Theory Assignment-1: ADA Winter-2024

Palak Bhardwaj(2022344) Yashovardhan Singhal(2022591)

January 28, 2024

1 Preprocessing

Preprocessing is not applicable.

2 Algorithm Description

Input

1. Three sorted arrays arr1, arr2, arr3 of same length n.
2. The length n.
3. An integer k representing the position of the desired element.

Algorithm

1. Initialize start as the minimum value among the 0th index (first) elements of each of the three arrays and end as the maximum value among the last index elements of each of the three arrays.
2. Initialize a while loop that iterates while start is less than or equal to end.
3. Inside the loop, set mid as the midpoint of start and end, calculated as start + (end-start)/2.
4. For each array, store the count of elements less than or equal to mid in a separate variable. This count is obtained by invoking a helper function, which performs a binary search on the array.
5. The helper function takes the array, its length, and the mid calculated as input. Within the helper function, a new start variable is initialized to 0 and end variable is initialized to the size of the array minus 1.
6. In the helper function, initiate a while loop that iterates while start is less than or equal to end.
7. Inside the loop, initialize a new mid as the midpoint of start and end which were initialised in this helper function. If the element at the mid index of the array is less than or equal to the target(passed as mid from the main function), set start as mid + 1, else set end as mid - 1 (update the mid initialised in the helper function).
8. After the while loop in the helper function concludes, return the value of start, representing the count of elements less than or equal to the target in the array.
9. Repeat steps 4-8 for each of the three arrays, obtaining three separate counts for each array.
10. Compute the total count (variable name totalCount) as the sum of the counts obtained from the three arrays calculated via the helper function above.
11. If totalCount is less than k, update low to mid + 1, in order to search for a higher answer(go right in the search space).
12. Otherwise, update high to mid - 1, in order to search for a lower answer(go left in the search space).
13. After the while loop breaks, the start value is returned, representing the kth smallest element.

3 Recurrence Relation

Recurrence relation is not applicable since it is an iterative code.

4 Complexity Analysis

- **Binary Search:**
  - `countLessEqual` performs binary search on a sorted array with a time complexity of $O(\log n)$.
  - `kthelement` uses three binary searches iteratively for each array.
- **Overall Time Complexity:**
  - The main loop in `kthelement` has a combined time complexity of $O(\log n \cdot \log n)$.
  - This complexity arises from the nested nature of binary searches on arrays of size n.

5 Pseudocode

Algorithm 1 Finding the k-th Smallest Element in Union of Three Sorted Arrays

```
function CountLessEqual($arr, len, target$)
    $start \leftarrow 0$
    $end \leftarrow len - 1$
    while $start \leq end$ do
        $mid \leftarrow start + \frac{(end-start)}{2}$
        if $arr[mid] \leq target$ then
            $start \leftarrow mid + 1$
        else
            $end \leftarrow mid - 1$
        end if
    end while
    return $start$
end function
function KthSmallest($arr1, arr2, arr3, n, k$)
    $start \leftarrow \min(arr1[0], \min(arr2[0], arr3[0]))$
    $end \leftarrow \max(arr1[n-1], \max(arr2[n-1], arr3[n-1]))$
    while $start \leq end$ do
        $mid \leftarrow start + \frac{(end-start)}{2}$
        $count1 \leftarrow$ CountLessEqual($arr1, n, mid$)
        $count2 \leftarrow$ CountLessEqual($arr2, n, mid$)
        $count3 \leftarrow$ CountLessEqual($arr3, n, mid$)
        $totalCount \leftarrow count1 + count2 + count3$
        if $totalCount < k$ then
            $start \leftarrow mid + 1$
        else
            $end \leftarrow mid - 1$
        end if
    end while
    return $start$
end function
```

6 Proof of Correctness

To prove the correctness of the given algorithm, we can use a proof by **contradiction**. The algorithm aims to find the k-th smallest element in the union of three sorted arrays. We can show that if the algorithm produces a result, that result is indeed the k-th smallest element, i.e. the result would not be greater or smaller than k.

Claim: If the algorithm returns a value result, then result is the k-th smallest element in the union of the three sorted arrays. Proof:

Assumption: Assume that the algorithm returns a value as result, but result is not the k-th smallest element.

Contradiction: We will show that this assumption leads to a contradiction.

1. **Definition of result:** The algorithm terminates when start is no longer less than or equal to end. At this point, start represents the k-th smallest element in the union of three sorted arrays.
2. **Assumption Contradicts the Definition:** If result is not the k-th smallest element, it means that either result is smaller than the k-th smallest element or greater than the k-th smallest element.

   This would lead to two cases -

   (a) **Case 1:** result < k-th smallest element: If result is smaller than the k-th smallest element, it contradicts the fact that start is the k-th smallest element. This is because the algorithm updates start to be mid + 1 when totalCount < k.

   (b) **Case 2:** result > k-th smallest element: If result is greater than the k-th smallest element, it contradicts the fact that end is the k-th smallest element. This is because the algorithm updates end to be mid - 1 when totalCount >= k.
3. **Conclusion:** In either case, our assumption that result is not the k-th smallest element leads to a contradiction. Therefore, the algorithm must correctly return the k-th smallest element in the union of three sorted arrays.

Another **general way** of looking at this can be -
The initial search space includes the entire union of three arrays, and the algorithm correctly initializes the pointers.
At each iteration, the algorithm refines the search space based on the count of elements less than or equal to the mid value. It is ensured that the k-th smallest element is always within the search space.
The algorithm terminates when the search space is reduced to a single candidate element, and this element is guaranteed to be the k-th smallest element.

7 Assumptions

1. Indexing is 0th based, first element is represented by 0th index.
2. There are at least k elements in the union of the three arrays.
3. The input arrays are sorted.
4. The input arrays are not empty.
5. The values in the arrays are integers.
6. The elements in the arrays are distinct, the duplicates are treated as distinct elements.

8 Resources

Link1 Link2