The Name of Task : K-th Element of Two Sorted Arrays

Project Idea : 5

Team Number : 48

Name	ID
محمد ضيف الله جابر	20210797
محمد ممدوح محمد عبد المعطي	20210839
عبدالرحمن صلاح محمد فكري	20210512
عبدالرحمن صفوت عبد القادر محمد	20210510
عمر احمد صبري محمد	20210592
عمر احمد امام حنفي	20210591

Merge Sort Alogrithm Psuedocode & Time Complexity

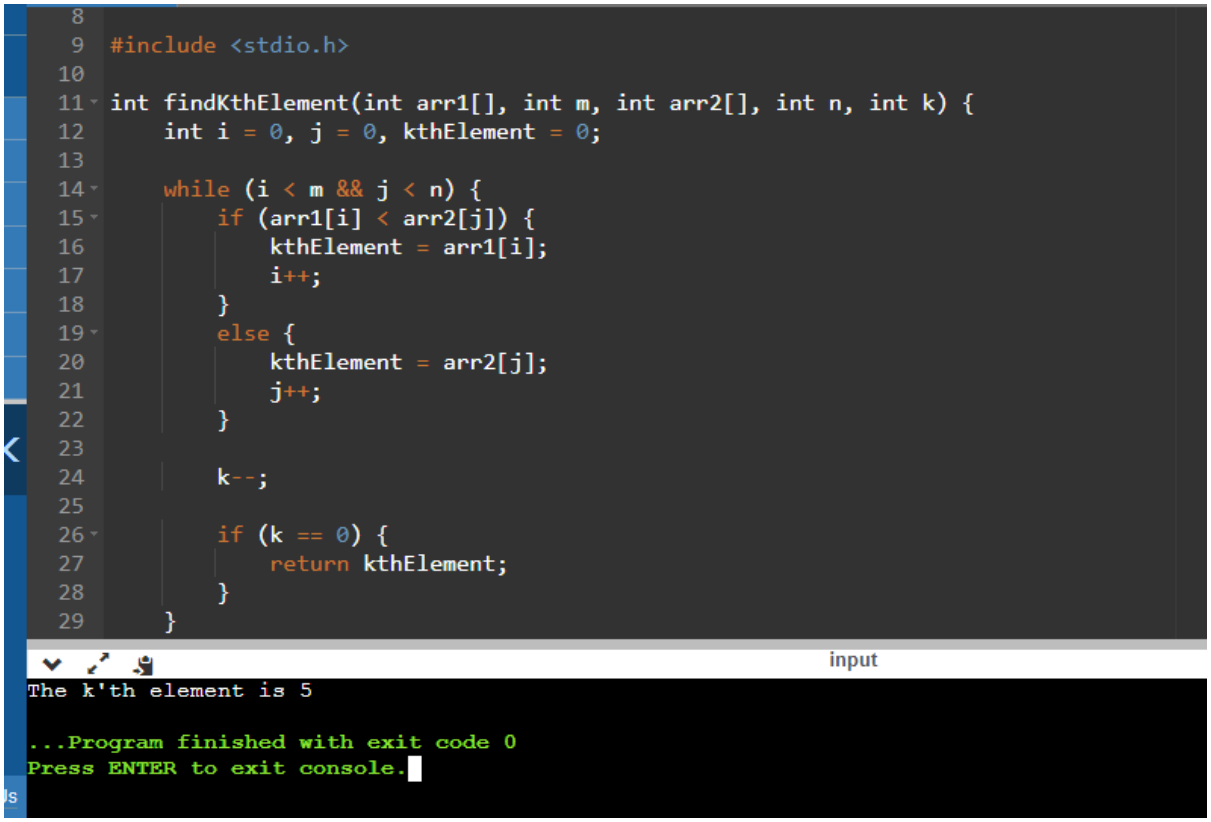
```
function find_kth_element(arr1, arr2, m, n, k):
    i = 0
    j = 0
    count = 0
    while i < m and j < n:
        if arr1[i] < arr2[j]:
            count += 1
            if count == k:
                return arr1[i]
            i += 1
        else:
            count += 1
            if count == k:
                return arr2[j]
            j += 1
    while i < m:
        count += 1
        if count == k:
            return arr1[i]
        i += 1
    while j < n:
        count += 1
        if count == k:
            return arr2[j]
        j += 1
```

Time Complexity With Documention :

1) The first step in this algorithm is to merge two sorted arrays into a single sorted array, which takes O(m + n) time.

2) The second step is to find the kth element in the merged array, which takes O(1) time.

3) Therefore, the time complexity of this algorithm is O(m + n).



```c

#include <stdio.h>

int findKthElement(int arr1[], int m, int arr2[], int n, int k) {
    int i = 0, j = 0, kthElement = 0;

    while (i < m && j < n) {
        if (arr1[i] < arr2[j]) {
            kthElement = arr1[i];
            i++;
        }
        else {
            kthElement = arr2[j];
            j++;
        }

        k--;

        if (k == 0) {
            return kthElement;
        }
    }
```

input

```
The k'th element is 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Binary Search Sort Alogrithm Psuedocode & Time Complexity

```
function findKthElement(arr1, arr2, k):
    if arr1.length == 0:
        return arr2[k]
    if arr2.length == 0:
        return arr1[k]
        if k == 0:
        return min(arr1[0], arr2[0])
    mid1 = (arr1.length - 1) / 2
    mid2 = (arr2.length - 1) / 2
    if (mid1 + mid2) < k:
        if arr1[mid1] > arr2[mid2]:
            return findKthElement(arr1, arr2[mid2+1:], k - (mid2+1))
        else:
            return findKthElement(arr1[mid1+1:], arr2, k - (mid1+1))
    else:
        if arr1[mid1] > arr2[mid2]:
            return findKthElement(arr1[:mid1], arr2, k)
        else:
            return findKthElement(arr1, arr2[:mid2], k)
```

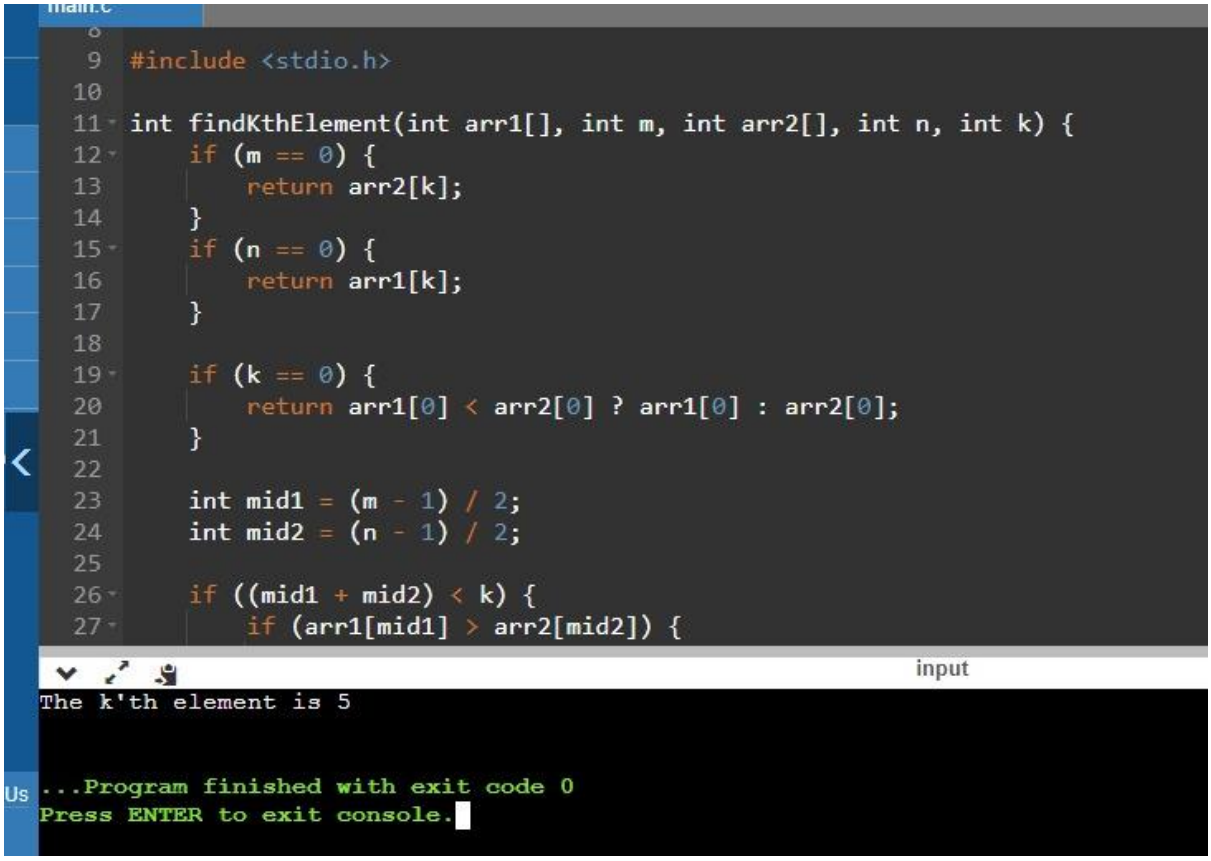
Time Complexity With Documention :

1) In the worst case, the algorithm will have to perform binary search on both arrays, which takes O(log m) + O(log n) time.

2) The algorithm also needs to count the number of elements from 0 to mid in both arrays for each iteration, which takes O(1) time.

3)Therefore, the time complexity of this algorithm is O(log m + log n).

.


main.c
#include <stdio.h>
int findKthElement(int arr1[], int m, int arr2[], int n, int k) {
if (m == 0) {
return arr2[k];
}
if (n == 0) {
return arr1[k];
}
if (k == 0) {
return arr1[0] < arr2[0] ? arr1[0] : arr2[0];
}
int mid1 = (m - 1) / 2;
int mid2 = (n - 1) / 2;
if ((mid1 + mid2) < k) {
if (arr1[mid1] > arr2[mid2]) {
input
The k'th element is 5
...Program finished with exit code 0
Press ENTER to exit console.

Comparison and Efficiency

Alogrithm	Merge Sort	Binary Search Sort
Time Complexity	O(m + n)	O(log m + log n)

The binary search algorithm is more efficient than the merge and sort algorithm, because it has a lower time complexity of O(log m + log n) compared to O(m + n). The binary search algorithm avoids the expensive operation of merging the two arrays, but instead relies on binary search to locate the kth element.

In terms of space complexity, both algorithms require O(m + n) extra space to merge the arrays in the merge and sort algorithm, and O(1) extra space for the binary search algorithm, since it doesn't require any additional memory.

Overall, the binary search algorithm is a more efficient solution for finding the kth element in two sorted arrays