Quaternary Search

Vidushi Pathak(IIT2019027), Shubham Netam(IIT2019028), Puneet Singhwaiya(IIT2019029)
B.Tech. 4th semester, Department of Information Technology.
Indian Institute of Information Technology, Allahabad

***Abstract*—In this paper we have discussed an algorithm to implement Quaternary Search. We have designed two algorithm, one is iterative and the other one is recursive. We have also discussed the time and space complexity of both the methods.**

I. Introduction

Quaternary search is like binary search, but divides the array into four parts instead of two. After evenly dividing the array, the three divisors are compared to the input value. If it matches the index is returned. If not then algorithm check for the value in subarray that contains the value. This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

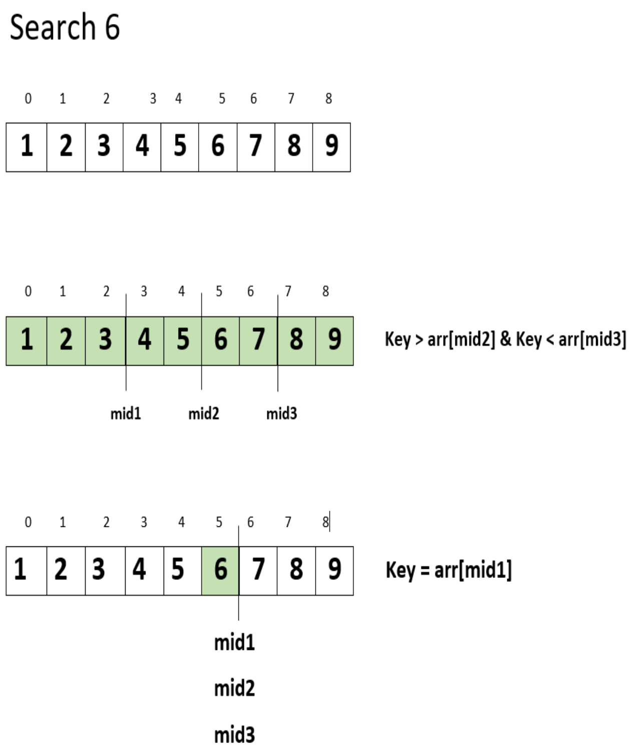


Fig. 1. Representation of quaternary search.

II. Algorithm

Our algorithm will first divide the array into 4 parts by finding three mid-points then it will check if the element which we want to find is present at any of the mid-points or not. If present we will return the index else we will check in which subarray the element is present and then move the mid-point accordingly.

III. Pseudo Code

A. Iterative Method

```
  Procedure quaternarySearch
A ← sorted array
n ← size of array
x ← value to be searched

Set left = 0
Set right = n-1

while left <= right

  mid1 = left + (right-1)/4
  mid2 = mid1 + (right-1)/4
  mid3 = mid2 + (right-1)/4

  if arr[mid1] is equal to x
        return mid1

  if arr[mid2] is equal to x
        return mid2

  if arr[mid3] is equal to x
        return mid3

  if arr[mid1] >x
        right = mid1-1

  else if arr[mid2] >x
        left = mid1+1
        right = mid2-1

  else if arr[mid3] <x
        left = mid3+1

  else if arr[mid2] <x and arr[mid3] >x
        left = mid2+1
        right = mid3-1
end while
return -1
end procedure
```

B. Recursive Method

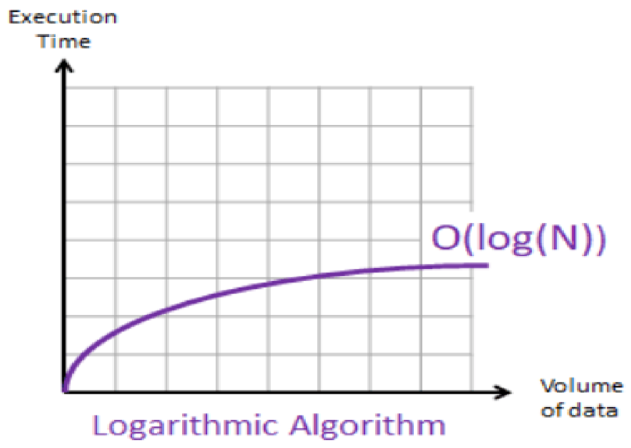
```
  Procedure quaternarySearch
A ← sorted array
```

```
n ← size of array
x ← value to be searched

Set left = 0
Set right = n-1

if left <= right

   mid1 = left + (right-1)/4
   mid2 = mid1 + (right-1)/4
   mid3 = mid2 + (right-1)/4

   if arr[mid1] is equal to x
         return mid1

   if arr[mid2] is equal to x
         return mid2

   if arr[mid3] is equal to x
         return mid3

   if arr[mid1] >x
         return quaternarySearch(arr,left,mid1-1,x)

   else if arr[mid2] >x
         return quaternarySearch(arr,mid1+1,mid2-1,x)

   else if arr[mid3] <x
         return quaternarySearch(arr,mid3+1,right,x)

   else if arr[mid2] <x and arr[mid3] >x
         return quaternarySearch(arr,mid2+1,mid3-1,x)

return -1
end procedure
```

IV. Experimental Study

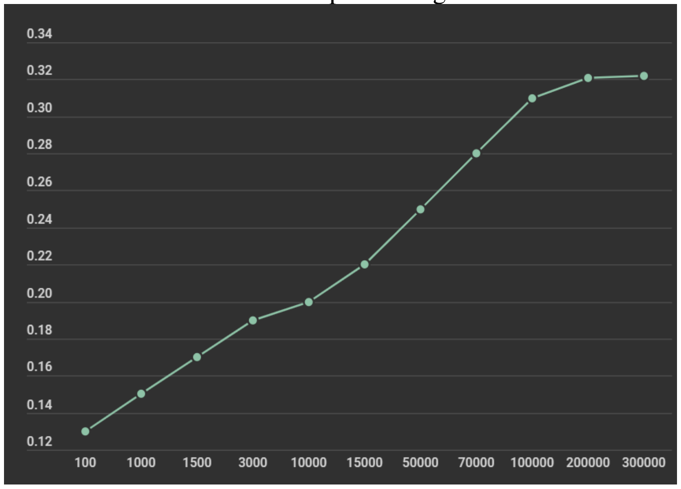
A. Apriori Analysis

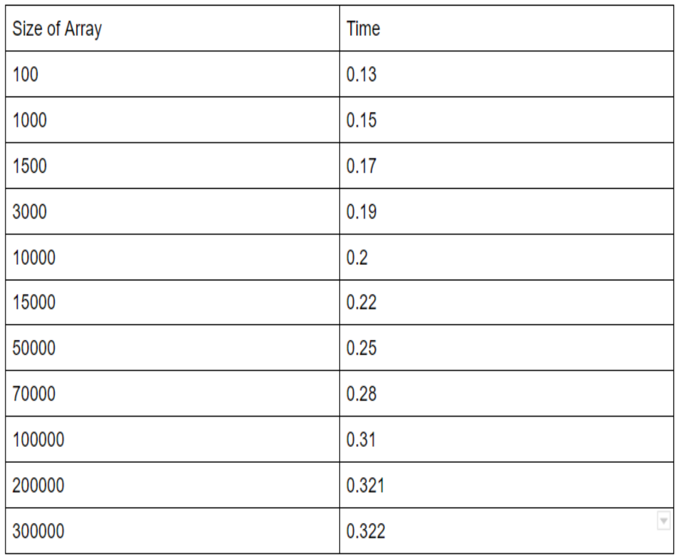
Apriori analysis means,analysis is performed prior to running it on a specific system.This analysis is a stage where a function is defined using some theoretical model.Hence,we determine the time and space complexity of an algorithm by just looking at the algorithm rather than running it on a particular system with a different memory,processor, and compiler.So,as we discussed under the heading complexity analysis we arrived at the conclusion that the best time complexity is O(log(n)).



B. Aposteriori Analysis

Aposteriori analysis of an algorithm means we per- form analysis of an algorithm only after running it on a system.It directly depends on the system and changes from system to system.So for the a aposteriori analysis of the algorithm,we have run our code on the compiler and get values of the time.





Size of Array	Time
100	0.13
1000	0.15
1500	0.17
3000	0.19
10000	0.2
15000	0.22
50000	0.25
70000	0.28
100000	0.31
200000	0.321
300000	0.322

V. Complexity Analysis

A. Time Complexity

In this algorithm at each iteration the array is divided by 4. Suppose length of array is n and after K iteration the length of array becomes 1.Then we get,
n/$4^k = 1$
$n = 4^k$
$log(n) = log(4^k)$
$log(n) = klog(4)$
$k = log(n)$

Therefore, Time complexity is O(log(n))

B. Space Complexity

In the iterative method, the space complexity would be O(1). While in the recursive method, the space complexity would be O(log(n)).

VI. Conclusion

Through this assignment we concluded that although Quaternary can seem faster than binary search, but is not because it do more comparisons per step.

VII. References

1.https://stackoverflow.com/questions/39845641/quaternary-search-algorithm
2.https://github.com/estensen/quaternary-search