DAA432C

Group-16 Assignment-01

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

Indian Institute of Information Technology, Allahabad

IIT2019099	IIT2019100	IIT2019101
Nitesh Rawat	Maitry Jadiya	Aryan Gupta
iit2019099@iiita.ac.in	iit2019100@iiita.ac.in	iit2019101@iiita.ac.in

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

***Keywords*— Arithmetic progression, Divide and conquer, array, Binary Search**

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.

An arithmetic sequence or progression(AP) is defined as a sequence of numbers in which for every pair of consecutive terms, the second number is obtained by adding a fixed number to the first one.

The array is already sorted either in increasing or decreasing order because, the elements of an array represent an arithmetic progression .

This paper also contains the analysis about the time and space complexities of the algorithm. By the end of the paper, we will be able to understand all the components of algorithm design and will learn different ways of analysing the algorithms.

II. ALGORITHM DESIGN

The given problem can be solved by divide and conquer algorithm.Here we will be using the binary search approach .We will divide a given problem into smaller subproblems and appropriately combine their solutions to get the solution to the main problem.

*Approach:*Idea is to compare the elements of the given array(A) with an array whose elements are in proper arithmetic progression(B). We will find the first index of mismatch of A and B.The element of B in this index will give us our missing element.

Algorithm:

1. Find the mid element of the array every time search range is divided and initialise a result variable which will keep index of mismatched index.Input array A with a missing element and an array B which has elements in proper arithmetic progression(also having the missing element of A) is taken.

2. Check the values of array A and B on the mid index and if the elements are same then it would mean that no element was missing in the AP till this index.In this case start searching only to the right of mid(right half of the current search range).

3. Check the values of array A and B on the mid index and if the elements are unequal then store the index in result and keep checking to the left of mid(left half of the current search range) as the minimum index of mismatched value is required.If a smaller index of mismatch is found then result is updated with this index.

4. The value of result is returned after the range becomes zero.

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

III. PSEUDO CODE

Function binarysearch(Argument a[],Argument b[], Argument n)
{ initialize l =0 , h=n-1 ; while l is less than h { initialize m = l+((h-l) / 2);
initialize res =-1;
If a[m]==b[m] l=m+1 range squeezes to right half
Else if a[m]!=b[m] res=mid, end=mid-1;
}
return res;
end
}
Main function(){
Initialize integer array arr[]
Initialize n as size of array
Input the elements of the array
d=a[n-1]-a[0]/n; b[0]=a[0] ; for 1 to n b[i]=b[i-1]+d
ans=binarysearch(a, b, n)
print ans
}
}

IV. ALGORITHM ANALYSIS

For the above approach based on divide and conquer we are effectively dividing the array in the array into 2 halves and taking one of the two halves which is further divided into two halves, until the missing element is found.

Calculating time complexity: Assume that k (the function missing Term) is called k times.

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

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

- And according to the required condition one of the two halves is taken into consideration.

- After the 2nd function call, the halved array from the previous step is divided again into two parts and length of array becomes n/4.

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

- Considering the same scenario 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

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, the best case arises. As we've found the required element in the 1st call so, there are no function calls involved and hence the time complexity would be O(1).

TABLE 1

TIME COMPLEXITY OF BINARY SEARCH APPROACH

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

The space complexity of the algorithm will be O(n) because we're allocating the memory to a new array of the same size of the given array. So, extra space needed is O(n).

Space Complexity: O(n)

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

[4] https://www.geeksforgeeks.org/find-missing-number-arithmetic-progression/