Design and Analysis of Algorithms

Assignment No. 1

Group-2 **Section-C**

Group members:

IIB2019003 Vasu Gupta

IIB2019004 Saloni Singla

IIB2019005 Sandeep Kumar

Problem:

Design an algorithm to find the element that would be at the k'th position of the final sorted array after sorting two arrays of size m and n.

(www.cse.in)Taken from reference [4]

Algorithm-1

- In this divide and conquer approach we will divide both the arrays arr1[] and arr2[] by half of the value of k and then recurse the function.
- If arr1[k/2] is less than arr2[k/2] then the required kth element will not be after arr2[k/2]. So, we will set arr2[k/2] as the last element of the arr2[].
- Else arr1[k/2] is greater than arr2[k/2] then the required kth element will not be after arr1[k/2]. So, we will set arr1[k/2] as the last element of the arr1[].
- Therefore, we will define new sub-problems with k/2 size of one of the arrays.

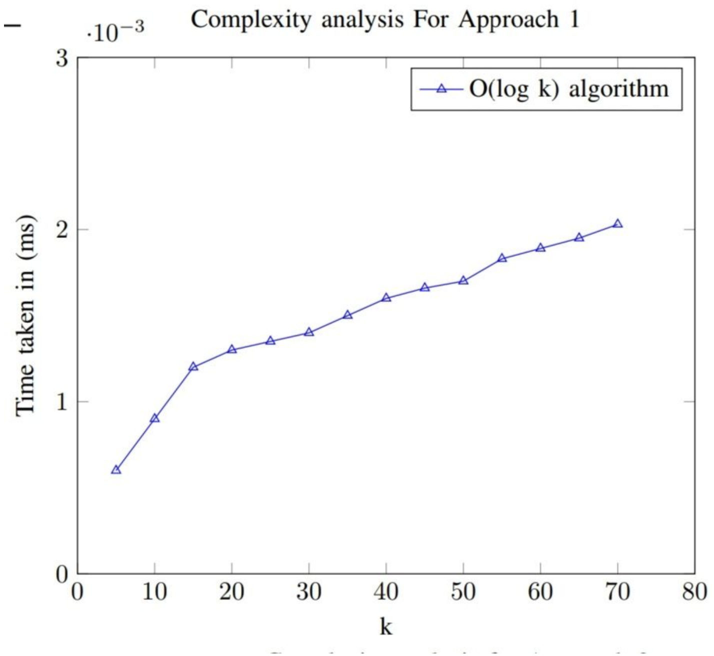
Pseudo Code :

```
kth(arr1,arr2,m,k)
    if k>m+n or k<1 then
    return -1
    if n==0 and m>0
    return arr2[k-n-1]
    if m==0 and n>0
    return arr1[k-m-1]
    if k=1 then
    return minimum(arr1[0],arr2[0])
    i=minimum(n,k/2)
    j=minimum(m,k/2)
    if arr1[i-1]<arr2[j-1] then
    return kth(arr1+i,arr2,(n-i),m,(k-i))
    else
    return kth(arr1,arr2+j,n,(m-j),(k-j))
```

Time Complexity and Space Complexity Analysis of Algorithm 1

Time Complexity Analysis
In this recursive divide and conquer approach, the function k this called a total of logm+logn times. Thus the time complexity of this approach would beO(logk). The best case complexity will be when either of m and n is zero or k is invalid or k is equal to 1. Thus, the best case time complexity isΩ(1)

Space Complexity Analysis
This algorithm has a space complexity of O(log k)



Algorithm 2

- Compare the middle elements of both the arrays that are arr1 and arr2, and define these indices as mid1 and mid2 respectively.

- If arr1[mid1]<arr2[k], then the elements after mid2 cannot have the required element.
- Set the last element of arr2 to be arr2[mid2].
- Similarly, define a new subproblem with half the size of one of the arrays.

Pseudo Code :

```
kth(array1,array2,end1,end2,k-1)
    if array1 = end1 then
        return array2[k]
    if array2 = end2 then
        return array1[k]
      mid1 = (end1 - array1) / 2
      mid2 = (end2 - array2) / 2
    if mid1 + mid2 < k then
        if array1[mid1] > array2[mid2] then
            return kth(array1, array2 + mid2 + 1, end1, end2, k - mid2 - 1)
        else
            return kth(array1 + mid1 + 1, array2, end1, end2, k - mid1 - 1)
  else
        if array1[mid1] > array2[mid2] then
            return kth(array1, array2, array1 + mid1, end2, k)
        else
            return kth(array1, array2, end1, array2 + mid2, k)
```

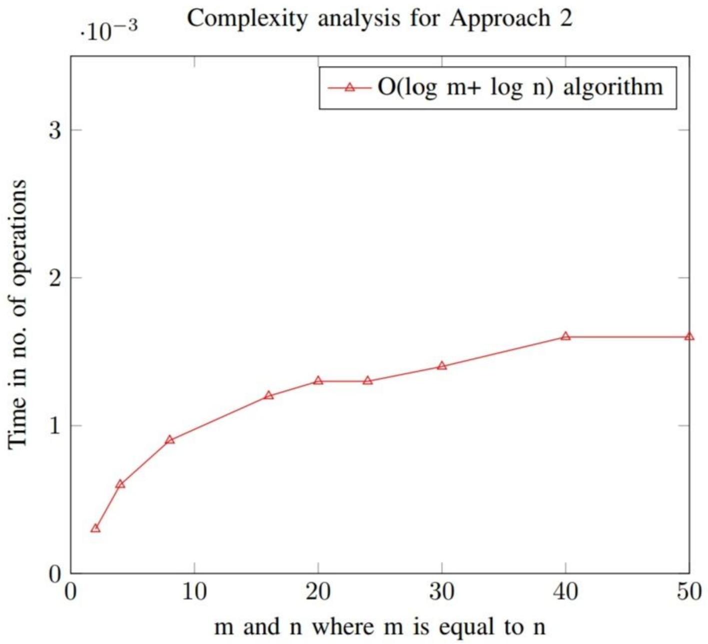
Time Complexity and Space Complexity Analysis of Algorithm 2

Time Complexity Analysis

In this recursive divide and conquer approach, the function kth is called log k times. Thus the time complexity of this approach would beO(logm+ logn).The best case complexity will be when either of m and n is zero or k is invalid or k is equal to 1. Thus, the best case timecomplexity isΩ(1)

Space Complexity Analysis

This algorithm has a space complexity of O(log m+log n)



Conclusion

Above two methods have different time and space complexities and meet to fulfill the problem statement.

The order in which they are good can be listed as:

I. Approach 1

II. Approach 2

Based on the time complexities and space complexities.

References:

1. https://www.geeksforgeeks.org/k−th−element−two−sorted−arrays/
2. https://tutorialspoint.dev/algorithm/divide-and-conquer/k-th-element-two-sorted-arrays