Ternary search

Indian Institute of Information technology, Allahabad

DAA ASSIGNMENT-3 , GROUP 8

Swaraj Bhosle IIT2019024

Ritesh Raj IIT2019025

Utkarsh Garg IIT2019026

Abstract: In this Paper we have devised an algorithm to implement ternary search and find whether a word is present in a given array or not using ternary search

I. INTRODUCTION

A ternary search algorithm is a technique in computer science for finding the minimum or maximum of a unimodal 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. A ternary search is an example of a divide and conquer algorithm

This report further contains -

- Algorithm Description.
- Algorithm And Analysis.
- Time Complexity Analysis
- Conclusion

II. ALGORITHM DESIGN

We are given an $array$ of some numbers and we need to find weather an element x is present in the array or not using ternary search.

Steps for implementing the ternary search are as follows :

- Take the input n from the user
- Input the $array$ from the user.
- Input the element x which is needed to be checked.
- Apply Merge sort algorithm to sort the array.
- Take two variables l and r in which we store left index and right index.
- Call the recursive $ternarySearch$ function.
- In each call we calculate the middle two indexes by which we break down the array into 3 equal parts.
- If x is equal to any of $mid1$ or $mid2$ the we return true.
- For each of the three sections check if x lies in that range, if it does then again we call the recursive function, and reduce the searching region.
- if we reach the length of 1 and x is not found we return $false$.
- Finally if the recursive function returns $true$ we print element found in the array else print element not found.

Advantages of using merge sort:- It is quicker for larger lists because unlike insertion and bubble sort it doesn't go through the whole list several times. It has

a consistent running time, carries out different bits with similar times in a stage.

III. ALGORITHM AND ANALYSIS

Algorithm 1 Algorithm for ternary Sort

Input: Size of array, array and element x

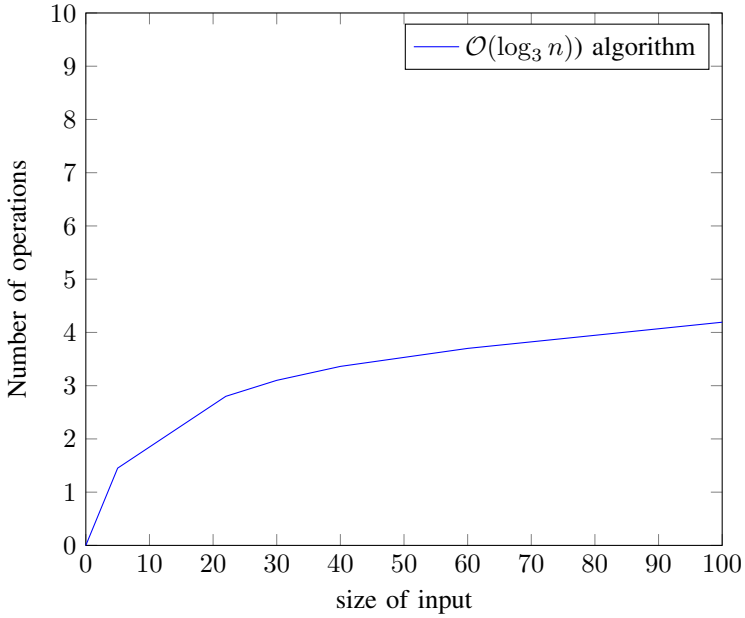
Output: True if x is present otherwise false.

Function `ternarySort` (*int arr,int l,int r,int x*) **:**

- $mid1 \leftarrow l + (r-l)/3$;
  $mid2 \leftarrow l + (r-l)/3$;
- **if** $l > r$ **then**
  - **return** $false$
- **if** $(arr[mid1] == x \;\; or \;\; arr[mid2] == x)$ **then**
  - **return** $true$;
- **else if** $x < ar[mid1]$ **then**
  - **return** $ternarySearch(l, mid1 - 1, x, ar)$;
- **else if** $x < ar[mid1]$ **then**
  - **return** $ternarySearch(l, mid1 - 1, x, ar)$;
- **else if** $x < ar[mid2]$ **then**
  - **return** $ternarySearch(mid1 + 1, mid2 - 1, x, ar)$;
- **else**
  - **return** $ternarySearch(mid2 + 1, r, x, ar)$;

IV. TIME CALCULATION

Complexity analysis for Naive $\mathcal{O}(n \log_3 n))$ algorithm



As the algorithm is logarithmic so it is a quite efficient algorithm, only very few operations are required to perform a check over a large array.

V. TIME COMPLEXITY

Let the average size of all strings be N. Then we merge sort the string in ascending and descending order and compare the obtained strings with the original string, and if the string matches with any of the sorted strings then we increment the answer.

The time complexity will be $\mathcal{O}(n \log n))$ due to the merge sort algorithm in a best, average and worst case. The time complexity will be affected due to the merge sort in sorting the 1D array and the total time complexity will become $\mathcal{O}(n \log n)$.

Here we study the complexity for ternary Sort,as the average cost of a successful search is about the same as the worst case where an item is not found in the array, both being roughly equal to $\mathcal{O}(\log_3 N)$. So, the average and the worst case cost of binary search, in big-O notation, is $\mathcal{O}(\log_3 N)$, the best case is when in the

first attempt x is found,so best case time complexity is $\mathcal{O}(1)$.

BEST	AVERAGE	WORST CASE
$\mathcal{O}(1)$	$\mathcal{O}(\log_3 N)$	$\mathcal{O}(\log_3 N)$

VI. CONCLUSION

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 but 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.

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.

VII. REFERENCES

1) https://doc.lagout.org/Alfred%20V.%20Aho%20-%20Data%20Structures%20and%20Algorithms.pdf
2) https://web.ist.utl.pt/~fabio.ferreira/material/asa/clrs.pdf
3) https://www.geeksforgeeks.org/ternary-search/