ALGORITHMS TASK

SPRING 2023

TASK NUMBER 5

K-TH ELEMENT OF TWO SORTED ARRAYS

NAMES

1. محي الدين ماهر	20210887
2. محمود عبد الدايم أبو المجد	20210867
3. مروة عمر محمد محمود	20210900
4. مريم محمود سيد محمود	20210926
5. محمود حسام الدين محمود	20210859
6. محمد مصطفى محمد عبد ربه	20210836

MERGE SORT APPROACH

A. PSEUDOCODE

```
mergeSort( arr1 , size1 ,arr2 ,size2 , finalArr ,i , j, k) :

 if i<size1 && j<size2
   if arr1[i]<arr2[j]
    finalArr[k] = arr1[i]
    i++

   else
     finalArr[k] = arr2[j]
     j++

   k++
   mergeSort( arr1 , size1 ,arr2 ,size2 , finalArr ,i , j, k)

 else
    while i<size1
      finalArr[k] = arr1[i]
      i++
      k++

   while j<size2
      finalArr[k] = arr2[ij]
      j++
      k++

end
```

B. SOURCE CODE

- [GITHUB LINK]

C. CODE ANALYSIS

Step 1)recursive call->O(s1 + s2)
Step 2)comparison arr1[i] and arr2[j] -> O(1)
Step 3) We increment i or j and k->O(1) time.
Step 4) recursive-> O(s1 + s2)
Step 5) After one array is traversed, we copy the remaining elements of the other array into finalarr.
This takes O(s1) or O(s2) time depending on which array remains.
the total time complexity is O(s1 + s2) + O(s1 + s2) = O(s1 + s2)
as s2>s1 : time complexity = O(s2) = O(n)

D. TIME COMPLEXITY:

O(n)

E.OUTPUT SCREENSHOT

```
int main()
 {
 int ar1[]={100,112,256,349,770};
 int ar2[]={72,86,119,265,445,892};
 int s1 = sizeof(ar1)/sizeof(ar1[0]);
 int s2 = sizeof(ar2)/sizeof(ar2[0]);
 int finalar[s1+s2];

 int k = 7;
 mergeSort(ar1,s1,ar2,s2,finalar,0,0,0);

 cout<<finalar[k-1]; // because the array is 0 indexe
return 0;
  }
```

```
E:\testingggg_c\Algorithm_Task\bin\Debug\Algorithm_Task.exe
265
Process returned 0 (0x0)   execution time : 0.139 s
Press any key to continue.
```

NON-RECURSIVE APPROAC

A. PSEUDOCODE

```
nonRecursive( arr1 , size1 ,arr2 ,size2 , finalArr) :

int i=0,j=0,k=0
 while i<size1 && j<size2
    if arr1[i]<arr2[j]
     finalArr[k] = arr1[i]
     i++
     k++

    else
      finalArr[k] = arr2[j]
      j++
      k++


  while i<size1
     finalArr[k] = arr1[i]
     i++
     k++

  while j<size2
     finalArr[k] = arr2[ij]
     j++
     k++

end
```

B. SOURCE CODE

- GITHUB LINK

C. CODE ANALYSIS

Step 1)while loop O(s1 + s2)
Step 2)comparison arr1[i] and arr2[j] -> O(1)
Step 3) We increment i or j and k->O(1) time.
Step 5) After one array is traversed, we copy the remaining elements of the other array into finalarr
This takes O(s1) or O(s2) time depending on which array remains.
the total time complexity is O(s1 + s2) + O(s1 + s2) = O(s1 + s2)

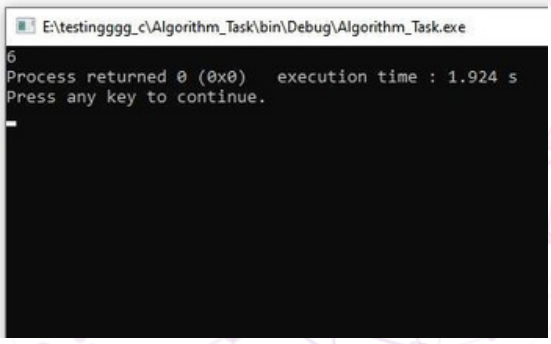
D. TIME COMPLEXITY:

O(n)

E.OUTPUT SCREENSHOT

```
int main()
 {
fast();
int a[] = {2,3,6,7,9};
int b[] = {1,4,8,10};
int s1 = sizeof(a)/sizeof(a[0]);
int s2 = sizeof(b)/sizeof(b[0]);
int f[s1+s2];
nonRecusize(a,s1,b,s2,f);

int k=5;
cout<<f[k-1];
return 0;
  }
```



DIVIDE AND CONQUER APPROACH

A. PSEUDOCODE

```
Algorithm kth_element(arr1,arr2,last1,last2, k)

  if arr1 == last1
    return arr2[k]
  if arr2 == last2
    return arr1[k]

     mid1 ← (last1 - arr1)/2
   mid2 ← (last2 - arr2)/2

  if mid1 + mid2 < k
    if arr1[mid1] > arr2[mid2]
      return kth_element(arr1, arr2+mid2+1, last1, last2, k-mid2-1)
    else
      return kth_element(arr1+mid1+1, arr2, last1, last2, k-mid1-1)
  else
    if arr1[mid1] > arr2[mid2]
      return kth_element(arr1, arr2, arr1 + mid1, last2, k)
    else
        return kth_element(arr1, arr2, last1 ,arr2 + mid2, k)
  end
```


B. SOURCE CODE

- GITHUB LINK

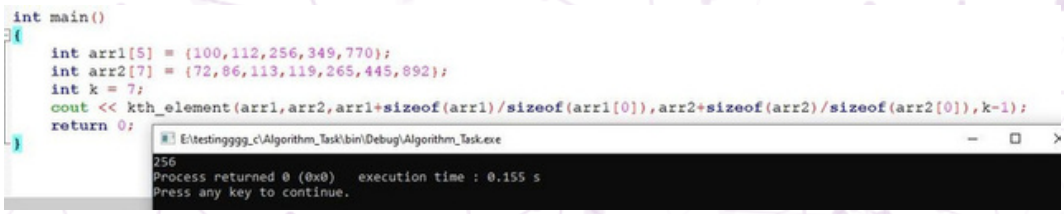
C. CODE ANALYSIS

The kth_element function takes 5 parameters: two sorted arrays (arr1 and arr2) , two pointers to the last element of each array (last1 and last2), and an int k representing the index of the desired element. If the length of one of the arrays is 0, then the function returns the k-th element of the other array. Two variables, mid1 and mid2, are set to the middle index of each array. If the sum of mid1 and mid2 is less than k, the function recursively calls itself with either arr2+mid2+1 (if arr1[mid1] is greater than arr2[mid2]) or arr1+ mid1+1 (if arr1[mid1] is less than or equal to arr2[mid2]), and an updated value of k. If k is less than the sum of mid1 and mid2, the function recursively calls itself with either arr1+mid1 and last2 (if arr1[mid1] is greater than arr2[mid2]) or last1 and arr2+mid2 (if arr1[mid1] is less than or equal to arr2[mid2]). The main function initializes two sorted arrays (arr1 and arr2) and an integer k. It then calls the kth_element function with the two arrays, their last elements, and k-1 (since arrays are 0-indexed). Finally, the kth_element function returns the kth element of the combined arrays.

D. TIME COMPLEXITY:

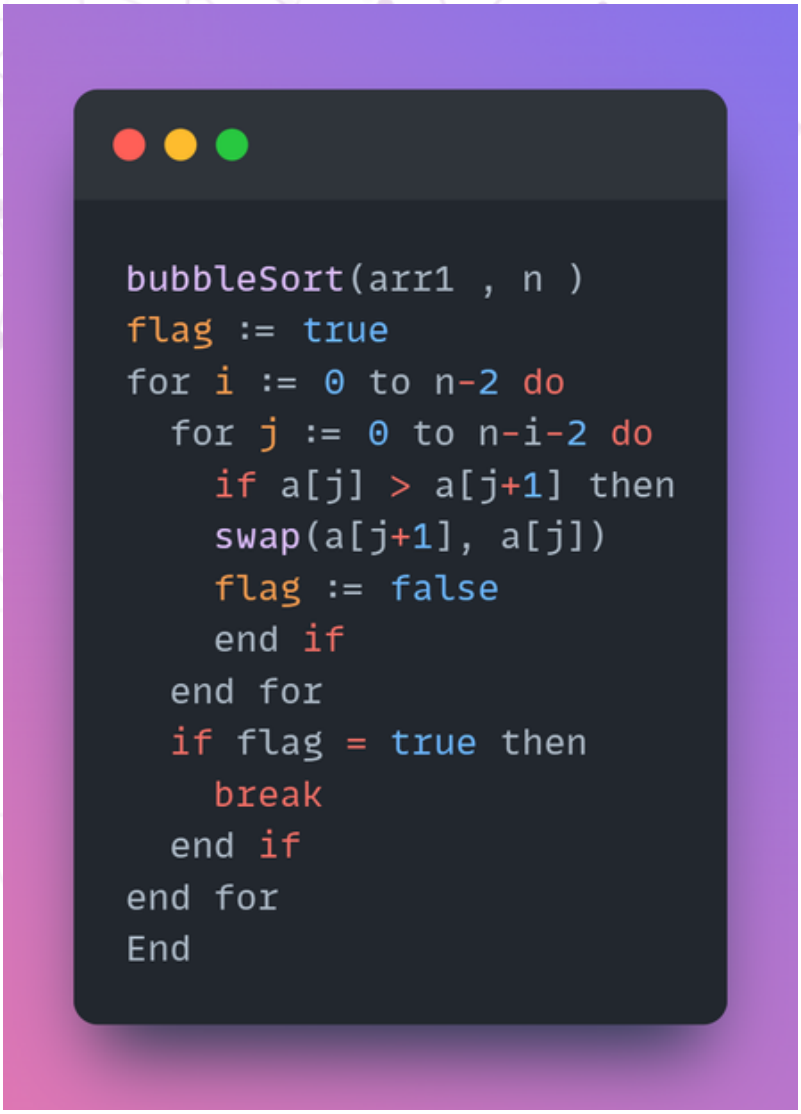
O(log m + log n)

E.OUTPUT SCREENSHOT



BUBBLE SORT APPROACH

A. PSEUDOCODE



```
bubbleSort(arr1 , n )
flag := true
for i := 0 to n-2 do
  for j := 0 to n-i-2 do
    if a[j] > a[j+1] then
    swap(a[j+1], a[j])
    flag := false
    end if
  end for
  if flag = true then
    break
  end if
end for
End
```

B. SOURCE CODE

- GITHUB LINK

C. CODE ANALYSIS

At pass 1: Number of comparisons = (n - 1) Number of swaps = (n - 1)
At pass 2: Number of comparisons = (n - 2) Number of swaps = (n - 2)
At pass 3: Number of comparisons = (n - 3) Number of swaps = (n - 3)
At pass n-2: Number of comparisons = 2 Number of swaps = 2
At pass n - 1 : Number of comparisons = 1 Number of swaps = 1 Total number of comparison required: (n - 1) + (n - 2) + (n - 3) +...2+1=n(n-1)/2 by using sum of N natural Number formula Basic Operation: swap(a[j+1],a[j])

D. TIME COMPLEXITY:

$O(n^2)$

E.OUTPUT SCREENSHOT

```
int main()
{
  int a[]={2,3,6,7,9};
  int b[]={1,4,8,10};
  int k=5;
  int s1 = sizeof(a)/sizeof(a[0]);
  int s2 = sizeof(b)/sizeof(b[0]);
  ll tot = s1 + s2; //the size of the final array
  int finalarray[tot];
  for(int i=0;i<s1;i++){
    finalarray[i]=a[i];
  }
  for(int i=s1;i<tot;i++){
    finalarray[i]=b[i-s1];
  }
  bubbleSort(finalarray,tot);
  cout<<finalarray[k-1] ; //because the array is 0 indexed
return 0;
  }
```

```
E:\testingggg_c\Algorithm_Task\bin\Debug\Algorithm_Task.exe
6
Process returned 0 (0x0)   execution time : 1.620 s
Press any key to continue.
```

INSERTION SORT APPROACH

A. PSEUDOCODE

```
insertionSort(ar , n)
  for i := 1 to n-1 do
    key := ar[i]
    j := i-1
    while j ⩾ 0 and ar[j] > key do
      ar[j+1] := ar[j]
      j := j-1
    end while
    ar[j+1] := key
  end for
End
```

B. SOURCE CODE

- GITHUB LINK

C. CODE ANALYSIS

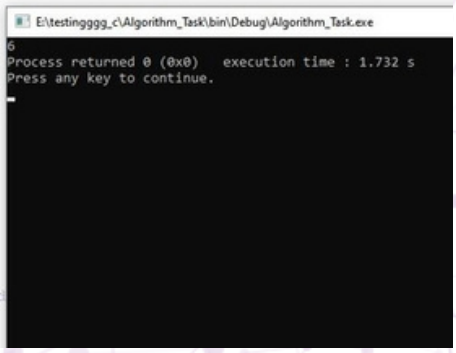
Step 1) for i = 1 to n-1 n
Step 2) key := A[i]
Step 3) n-1
Step 4) ji-1
Step 5) n-1
Step 6) while j >= 0 and A[j] > key –
Step 7) A[j + 1] = A[j]
Step 8) Ση - 1,j
Step 9) Ση - 1,j
Step 10) j = j - 1
Step 11) Ση - 1,j
Step 12) A[j + 1] = key
Step 13) n-1

D. TIME COMPLEXITY:

$O(n^2)$

E.OUTPUT SCREENSHOT

```
int main()
 {
 int a[]={2,3,6,7,9};
 int b[]={1,4,8,10};
 int k=5;
 int s1 = sizeof(a)/sizeof(a[0]);
 int s2 = sizeof(b)/sizeof(b[0]);
 ll tot = s1 + s2; // the tot size of the final array
 int finalarray[tot];
 for(int i=0;i<s1;i++){
   finalarray[i]=a[i];
 }
 for(int i=s1;i<tot;i++){
   finalarray[i]=b[i-s1];
 }
 insertionSort(finalarray,tot);
 cout<<finalarray[k-1] ; // because the array is 0 indexed
return 0;
  }
```



Comparing between approaches from the complexity

Divide and Conquer	Non-Recursive	Merge Sort	Bubble Sort	Insertion Sort
O(log m + log n)	O(n)	O(n)	$O(N^2)$	$O(N^2)$

The best algorithm used for finding the k-th element of two sorted arrays according to the previosly mentioned algorithms above, is the divide and conquer algorithm which is a recursive algorithm that works with time complexity O(log m+log n) since its complexity is less than both merge's and non recursive's complexity O(n) and also less than bubble sort and insertion sort complexity $O(n^2)$