.

1. Init min and max variables to lowest and highest index
2. Repeat while min ≤ max,
   a. calculate middle1 = (min + max)/3 and middle2 = (middle1+max)/2.
   b. Compare item at the middle1 and middle2 index with that being sought (val)
   c. If item at middle1 or middle2, equals val, return val found
   d. If val comes before middle1, then reset max to middle1-1
   e. If val comes after middle2, reset min to middle2+1
   f. If val comes after middle1 and before middle2 then reset min to middle1+1 and max to middle2-1.
3. If min > max, val not found

Procedure:

3.1 Complexity Analysis of Binary Search

After 1st iteration, N/3 items remain ($N/3^1$)

After 2nd iteration, N/9 items remain ($N/3^2$)

After 3rd iteration, N/27 items remain ($N/3^3$)

Search stops when items to search ($N/3^K$) → 1

i.e. $N = 3^K$, $\log_3(N) = K$

Worst case: Number of iterations is $\log_3(N)$, which is better than binary search searching time $\log_2(N)$

4. CONCLUSION AND FUTURE SCOPE

From the results it can be concluded that Ternary Search Algorithm is working well for all length of input values. It takes lesser CPU time than existing searching algorithm linear search and binary search. In the future more effective searching algorithm can be proposed.

5. ACKNOWLEDGMENTS

My heartily thanks to my wife Ms. Mamta Martolia Arora, Assistant Professor, WIT Dehradun for her valuable support in writing this paper.

6. REFERENCES

[1] Arora, N., Tamta, V., and Kumar S. 2012. A Novel Sorting Algorithm and Comparison with Bubble Sort and Selection Sort. International Journal of Computer Applications. Vol 45. No 1. 31-32

[2] Herbert Schildt Tata McGraw-Hill [2005], "The Complete Reference C fourth Edition"

[3] Alfred V., Aho J., Horroroft, Jeffrey D.U.(2002) Data Structures and Algorithms.

[4] Frank M.C. (2004) Data Abstraction and Problem Solving with C++. US: Pearson Education, Inc

[5] Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C.(2003) Introduction to Algorithms MIT Press, Cambridge, MA, 2nd edition

[6] Seymour Lipschutz (2009) Data Structure with C, Schaum Series, Tata McGraw-Hill Education.