Design and Analysis of Algorithms Assignment - 4

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

Suhaib Khan(IIB2019039), Sangam Barnwal(IIT2019094), Sanjana Reddy (IIT2019095)

Abstract: ***Given an array and a value x,the problem is to find the floor of x which is the largest element in the array greater than or equal to x.In this paper,we will see the algorithm using divide and conquer approach to solve the problem.Then we will analyse the time complexity of the algorithm.We will see a O(log n) solution which will be much faster than the brute force approach.***

Index Terms: ***Arrays, Divide and Conquer, Binary search***

INTRODUCTION

We are given an array and a value x whose floor is to be found.A naive solution can be to traverse through the array and find the first element greater than x. The element just before the found element is the floor of x.This is similar to linear search and gives the O(n) solution.If this solution uses linear search ,we can get the intuitive idea of using binary search to solve the problem efficiently.

Divide and Conquer Divide and Conquer is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.

1. **Divide:** Break the given problem into subproblems of same type.
2. **Conquer:** Recursively solve these subproblems.
3. **Combine:** Appropriately combine the answers.

This report further contains:

- Algorithm Designs
- Algorithm Analysis
- Experimental Study and Profiling
- Conclusion
- References
- Appendix

ALGORITHM DESIGN

The algorithm is based on binary search which is a divide and conquer approach.Given the array and x,we will first calculate the middle index of array,which is the divide step as it divides the array into half.Then depending on the condition that x is greater than or less than the middle element we will recursively call the function but with upper half or lower half of array respectively,which is the conquer step.The array gets divided in half and function returns when middle element of the current array is equal to x or when array has one or less element.This procedure gets close to the number step by step.

Algorithmic Steps:

1. Create three variables l = 0, mid and r = n-1.
2. Recurse until and unless low is less than high.
3. check if the middle ((l + r) /2) element is equal than x, if yes then return the mid index.
4. else check if middle element is greater than x ,if yes then update the r, i.e r = mid - 1, and recurse. In this step we are reducing the search space to half.
5. Else update the l, i.e l = mid +1 and recurse.
6. when l $\geq$r ,return l.
7. If arr[l] = x,floor = x.
8. If arr[l] $<$ x,floor = arr[l].
9. If arr[l]$>$ $x$and l$\neq$ 0, $floor = arr[l-1]$
10. else floor not present.

```
int:
Function floorSearch(int arr[],int l.int r,int x)
if (l>=r)
    return l

else
  int mid =(l+r)/2
  if (arr[mid] == x)
          return mid
  else if (arr[mid] > x)
          return floorSearch(arr, l, mid - 1, x)
  else
          return floorSearch(arr, mid + 1, r, x)
```

```
int:
Function main()
  int arr[]
  input arr
  int n=sizeof(arr) / sizeof(arr[0])
  int x
  input x
  int result = floorSearch(arr, 0, n - 1, x);

   if(arr[result]==x)
        print x
    else if(arr[result]<x&&arr[result+1]>x)

                print arr[result];

    else if(arr[result]>x&&result!=0)
                print arr[result-1];
        else
            print -1;
```

ALGORITHM ANALYSIS

APRIORI ANALYSIS: We have tried to do the Apriori Analysis of the approach based on Divide and Conquer.

Let T(n) and S(n) is the time and space respectively.

TIME COMPLEXITY DERIVATION: Let Time complexity of above algorithm be T(n). In the base case the funtion compares l,r and returns which takes constant time.Otherwise, comparison and calculation of the middle element will take constant time and then the problem is divided into another problem of size n/2 (either floorSearch(arr, l, mid - 1, x) or floorSearch(arr, mid + 1, r, x).

TIME COMPLEXITY DERIVATION:

T(n) = T(n/2) + c

Using above relation, we get for T/2, T/8 etc as:
T(n/2) = T(n/4) + c
T(n/4) = T(n/8) + c
T(n/8) = T(n/16) + c and so on.....
.
.
.
.
$T(n)=T(n/2^k)+c+c+c...+c$
when $n=2^k, T(n)=T(1)+c*log_2 n$

DIFFERENT CASES: Now let us consider our algorithm in different scenarios.

BEST CASE: The time complexity of Divide and Conquer approach in the best case is O(1) ,as there will be one element in the array which will be compared and returned in constant time.

AVERAGE CASE: The average case complexity is same as worst case that is O(log n).

WORST CASE: The worst case compexity as derived from the recurrence realtion is O(log n).

SPACE COMPLEXITY: The space compexity of the algorithm is O(n).

PROFILING

So, after the above analysis of Apriori Analysis, we come to the Posteriori Analysis or Profiling. Now let us have the glimpse of space and time graph.

TIME ANALYSIS: The data for execution time(in ms) with respect to n is tabulated below.

n	time(ms)
100	0.014
1000	0.015
10000	0.039
100000	0.203
1000000	2.146

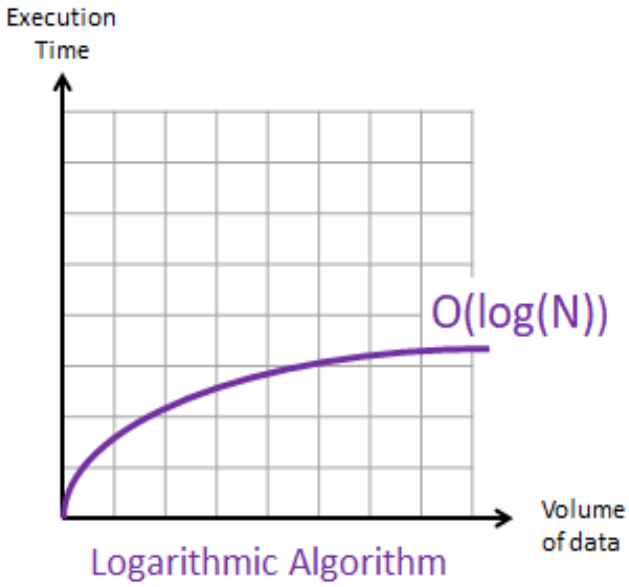

Figure 1: Time Complexity Graph

SPACE ANALYSIS: Following is the graph representing the space complexity of the algorithm.

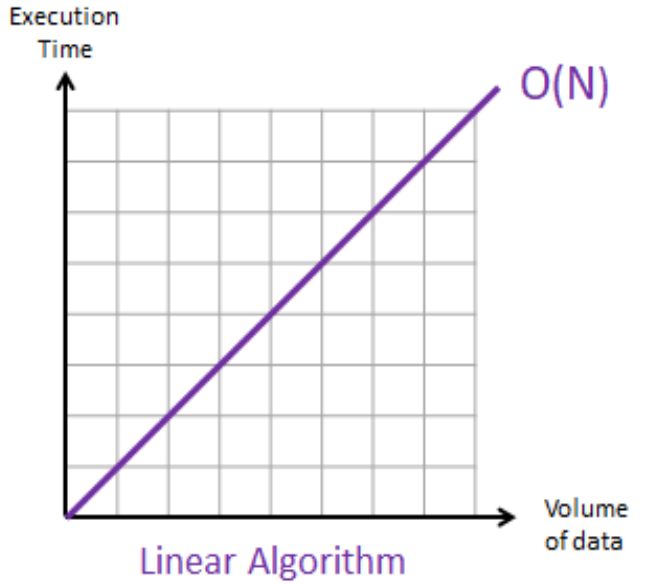

Figure 2: Space Complexity Graph

By the experimental analysis, we found that in case of optimized approach, on increasing value of n (increasing the number of elements in array) the graph is strictly increasing Thus the overall space increases with an increase in no. of elements

CONCLUSION

So, with the above mentioned algorithms and their profiling, we come to the conclusion that this problem of finding the floor of a given number in a sorted array is achieving its best time complexity of O(logn) and space complexity of O(n).

ACKNOWLEDGMENT

We are very much grateful to our Course instructor Dr Mohammed Javed and our mentor, Md Meraz, who have provided the great opportunity to do this wonderful work on the subject of Data Structure and Algorithm Analysis specifically on the programming paradigm of Divide and Conquer.

REFERENCES

1. Introduction to Divide and Conquer Technique: https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/

2. Introduction to Algorithms by Cormen,Charles, Rivest and Stein.
https://web.ist.utl.pt/ fabio.ferreira/material/asa

APPENDIX

To run the code, follow the following procedure:

1. Download the code(or project zip file) from the github repository.
2. Extract the zip file downloaded above.
3. Open the code with any IDE like Sublime Text, VS Code, Atom or some online compilers like GDB.
4. If required, save the code with your own desirable name and extension is .cpp
5. Run the code following the proper running commands(vary from IDE to IDE)
   (a) **For VS Code:** Press Function+F6 key and provide the input on the terminal.
   (b) **For Sublime Text:** Click on the Run button and provide the input.

Code for Implementation is:

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

int floorSearch(int arr[], int l, int r, int x)
{
        if (r > l) {
                int mid =(r + l) / 2;

                if (arr[mid] == x)
                        return mid;

                if (arr[mid] > x)
                        return floorSearch(arr, l, mid - 1, x);

                return floorSearch(arr, mid + 1, r, x);
        }
        if(r<=l)

                return l;



}
int main(void)
{
        int n;
        cin>>n;

        int arr[n];
        int x ;

        cin>>x;
        for(int i=0;i<n;i++)
        cin>>arr[i];
        int result =floorSearch(arr, 0, n - 1, x);

        if(arr[result]==x)
        cout<< x;
        else if(arr[result]<x&&arr[result+1]>x)
                {
                                cout<< arr[result];
                }
                else
                {
                        if(arr[result]>x&&result!=0)
                        cout<< arr[result -1];
                        else
                        cout<<-1;
                }
}
```