

Theory Assignment-1: ADA Winter-2024

Vikranth Udandaraao (2022570)

Ansh Varshney (2022083)

1 Preprocessing

In this problem, no specific preprocessing steps are required. We assume or have been given the following:

1. A, B, C are sorted arrays.
2. k lies between 1 and $3n$.
3. Indices for all arrays start from 0 and end at $n - 1$.
4. Comparing two elements can be done in $O(1)$ time.
5. The same numbers are considered unique in different arrays, i.e if the numbers in the union are 1,2,2,3 and we have to find 3rd smallest element, then it will be 2.

2 Algorithm Description

The algorithm uses a nested binary search approach to find the k -th smallest element in the union of three sorted arrays A, B , and C . In the `KTH SMALLEST ELEMENT` function, we first find the maximum (called high) and minimum (called low) element of the Union Array (say $A' = A \cup B \cup C$) by comparing $A[0], B[0], C[0]$, and $A[n - 1], B[n - 1], C[n - 1]$ respectively.

We use binary search with mid being the middle of the range $[low, high]$. Now, the helper function `RECURSIVE-BINARYSEARCHCOUNT` is called for each array separately.

`RECURSIVEBINARYSEARCHCOUNT` is a recursive binary function that finds the number of elements that appear before mid in the array. We find this value for each function and then add them to evaluate the variable *totalCount* in `KTH SMALLEST ELEMENT` function.

If *totalCount* is less than k , then our answer would lie in the half right of mid, so the algorithm shifts *low* to *mid* + 1. If *totalCount* is greater than or equal to k , our answer would lie in the half left of mid, so *high* shifts to *mid* (as the answer can be equal to mid also). We return *low* after finishing this search, which is precisely the k -th smallest element in A' .

3 Recurrence Relation

The algorithm used is not recursive in nature and so, there will not be any recurrence relation.

4 Complexity Analysis

The time complexity of the algorithm is $O(\log^2 n)$, where n is the size of the input arrays A, B , and C . We can derive this in the following way:

1. We used binary search in Kth SMALLEST ELEMENT FUNCTION. The range will be $3n$ elements for this binary search as it covers all the elements of all the arrays.
2. For each iteration in the Binary Search, we use three different binary searches for each array, i.e. for n elements.

3. We know that any binary search (with m elements) has a time complexity of $O(\log m)$ as the algorithm keeps on reducing the array size by half in each go.
4. So, the overall algorithm will have the complexity of $O(\log(3n) \cdot 3 \log(n))$ which reduces to $O(\log^2 n)$.

The $O(\log^2 n)$ time complexity implies that the algorithm efficiently finds the k -th smallest element with significantly reduced computational effort compared to linear search methods.

5 Pseudocode

Algorithm 1 Kth Smallest Element

```

1: function RECURSIVEBINARYSEARCHCOUNT(arr, target, low, high)
2:   if low > high then
3:     return 0
4:   end if
5:   mid  $\leftarrow$  low + (high - low)/2
6:   if arr[mid]  $\leq$  target then
7:     return mid - low + 1 + RECURSIVEBINARYSEARCHCOUNT(arr, target, mid + 1, high)
8:   else
9:     return RECURSIVEBINARYSEARCHCOUNT(arr, target, low, mid - 1)
10:  end if
11: end function

12: function KTHSMALLESTELEMENT(A, B, C, k, 3n)
13:   low  $\leftarrow$  min(A[0], B[0], C[0])
14:   high  $\leftarrow$  max(A[n - 1], B[n - 1], C[n - 1])
15:   while low < high do
16:     mid  $\leftarrow$  low + (high - low)/2
17:     a_count  $\leftarrow$  RECURSIVEBINARYSEARCHCOUNT(A, mid, 0, n - 1)
18:     b_count  $\leftarrow$  RECURSIVEBINARYSEARCHCOUNT(B, mid, 0, n - 1)
19:     c_count  $\leftarrow$  RECURSIVEBINARYSEARCHCOUNT(C, mid, 0, n - 1)
20:     total_count  $\leftarrow$  a_count + b_count + c_count
21:     if total_count < k then
22:       low  $\leftarrow$  mid + 1
23:     else
24:       high  $\leftarrow$  mid
25:     end if
26:   end while
27:   return low
28: end function

```

6 Proof of Correctness

Let's assume that the arrays A , B , and C are sorted in ascending order. Also, let A' be the union of A , B , and C . We want to find the k -th smallest element in A' .

The algorithm uses a binary search approach to efficiently narrow down the search space. The variable low represents the minimum possible value for the k -th smallest element, and $high$ represents the maximum possible value.

In each iteration of the while loop, the algorithm calculates the middle value mid between low and $high$ and counts the number of elements less than or equal to mid in each of the arrays A , B , and C using the RECURSIVEBINARYSEARCHCOUNT function.

The total count is then calculated as the sum of counts from all three arrays. If the total count is less than k , it means that the k -th smallest element must be in the right half of the current search space. Therefore, the algorithm updates low to $mid + 1$. If the total count is greater than or equal to k , the k -th smallest element must be in the left half of the current search space, and $high$ is updated to mid .

The algorithm continues this process until low and $high$ converge, at which point low represents the k -th smallest element.

Now, let's consider the correctness of the algorithm:

1. Initialization: - At the beginning of the algorithm, low is set to the minimum element in A' , and $high$ is set to the maximum element in A' . This ensures that the search space is initialized correctly.
2. Maintenance: - In each iteration of the while loop, the search space is narrowed down by updating either low or $high$ based on the total count of elements less than or equal to mid . The algorithm correctly maintains the search space to eventually converge to the k -th smallest element.
3. Termination: - The while loop terminates when low and $high$ converge. At this point, low represents the k -th smallest element in A' . The algorithm correctly identifies the k -th smallest element within the sorted union of arrays.

Therefore, the algorithm is correct and efficiently finds the k -th smallest element in the union of three sorted arrays.

DEMO TESTCASE 1: $A=[2,3,6]$ $B=[1,5,8]$ $C=[4,7,9]$; $n=3$; $k=2$

OUTPUT: 2

EXPLANATION: The first binary search runs in the range $[1,9]$. We keep on finding the values for `totalCount` till we reach the desired k value. We will return `low` which will be equal to 2.