Ternary Search

Mohammad Arshad Hossain Ratul
Id: 1930319
Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh
Email:1930319@iub.edu.bd

Syed Ali Redwanul Haider
Id: 1930580
Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh
Email: 1930580@iub.edu.bd

Tunisha Yanoor Bristy
Id: 1930639
Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh
Email: 1930639@iub.edu.bd

Adiba Haque
Id: 1931051
Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh
Email: 1931051@iub.edu.bd

Abstract:
Algorithms like searching algorithms generally search for a specific data or element from a set of elements. There are uncountable searching algorithms varying with time and space complexity which are very available and implemented these days. The purpose of this research paper is on Ternary search algorithm. Ternary search algorithm is the algorithm which divides the set of elements into three subsets. We also compared similar algorithms along with complexity analysis and practical implementation. In short we have achieved results which define ternary search better on maximum major cases.
Keywords: Algorithm, Ternary, Search.

Introduction of the Paradigm of the Algorithm

Divide and conquer is an algorithm which repeatedly splits a problem into two or more sub-problems of the same or similar type, until it becomes easy enough to solve the problem precisely. The solutions to sub-problems are then united together to give solutions to the initial problem. Divide and conquer works swiftly because they lead to less work. Early examples of these

algorithms are called decrease and conquer for the single-subproblem category which can be solved recursively. A direct explanation of the design on CPU presented in 1946 in a paper by John Mauchly, the design of using a sorted list of items to simplify searching in the time as long as Babylonia in 200 BC. One more prehistoric decrease and conquer algorithm is the Euclidean algorithm to calculate the common measure of two numbers by lessening it to smaller and smaller subproblems, which goes back to several centuries BC . An early example of this paradigm with many subproblems is Gauss's 1805 which is now called Cooley-Tukey fast Fourier transform algorithm which he did not examine its functioning outcomes. A primary two subproblem of divide and conquer algorithms respectively designed for computers and properly studied is the merge sort algorithm, invented by John von Neumann in 1945. In a paradigm such as divide and conquer there are advantages and disadvantages such as it lets us to solve difficult problems, it assists us to discover productive algorithms, and they are useful at memory caches but the drawbacks are an exact stack may worn out the space, it may even crash the system if the repetition is executed strictly greater than the stack present in the CPU. [4]

Introduction of the Selected Algorithm

Ternary search is a divide and conquer method that helps us through the process of finding a specific element in a sorted array. It's equivalent to binary search where we split the array into two subsets, but this technique divides the array into three subsets and finds which one has the searched element. [6] Ternary search is an approach in computer science for discovering the highest or lowest of an unimodal task. The time complexity of ternary search technique is O(log3 n) and the space complexity is O(1).

Pseudocode:

Recursive:

```
Ternary Search(Left, Right, Key, A[])
            if r >= Left
                Mid1 = Left + (Right -Left) / 3
                Mid2 = Right - (Right - Left )/3

               if A[Mid1]  = key
                   return Mid1

               if A[Mid2]  = key
                   return Mid2

               If key<A[Mid1]
                   return Ternary Search(
Left, Mid1 - 1, Key,  A)

               else if key > A[Mid2]
                  return Ternary Search(Mid2
+ 1, Left, Key, A)
               else
                 return Ternary Search (Mid1
+ 1, Mid2-1, Key, A)
 return -1
```

Iterative Approach:

```
TernarySearch(int Left, int Right, int Key, int A[])

    while Right >= Left

        int Mid1 = Left + (Right - Left) / 3
        int Mid2 = Right - (Right - Left) / 3

        if (A[Mid1] = Key
            return Mid1;

        if (A[Mid2] = Key
            return Mid2;

        if (Key < A[Mid1]

            Right = Mid1 - 1

        else
            if Key > A[Mid2]

            Left = Mid2 + 1

        else

            Left = Mid1 + 1
            Right = Mid2 - 1

      return -1
```

Recursive Implementation:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int TernarySearch(int Left, int Right, int key, int A[])
{
   if (Right >= Left) {

      int Mid1 = Left + (Right - Left) / 3;
      int Mid2 = Right - (Right - Left) / 3;

      if (A[Mid1] == key) {
         return Mid1;
      }
      if (A[Mid2] == key) {
         return Mid2;
      }

      if (key < A[Mid1]) {

          return TernarySearch(Left, Mid1 - 1, key, A);
      }
      else if (key > A[Mid2]) {

         return TernarySearch(Mid2 + 1, Right, key, A);
      }
      else {

         return TernarySearch(Mid1 + 1, Mid2 - 1, key, A);
      }
   }

   return -1;
}

int main()
{
   int Left, Right, p, key;

   int A[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

   Left = 0;
```

```
    Right = 9;
    key = 5;
    p = TernarySearch(Left, Right, key, A);

    cout << "Index of " << key
        << " is " << p << endl;

    key = 50;
    p = TernarySearch(Left, Right, key, A);
    cout << "Index of " << key
        << " is " << p << endl;
}
```

Iterative Approach Implementation:

```
#include <iostream>
using namespace std;

int TernarySearch(int Left, int Right, int Key,
int A[])
{
    while (Right >= Left) {

        int Mid1 = Left + (Right - Left) / 3;
        int Mid2 = Right - (Right - Left) / 3;

        if (A[Mid1] == Key) {
            return Mid1;
        }
        if (A[Mid2] == Key) {
            return Mid2;
        }

        if (Key < A[Mid1]) {

            Right = Mid1 - 1;
        }
        else if (Key > A[Mid2]) {

            Left = Mid2 + 1;
        }
        else {

            Left = Mid1 + 1;
            Right = Mid2 - 1;
        }
    }

    return -1;
}

int main()
{
    int Left, Right, p, Key;

    int A[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    Left = 0;

    Right = 9;

    Key = 5;

    p = TernarySearch(Left, Right, Key, A);

    cout << "Index of "<<Key<<" is " << p <<
endl;

    Key = 50;

    p = TernarySearch(Left, Right, Key, A);

    cout << "Index of "<<Key<<" is " << p;
}
```

Complexity using substitution method:

T(n) = T(n/3) + O(1)

= T(n/3) + C

= T($n/3^2$) + C + C

= T($n/3^3$) + C + C + C

= T($n/3^4$) + C + C + C + C

.

.

.

= T($n/3^k$) + k*C

Let,

$n/3^k$= 1

n = 3^k

$log_3 n = log_3 3^k$

$log_3 n$ = k

Substituting, $n/3^k$ and k in the equation.

∴ T(1) + $C * log_3 n$

O($log_3 n$)

≈ O($log\ n$) [1]

Complexity using master theorem:

T(n) = T(n/3) + O(1)

Here,

a = 1, b = 3 and f(n) = 1

So,

$n^{log_a^b} = n^{log_3^1} = n^0 = 1$

Thus,

Case 2:

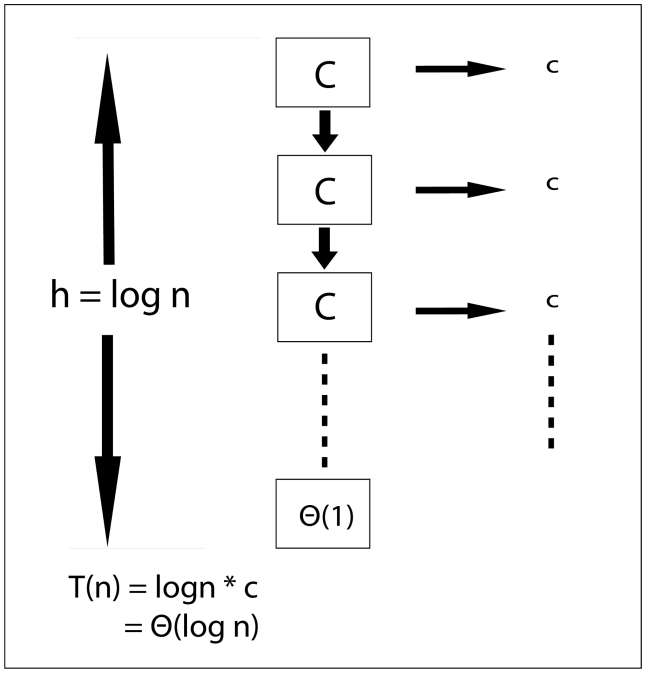
T(n) = Θ($1 * log^{(0+1)} n$)

= Θ($log^1 n$)

= Θ($log\ n$)

Complexity using recursion tree:

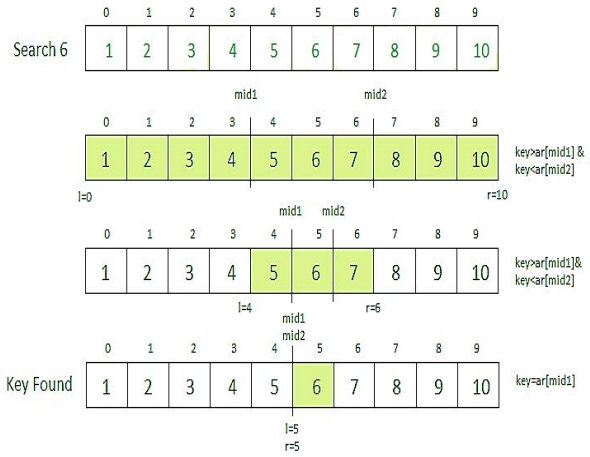
T(n) = T(n/3) + O(1)



Use Cases:

The fundamental idea of ternary search is to determine the highest or the lowest element out of the set of elements or from unimodal function. Basically it divides the set into 3 subsets with two middle elements or keys. In this chance if the targeted element matches with any one of the two keys, the program will return that significant key which matched. If not then the search function will check where the targeted element is lower than the first middle key(M1) . If yes then the search is again implemented on the portion which is lower than the first middle key(M1). The pursuit will be implemented between the 2nd middle key(M2) and 1st middle key(M1). If the targeted element is greater than the 1st middle key(M1) and

if the targeted element is greater than the 2nd middle key(M2), the search will be implemented on the elements greater than the 2nd middle key overlooking rests. The process will follow recurrence until one of the middle key matches with the targeted element. On that chance if the targeted element is not found then the function returns -1. This technique is implemented on puzzle solvings, map paths and many others which is mentioned in the section of recommended use.
[5][6]



Recommended Use:

This unimodal function can figure the minimum distance between two coordinates. This ability of the algorithm can help to figure out the minimum distance between two moving objects like planes, matrix sum or comparing arrays to find sub arrays. The search algorithm also can be used in maps to find the shortest path between two points or solving puzzles.[3]

Comparison:

Binary search is similar to ternary search, both works only when the array is sorted but with different searching methods and they both follow divide and conquer paradigm.
Binary search divides the array into two parts, whereas ternary search divides it into three parts, which comes to a conclusion that binary search has a complexity of O($log_2 n$) and ternary has a complexity of O($log_3 n$). Seeing their complexity we can say ternary search works faster than binary search. [1] [5].
But when we consider the constants we can see ternary is much slower than binary search.
Proof:
Binary search:
T(n) = T(n/2) + 2c

$= \frac{2C \log n}{\log 2}$ + c1

Ternary Search:
T(n) = T(n/3) + 4c

$= \frac{4C \log n}{\log 3}$ + c1

Therefore, ignoring c1 and denominator log X.
We can say binary is 2C*log n and ternary is 4C*log n.
Hence, binary search is faster and is preferred more than ternary search.
Why do we use ternary over binary search sometimes?

- Ternary search has more capability to work with huge data.
- Ternary search can be used to find extreme values in unimodal functions, which binary search cannot.

Therefore we can say all types of problems which can be solved with binary search can be solved with ternary but all problems solved with ternary search cannot be solved with binary search.

Conclusion:

Throughout the paper we investigated the ternary search algorithm and its searching techniques and abilities. We have overviewed its history and determined its space & time complexity by analysing pseudocode. By our findings we can justify that binary search is convenient in some cases but also we have observed that ternary is a suitable algorithm to find extreme maximum and minimum for unimodal functions and we can conclude that all types of problems which can be solved with binary search can be solved with ternary but all problems solved with ternary search cannot be solved with binary search.

Reference:

1. Nitin Arora, Mamta Martolia Arora & Esha Arora
   A Novel Ternary Search Algorithm, International Journal of Computer Applications (0975 – 8887) Volume 144 – No.11, June 2016.
2. https://www.geeksforgeeks.org/ternary-search/
3. https://www.hackerearth.com/practice/algorithms/searching/ternary-search/practice-problems/
4. https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm#Divide_and_conquer
5. Manpreet Singh Bajwa, Arun Prakash Agarwal & SumatiManchanda
   Ternary Search Algorithm: Improvement of Binary Search

   2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)
6. https://www.geeksforgeeks.org/ternary-search/
7. https://en.wikipedia.org/wiki/Ternary_signal