Merge two sorted arrays in optimal way

Group Members

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

IV semester - Bachelor's of Technology in Information technology

Indian Institute of Information Technology , Allahabad, India

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

Abstract: In this paper, we have devised 3 algorithms which merge two arrays in an order, while optimizing time and space complexity .

I. INTRODUCTION

Array: An array is simply a data structure that holds multiple instances of a particular type of data, and which is accessed by providing an "index" as to which of the multiple pieces of data you want to read from or write into.

Sorted array: is an array in which all elements are in ascending or descending order. We will sort the given array in this paper.

A. ALGORITHM 1

1. ALGORITHM DESIGN:

Create the third array of size of array1 and array2 Let the size of array1 be 'n1' and of array2 be 'n2'. We create a 3rd array of size 'n1+n2'. n1 and n2 are non-zero.

Copy the arr1 elements to the arr3. The last n2 elements of arr3 are empty.

Store the index upto where the arr3 is filled. Now, arr3 is filled with n1 elements. We will keep track of the ending index of arr3 by storing in variables.

Traverse the arr2 and insert each element in arr3 where left side elements are lesser and right are greater than. insert function inserts the elements in the correct position. Insert function takes 3 arguments.This function 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. 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. After traversing the whole arr2 stop the insertion.Now, arr3 is a sorted merged array of arr1 and arr2.

2. ALGORITHM AND ANALYSIS

Input: Two sorted arrays.(in ascending order) arr1,arr2.

Output: Merged sort array of two given arrays.

n1=size_of_arr1, n2=size_of_arr2

Step 1: Array arr3(n1+n2)
Step 2: for(i=0 to n1-1):
Step3: arr3[i]=arr1[i]

Step 4: insert(Array arr1,Array arr2 ,Array arr3):

Step 5: n1=size_of_arr1,n2=size_of_arr2
Step 6: end=n1-1
Step 7: for(i=0 to n2-1):
Step 8: j=end
Step 9: while(j>=0 && arr2[i]<arr3[j]):
Step 10: arr3[j+1]=arr3[j]
Step 11: j--
Step 12: arr3[j+1]=arr2[i]
Step 13: end++
Step 14: return

Apriori Analysis:

	Time	FreqBest	Freq
Step 1	1	1	1
Step 2	1	2	n1+1
Step 3	1	1	n1
Step 4	1	1	1
Step 5	2	1	1
Step 6	2	1	1
Step 7	2	1	n2
Step 8	1	0	n2-1
Step 9	2	0	(n2-1)* (n1-1)
Step 10	2	0	(n2-1)* (n1-1)
Step 11	1	0	(n2-1)* (n1-1)
Step 12	2	0	n2-1
Step 13	1	1	1
Step 14	1	1	1

$T_\Omega \alpha\ 1*1 + 1*2 + 1*1 + 1*1 + 2*1 + 2*1 + 2*1 + 1*0 + 2*0 + 2*0 + 1*0 + 2*0 + 1*1 + 1*1$

$T_\Omega \alpha\ 13$

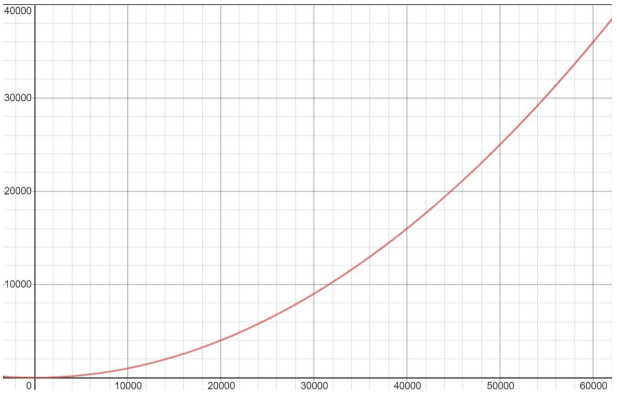
Thus, the best case time complexity is Ω(1) when n1=n2=1.

$T_o \alpha\ 1*1 + 1*(n1+1) + 1*n1 + 1*1 + 2*1 + 2*1 + 2*n2 + 1*(n2-1) + 2*(n2-1)*(n1-1) + 2*(n2-1)*(n1-1) + 1*(n2-1)*(n1-1) + 2*(n2-1) + 1*1 + 1*1$

$T_o \alpha\ 8 + (n1+1) + n1 + 2n2 + 3(n2-1) + 5(n2-1)(n1-1)$

Clearly this equation will come out to be of the quadratic form
Thus, worst case time complexity is O(n1*n2)

The space complexity of this algorithm will be O(n1+n2).



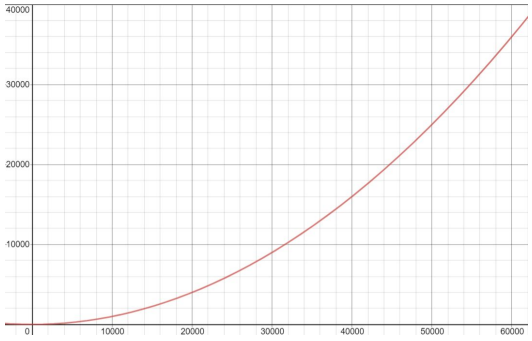
Time complexity graph for apriori analysis

3. PROGRAM RUNNING TIME

Aposteriori Analysis:

n1*n2	Time (in sec)
4	0.0003
8	0.0006

16	0.0012
24	0.0018
48	0.0036



Time complexity graph for aposteriori analysis

The above graph shows that aposteriori analysis is consister with the apriori analysis.

B. Algorithm 2

1. ALGORITHM DESIGN

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.

Stages:

Stage1: We Create the third array of size of array1 and array2. Let the size of array1 be 'n1' and of array2 be 'n2'. We create a 3rd array of size n1+n2, and pass to the function MergeArray

Stage2: 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 and increment the respective iterators. Compare and insert until we reach the end of either arr1 or arr2.

Stage3: Now either arr1 or arr2 is completely traversed. The array which is not completely traversed, will obviously contain the remaining sorted element greater than the elements in sorted array3. So we copy all the elements left in that array to our array3.

Stage4: Finally all the elements of array1 and array2 will be in array3 and sorted in increasing order

We make a MergeArray function, which takes 3 arguments as arrays arr1,arr2,arr3.
mergeArray(Array arr1 ,Array arr2,
array arr3)

2. ALGORITHM AND ANALYSIS:

Algorithm:

Input: Two sorted arrays.(in ascending order)

Output: Merged sort array of two given arrays.

Method:

Step 1: Pass in arr1, arr2 and arr3 to Merge function mergeArray(Array arr1 ,Array arr2, Array arr3)

Step 2: Initialize three iterators 'a', 'b' and 'c'.

Step 3: Take size of arr1 and arr2 in n1 and n2

Step 4: while(a<n1 && b<n2):

Step 5: if(arr1[a]<=arr2[b])
Step 6: arr3[c]=arr1[a]
Step 7: a++; **Step 8:** c++;

Step 9: else
Step 10: arr3[c]=arr2[b]
Step 11: b++; **Step 12:** c++

Either a or b exceeded n1 or n2 respectively. Enter the elements in either arr1 or arr2 after index a or b(either a<n1 or b<n2) into arr3.

Step 13: while (a<n1)

Step 14: arr3[c]=arr1[a]

Step 15: a++

Step 16: c++

Step 17: while (b<n2)

Step 18: arr3[c]=arr1[b]

Step 19: b++

Step 20: c++

As array is by default passed by reference so arr3 will now have all sorted elements from arr1 and arr2.

	Time	FreqBest	Freq
Step 1	1	1	1
Step 2	3	1	1
Step 3	2	1	1
Step 4	2	1	N+1 (where N is the greater number of n1 and n2)
Step 5	1	1	N
Step 6	1	1	N
Step 7	1	1	N
Step 8	1	1	N
Step 9	1	1	N
Step 10	1	1	N
Step 11	1	1	N
Step 12	1	1	N
Step 13	1	1	n1-a+1
Step 14	1	0	n1-a
Step 15	1	0	n1-a
Step 16	1	0	n1-a
Step 17	1	1	n2-b+1
Step 18	1	0	n2-b
Step 19	1	0	n2-b
Step 20	1	0	n2-b

$T_\Omega \alpha$ 1*1 + 3*1 + 2*1 + 2*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 1*0 + 1*0 + 1*0 + 1*1 + 1*0 + 1*0 + 1*0

$T_\Omega \alpha$ 18

Thus, the best case time complexity is Ω(1) when n1=n2=1.

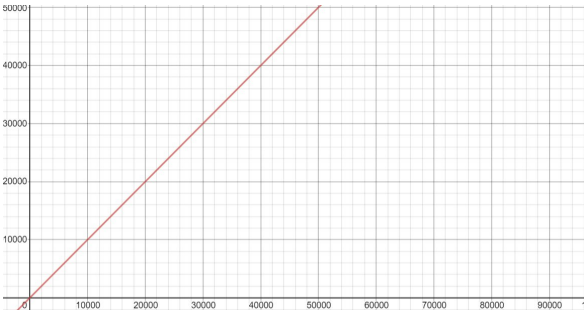
$T_o \alpha$ 1*1 + 3*1 +2*1 + 2*(N+1)+ 8*1*N + 1*(n1-a+1) + 3*1*(n1-a) + 1*(n2-b+1) + 3*1*(n2-b)

$T_o \alpha$ 6 + 2(N+1) + 8N + (n1-a+1) + 3(n1-a) + (n2-b+1) + 3(n2-b)

Clearly this equation will come out to be of the linear form

Thus, **worst case time complexity is O(n1+n2)**, since N will be either n1 or n2 depending on which is larger and either the loop on step 13 or the loop on step 17 won't be executed at all accordingly.

The space complexity of this algorithm will be O(n1+n2).

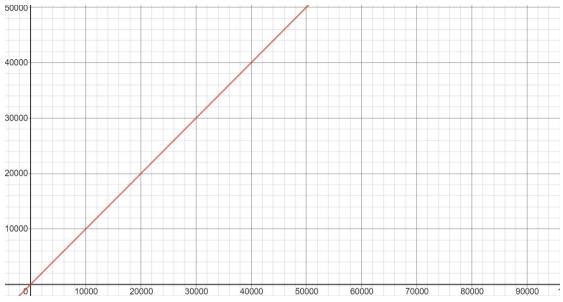


Time complexity graph for apriori analysis

3. PROGRAM RUNNING TIME

Aposteriori Analysis:

n1+n2	Time (in sec)
10	0.0002
20	0.0004
30	0.0007
40	0.0009
50	0.0010



Time complexity graph for aposteriori analysis

The above graph shows that aposteriori analysis is consister with the apriori analysis.

C. Algorithm 3

1. ALGORITHM DESIGN

Here we compare elements in combine of array1 and array2 assuming it as one big array of elements from both arrays, at a distance of a gap.

Gap is calculated as ceil of (n1+n2)/2 and reduces the gap by dividing it by 2 every time.

As, this algorithm take same **Time and Space Complexity as Algorithm II, i.e TC O(n1+n2) and SC(n1+n2)** and is more complex than it, it is not elaborated more.

PROPOSED ALGORITHM

Algorithm II

```
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 and  b < n2 do

        If arr1[a] <=  arr2[b]
            arr3[c] =  arr1[a]
             a++  c++
        else
            arr3[c]  = arr2[b]
             b++  c++

    While a < n1 do

        arr3[c]  = arr1[a]

        a++   c++

    While  b < n2 do

        arr3[c] = arr1[b]

        b++    c++
```

Conclusion

From the above two algorithms we conclude that the ALGORITHM II (Merge Sort Algorithm) takes significantly less time complexity than the former one, while the space complexity (extra space used) will remain the same) for any algorithm. Because for minimum we need to make a third array for putting all the sorted elements from given two sorted arrays.

So, we can conclude that the best possible way to merge two sorted arrays is by using the Merge Sort algorithm, as it is easy and fast to implement. There are two more ways to implement this problem with same time and space complexity, but are complex to implement.

This algorithm takes the minimum time when the value of n1 and n2 are both 1.

References

https://www.geeksforgeeks.org/merge-two-sorted-arrays/

https://www.geeksforgeeks.org/merge-two-sorted-arrays-o1-extra-space/

https://medium.com/swlh/merge-two-sorted-arrays-without-extra-space-efficiently-o-1-gap-method-detailed-simplified-57a336146601

https://www.geeksforgeeks.org/merge-k-sorted-arrays-set-2-different-sized-arrays/

https://www.geeksforgeeks.org/efficiently-merging-two-sorted-arrays-with-o1-extra-space/