Optimising Binary Search Using Heuristics

DAA ASSIGNMENT-3 , GROUP 7

Medha Balani
IIT2019021

Saksham Aggarwal
IIT2019022

Utkarsh Gangwar
IIT2019023

a) ***Abstract: In this paper we have described an algorithm that is an improved version of binary search using heuristics . This algorithm optimizes the worst and average case of binary search algorithm . We have also done the apriori and aposteriori analysis of our algorithm.***

I. INTRODUCTION

Searching Algorithms are used to check for an element in a data structure or retrieve its information if it exists.Search algorithms can be classified based on their mechanism of searching.
Linear search algorithms check every record for the one associated with a target key in a linear fashion.
Binary searches, repeatedly target the center of the search structure and divide the search space into half.
Comparison search algorithms improve on linear searching by successively eliminating records based on comparisons of the keys until the target record is found, and can be applied on data structures with a defined order.
Finally, hashing directly maps keys to records based on a hash function.
Heuristic is a technique designed for solving a problem more quickly when trivial methods are too slow, or for finding an approximate solution when trivial methods fail to find any exact solution.
Binary search is the most popular Search algorithm.It is efficient and also one of the most commonly used techniques that is used to solve problems.

If all the names in the world are written down together in order and you want to search for the position of a specific name, binary search will accomplish this in a maximum of 35 iterations.

Binary search works only on a sorted set of elements. To use binary search on a collection, the collection must first be sorted.

When binary search is used to perform operations on a sorted set, the number of iterations can always be reduced on the basis of the value that is being searched.

II. ALGORITHM DESIGN

A. Problem Statement

We are required to return the index of the element key in the array Arr which is sorted in non-decreasing order.We return -1 if key doesn't occur in Arr.

B. Algorithm Description

Similar to the binary search algorithm we initialise the lower bound $l = 0$(lowest index in the array Arr) and Upper bound $r = n - 1$ (where n is the size of array Arr).

In binary search we repeatedly compared the middle element of our search space with the given key , this indicates that if the key is present in the left or right end of the array Arr binary search would take comparatively more iterations to find it out than it would take if element is in the middle part of the array Arr.

In this algorithm we not only compare the middle element of our search space with key but also compare key with left and right bounds , so that we don't take many iterations to find elements present at the ends.

Similar to Binary Search we reduce our search space to half based on the comparison between the values of key and element present at middle in the array Arr.

C. Pseudo-code

Algorithm 1: Optimised Binary search

Input: Sorted Array Arr of size n and key
Output: Index of key

```
Function OptimisedBinarySearch(Arr,n,key):
    l ← 0
    r ← n − 1
    Ans ← −1
    while l ≤ r do
        mid ← l + (r−l)/2
        if Arr[l] = key then
            Ans ← l
            return Ans
        if Arr[r] = key then
            Ans ← r
            return Ans
        if Arr[mid] = key then
            Ans ← mid
            return Ans
        if Arr[mid] < key then
            l ← mid + 1
        else
            r ← mid − 1
    return Ans
```

III. ALGORITHM ANALYSIS

A. Apriori Analysis

Table : Time Complexity			
Steps	Time	Best Freq.	Freq.
2,3,4	3	1	1
5	1	1	$logN$
6	3	1	$logN$
7,8,9	4	1	$logN$
10,11,12	4	1	$logN$
13,14,15	4	1	$logN$
16	1	0	$logN$
17	2	1	$logN$
18	1	0	$logN$
19	2	1	$logN$

From the above table we can say that best case for this algorithm would be when key is present at either of mid , end or start of the array Arr.

$T_\Omega \propto 3*1+1*1+3*1+4*1+4*1+4*1+1*0+2*1+1*0+2*1$

$T_\Omega = \Omega(1)$

While in the worst Case :

$T_O \propto 3*1+1*logN+3*logN+4*logN+4*logN+4*logN+1*logN+2*logN+1*logN+2*logN$

$T_O \propto 3+21*logN$

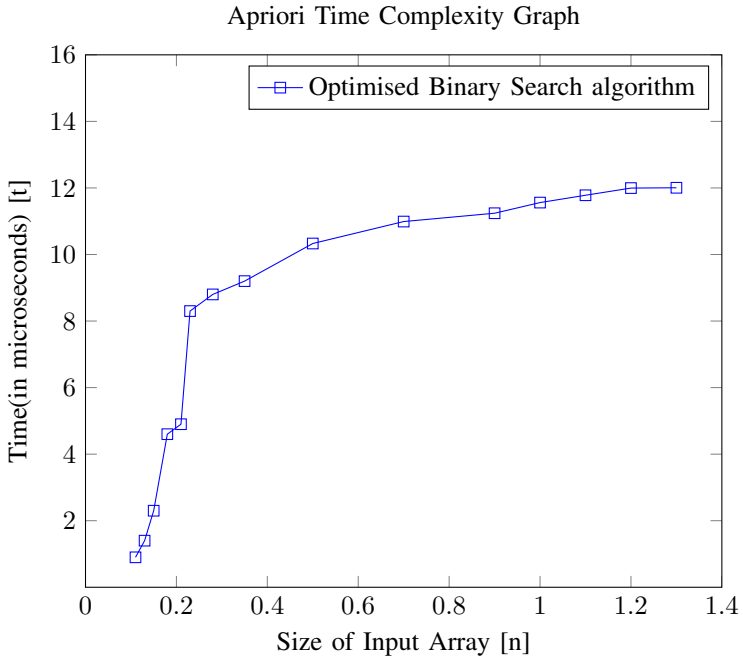
$T_O = O(logN)$

B. Space Complexity

Since no extra space is used in this algorithm , so auxiliary space is constant. Only the input array is of size n. So , Space Complexity = Input Space + Auxiliary Space = O(n), this Algorithm uses linear Space .

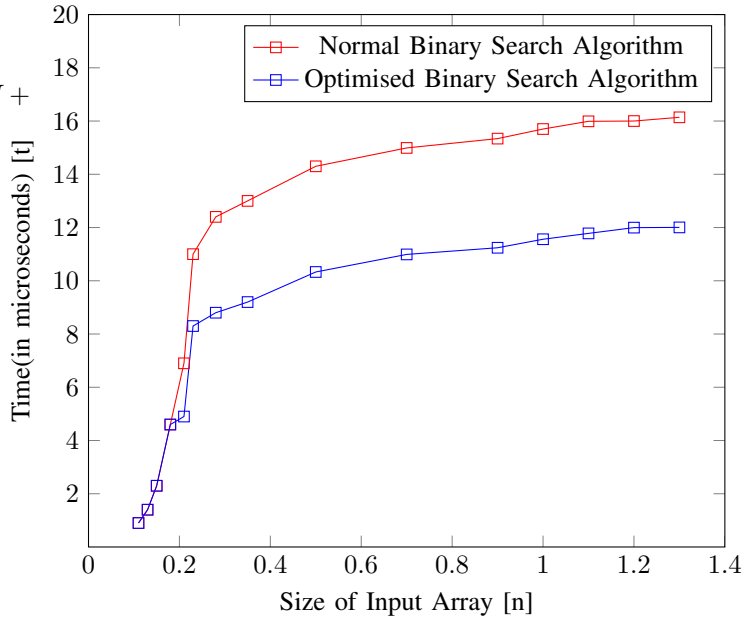
IV. EXPERIMENTAL STUDY

A. Graph-1



B. Graph-2

Comparison of Normal and Optimised Binary Search Algorithms



From the above graphs we can conclude that by using heuristics we can optimise binary search algorithm in the worst case as well as average case.
Also the graphs are consistent with the apriori analysis.

V. ILLUSTRATION

Let the given array be: {5,10,17,30,35,65,80,120,150,200}
Key is the element to be searched

Case 1 : Element is at first or last position

Key = 200
Using Binary Search -
First iteration: low =0 high =9 mid =4 a[4] =35<200
low= mid+1 =5
Second iteration: low =5 high =9 mid =7 a[7] =120<200
low= mid+1 =8
Third iteration: low =8 high =9 mid =8 a[8] =150<200
low= mid+1 =9
Fourth iteration: low =9 high =9 mid =9 a[9] =200 == Key,
Total iterations = 4
Using improved Binary Search -
First iteration: low = 0 , high = 9 mid =4
a[high]= 200 == key
Total iterations = 1

Case 2 : Element present in first half

Key = 30
Using Binary Search -
First iteration: low = 0 , high = 9 mid =4
a[4] =35 >30
high = mid -1 = 3
Second iteration: low =0 high =3 mid = 1
a[1]=10<30
low =mid +1= 2
Third iteration: low =2 high =3 mid = 2
a[2] = 17 < 30
low = mid +1 = 3
Fourth iteration: low =3 high=3 mid = 3
a[3]=30 == Key
Total iterations =4
Using Improved Binary Search -
First iteration: low = 0 , high = 9 mid =4
a[4]=35 >30
high = mid -1= 3 low =low +1
Second iteration: low = 1 high = 3 mid = 2
a[high]=30 ==Key
Total iteration = 2

VI. CONCLUSION

The above proposed algorithm improves traditional binary search algorithm by eliminating redundant iterations by adding a few if statements to also check at left and right bounds along-with the middle element.But the overall complexity of the algorithm still remains O($logN$).

VII. REFERENCES

1) https://www.hackerearth.com/practice/algorithms/searching/binary-search/tutorial/
2) https://en.wikipedia.org/wiki/Heuristic_(computer_science)