Design and Analysis of Algorithms

Assignment No. 1

Group members:

Harsh Meena -IIT2019005
Asha Jyothi Donga -IIT2019006
Sampada Kathar -IIT2019007

Merge two sorted arrays in optimal way

Problem:

Two sorted arrays are given, which are to be merged in a single sorted array in the best possible optimal way.

Algorithm 1: (Insertion)

1. Create the third array of size of array1 and array2
2. Copy the arr1 elements to the arr3. The last n2 elements of arr3 are empty.
3. Keep track of the ending index of arr3 by storing in variable.
4. Traverse the arr2 and insert each element in arr3 where left side elements are lesser and right are greater than.
5. For each element in arr2,checks from the end of the arr3 and traverses to the front . It will stop when it finds the smallest greater element than the element which is to be inserted.
6. While traversing to the front it moves each element to the next index, so that it can leave space for the element to be inserted.
7. After traversing the whole arr2 stop the insertion.Now, arr3 is a sorted merged array of arr1 and arr2.

Pseudo Code :

```
Array mergeArray(Array arr1, Array arr2)
    n1=size_of_arr1,  n2=size_of_arr2
    Array arr3(n1+n2)
    for(i=0 to n1-1):
       arr3[i]=arr1[i]
    end=n1-1
    for( i=0 to n2-1):
          j=end
          while(j>=0 &&arr2[i]<arr3[j]):
              arr3[j+1]=arr3[j]
               j--
          arr3[j+1]=arr2[i]
          end++
    Return arr3
```

Algorithm 2(Merge Sort Algorithm)

The idea is to traverse arr1 and arr2 simultaneously, using two iterators and compare the element of current position of both arrays and insert the smaller one in the 3rd array.

Steps :

1. Create the third array of size of array1 and array2
2. Copy the arr1 elements to the arr3. The last n2 elements of arr3 are empty.
3. Keep track of the ending index of arr3 by storing in variable.
4. Traverse the arr2 and insert each element in arr3 where left side elements are lesser and right are greater than.
5. For each element in arr2,checks from the end of the arr3 and traverses to the front . It will stop when it finds the smallest greater element than the element which is to be inserted.
6. While traversing to the front it moves each element to the next index, so that it can leave space for the element to be inserted.
7. After traversing the whole arr2 stop the insertion.Now, arr3 is a sorted merged array of arr1 and arr2.

Pseudo Code :

```
mergeArray(Array arr1, Array arr2, Array arr3)
    Int a=0, b=0,c=0
    n1 = size_of_arr1, n2 =size_of_arr2
    while(a<n1 && b<n2)
        if(arr1[a]<=arr2[b])
            arr[c]=arr1[a]
            a++ c++
        else
            arr[c]=arr2[b]
            b++ c++
    while(a<n1)
        arr3[c]=arr1[a]
        a++ c++
    while(b<n2)
        arr3[c]=arr2[b]
        b++ c++
    return
```

Time Complexity and Space Complexity Analysis

ALGORITHM 1:

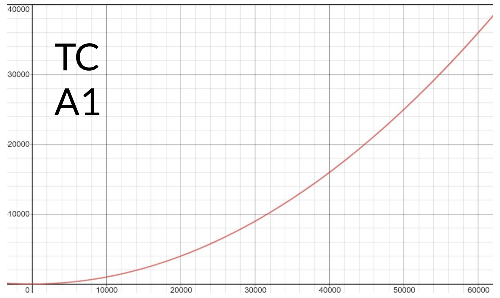
TC (worst case) : O(n1*n2) where n1 and n2 are size of array1 and array2

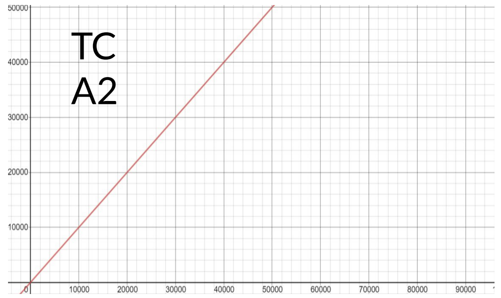
SC (Extra Space): O(n1+n2)

ALGORITHM 2:

TC (worst case) : O(n1+n2) where n1 and n2 are size of array1 and array2

SC (Extra Space): O(n1+n2)





As, we can see that Time complexity for algorithm2 is much optimized than the algorithm1, its a best approch to do it.

Besides these two algorithms there are more 2-3 algorithms to solve this problem, but they take same Time and Space Complexity as of ALGORITHM 2 discussed above:

1) *GAP Method for merging two sorted arrays.*
2) *Using MIN Heap, to merge two sorted arrays in one.*
3) *Using Priority queue to merge two sorted arrays in one.*

All of these methods take same Time Complexity as of ALGORITHM 2 i.e. Merge Sort Method, but these are more complex to implement than those.

So, The best optimal solution for merging two sorted arrays into one big sorted array is by using ***Merge Sort Algorithm(i.e. ALGORITHM 2)***.