DAA432C Group-30 Assignment-04

B. Tech IT 4^{th} Semester Sec-A

Indian Institute of Information Technology, Allahabad

IIT2019092	IIT2019090	IIT2019091
Tanish Patel	Smitesh Hadape	Varun Bhardwaj
iit2019092@iiita.ac.in	iit2019090@iiita.ac.in	iit2019091@iiita.ac.in

***Abstract*— In this report, the design and analysis of an algorithm that finds the missing element in an array that represents elements of an arithmetic progression in order using divide and conquer approach.**

***Keywords*— Divide and conquer approach, arithmetic progression, array**

I. INTRODUCTION

This paper discusses about an algorithm that is designed to find the missing element in an array that represents elements of an arithmetic progression in order using divide and conquer approach. Arithmetic Progression (AP) is a sequence of numbers in order in which the difference of any two consecutive numbers is a constant value.

Since the elements of an array represent an arithmetic progression, the array is already sorted either in increasing or decreasing order. We have also analysed about the time and space complexities of the algorithm.

By the end of the paper, we will be able to understand the components of algorithm design and will learn different ways of analysing the algorithms.

II. ALGORITHM DESIGN

The given problem can be solved using divide and conquer approach which is similar to binary search.Basically we divide the given problem into smaller sub-problems and appropriately combine their solutions to get the solution to the main problem.

*Approach:*The idea is to keep on checking the difference between the middle element and its adjacent elements unless the difference is not equal to the desired common difference.

Algorithm:

1. Find the mid element of the array and initialise an answer variable as the smallest integer that can be stored.

2. Check the difference between the middle element and its previous element. If the difference

is not equal to the common difference of the AP, then store the missing number in an answer variable and make no further calls to the function, else proceed to next step.

3. Check the difference between the middle element and its next element. If the difference is not equal to the common difference of the AP, then store the missing number in a variable and make no further calls to the function, else proceed to next step.

4. If the current element is at its correct position, then divide the array into 2 halves, and perform the above steps in the later half, i.e. in the sub array from mid+1 till the end.

5. If the current element is not at its correct position, then divide the array into 2 halves, and perform the above steps in the first half, i.e. in the sub array from starting element till the mid.

6. After performing all the steps for all the subproblems, if the value of the answer variable is unchanged, then no element is missing in the array, otherwise, print the value of the answer variable.

III. PSEUDO CODE

```
Declare global variable ans=INT_MIN
Function missingTerm(Argument a[], Argument l, Argument h, Argument d)
{
    If l is greater than or equal to h
        return ;
    initialize m = (l + h) / 2
    initialize current=a[0]+m*d
    If a[$m$]-a[$m-1$] is not equal to d and m is greater than 0
        ans=a[$m-1$]+d
    Else if a[$m+1$]-a[$m$] is not equal to d and m+1 is less than h
        ans=a[$m$]+d
    Else if a[$m$] is equal to current
        missingTerm(a, m+1, h, d)
    Else
        missingTerm(a. l, m-1, d)
    return ;
end
}
  Main function(){
    Initialize integer array arr[]
    Initialize n as size of array
    Input the elements of the array
    If n is less than 3, print "invalid input"
    Else {
    Initialize d
    If a[2]-a[1] is equal to a[1]-a[0]
        d=a[1]-a[0]
    Else if a[3]-a[2] is equal to a[2]-a[1]
        d=a[2]-a[1]
    Else
        d=a[1]-a[0]
    missingTerm(a, 0, n, d)
    If ans is greater than INT_MIN
        print ans
    If ans is greater than INT_MIN
        print No term is missing
}
}
```

IV. ALGORITHM ANALYSIS

For the above approach based on divide and conquer we are effectively dividing the array in the array into 2 halves each time, until the missing element is found.

Calculating time complexity: Assume that k the function missingTerm is called k times.

- At each function call, the array is divided into 2 halves. Assume the length of the array before any function calls is n.

- After the 1st function call, length of array becomes n/2.

- After the 2nd function call, length of array becomes n/4.

- After the 3rd function call, length of array becomes n/8.

- After the kth function call, length of array becomes n/2^k.

- Since the length of the array becomes 1 after k function calls(worst case)

=> n = 2^k

Hence k = $\log_2$ (n)

Hence, the time complexity for the above approach is $\log_2$ (n).

Best Case

The best case arises when the element at the middle of the array is missing, i.e. the element at the middle position is not at the appropriate position in the AP. In this case, there are no function calls involved and hence the time complexity would be O(1).

TABLE 2

TIME COMPLEXITY OF BINARY SEARCH APPROACH

Class	**Time**
Worst case Complexity	O(log n)
Best Case Complexity	O(1)

The space complexity of the algorithm will be O(log n) because the number of recursive calls to the function increase after the value of the length of array is halved.

Space Complexity: O(logn)

VI. PROFILING

A. Time Complexity and Space Complexity:

In this section, the *Posteriori Analysis or Profiling has been discussed.* Now let us have the glimpse of time graph and then have a glimpse of the space graph.

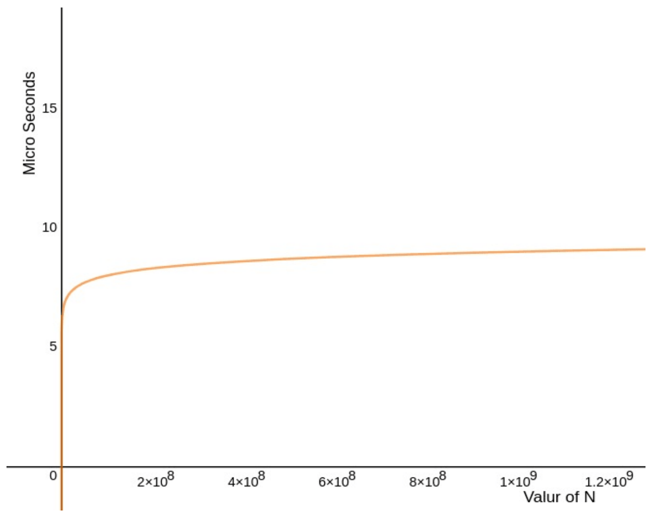

Figure 1: Time Complexity

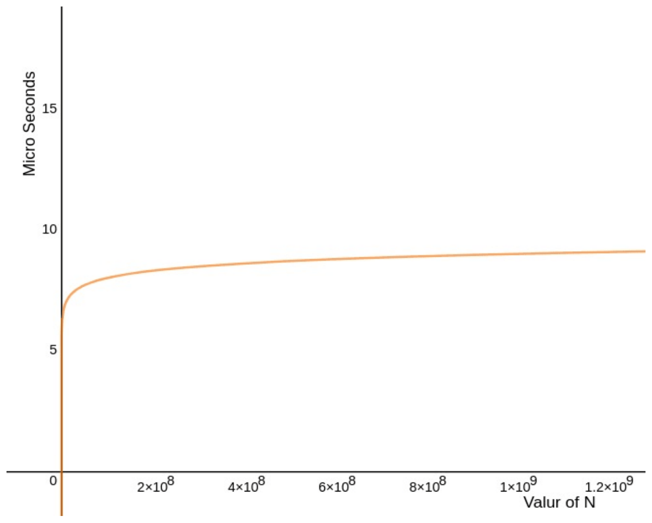

Figure 2: Space Complexity

VII. CONCLUSION

We can conclude that the above algorithm has the least time and space complexity to find the missing element in an array that represents elements of an arithmetic progression in order.

REFERENCES

[1] Introduction to Algorithms / Thomas H. Cormen . . . [et al.]. - 3rd edition.

[2] The Design and Analysis of Algorithms (Pearson) by A V Aho, J E Hopcroft, and J D Ullman

[3] Algorithm Design (Pearson) by

J Kleinberg, and E Tard missing-number-arithmetic-progression/
[4] https://www.geeksforgeeks.org/find-