**Algorithm Merge Sort (Algo 1) [Pseudo Code]**

merge(dest, arr1, sz1, arr2, sz2){

// dest (destination array), arr1 (first array), sz1 (size of arr1), arr2 (second array), sz2 (size of arr2)

J := 0 , k := 0

For i := 0 to sz1 + sz2 - 1 step 1

Do {

If arr1[j] <= arr2[k] then

dest[i] := arr1[j++]

Else

Set dest[i] := arr2[k++]

endif

}

}

find\_kth\_el( arr1, sz1, arr2, sz2, k ){

// arr1 (first array), sz1 (size of arr1), arr2 (second array), sz2 (size of arr2), k (kth element to find)

dest[sz1 + sz2]

merge(dest, arr1, sz1, arr2, sz2)

Return dest[k - 1]

}

**Algorithm Merge Sort (Algo 1) [Time Complexity]**

1. **Merge Function Time Complexity:**

**=**

Sz1 + Sz2 - 1 - 0 + 1 =

Sz1 +Sz2 = **O (Sz1 + Sz2)**

2. **Find\_Kth\_el Function Time Complexity:**

O (Sz1 + Sz2) + O (1) = **O (Sz1 + Sz2)**

**Result Of The Time Complexity of This Algorithm is**:

**O (Sz1 + Sz2) = O (n)**

**Non-recursive algorithms (Algo 2) [Pseudo Code]**

kth(int arr1[], int arr2[], int n, int m, int k)

{

sorted1[m + n]

i := 0, j := 0, d := 0

while i < n && j < m do

{

if arr1[i] < arr2[j] then

sorted1[d++] := arr1[i++]

else

sorted1[d++] := arr2[j++]

}

while i < n do

{ sorted1[d++] := arr1[i++] }

while j < m do

{ sorted1[d++] := arr2[j++] }

return sorted1[k - 1]

}

**Non-recursive algorithms (Algo 2)**   **[Time Complexity]**

* Let T(n, m) be the time complexity function where n is the length of arr1 and m is the length of arr2.
* The total work done W is the sum of the work done in each loop:

W=i=1∑min(n,m)​O(1)+i=1∑n​O(1)+i=1∑m​O(1)

* Simplifying the sums, we get:

W=min(n,m)⋅O(1)+n⋅O(1)+m⋅O(1)

* Since min(n,m) is at most n or m, we can simplify further:

W=n⋅O(1)+m⋅O(1)

* Therefore, the final time complexity is:

T(n,m)=O(n+m)=O(n).