

Counting Number of Inversions in an Array

Harshdeep Singh¹ Jaya Ram² Shivansh Gupta³

¹IIT2019105 ²IIT2019106 ³IIT2019107

22 March 2021

- 1 Introduction
- 2 Algorithm Description and Analysis
- 3 Pseudo Code
- 4 Time Complexity
- 5 Space Complexity
- 6 References

Introduction

- The Inversion Count of an array **A** is the count of all the pairs (i, j) such that $i < j$ and $A[i] > A[j]$

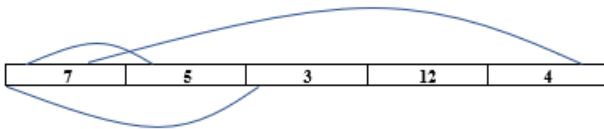


Figure: An example of inversion in array.

- Thus, Inversion Count for an array indicates – how far the array is from being sorted.

Algorithm Description and Analysis

Naive Approach involves a Brute Force approach where we iterate over all the pairs of the array (which can be done through a nested loop), and if any pair (i, j) satisfies our condition, we increase our Inversion Count by one.

Divide and Conquer Approach

Suppose we divide our array A into two equal, or almost equal parts. Let's call the first one L and the other one R . Also, let's say we know the Inversion Count of both L and R . Let's call them inv_1 and inv_2 .

Now, any inversion in A would be of type:

- 1 $i \in L$ and $j \in R$.
- 2 $i \in L$ and $j \in L$.
- 3 $i \in R$ and $j \in R$.

Algorithm - Divide and Conquer

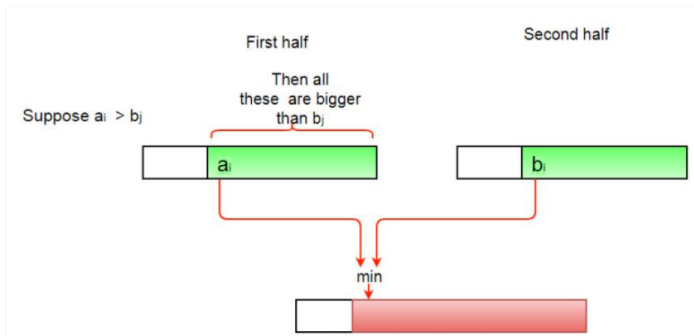
For Case 1

- The total inversions upon combining will be at least $inv_1 + inv_2$. But there may be some more inversions, as we **merge** **L** and **R**. So we need to find for each index $i \in L$, the number of indices $j \in R$ such that $L_i > R_j$.
- Let's say that we have sorted the arrays **L** and **R**. and their sizes are **N** and **M** respectively.

Algorithm - Divide and Conquer

Case 1(contd.)

- Say, at certain point we encounter $L[l] > R[r]$.
Now, because **L** and **R** are sorted, we know that there are going to be **N-i** more inversions in **A**, because all elements to the right of $L[l]$ will also be greater than $R[r]$.



Source - GeeksforGeeks

Algorithm - Divide and Conquer

For Cases 2 and 3

- Now for the second and the third case: If we encounter that situation, we again divide them up into further two arrays and repeat this recursively, until we have arrived on the base case - that includes just one element. And in this case, we know the inversions will be 0, because there is just a single element in the array.

Note

Here, one more important thing is that in the first case, we require the arrays **L** and **R** to be sorted. So we need to incorporate that when we merge two arrays

Pseudo Code

mergeTwoArrays

i, k = beg
j = end
newarr

```
while i < mid AND j ≤ end do  
  if arr[i] < arr[j] then  
    newarr[k] = arr[i]  
    increment k, i  
  else  
    newarr[k] = arr[j]  
    invCount += mid - i  
    increment k, j  
  end if  
end while  
while i < mid do  
  temp[k] = arr[i]  
  increment k, i  
end while  
while j < end do  
  temp[k] = arr[j]  
  increment k, j  
end while  
it = beg  
while it < end do  
  arr[it] = newarr[it]  
  increment it  
end while  
return invCount
```

enhancedMergeSort

arr: The Array
beg: Start iterator
end: End iterator
mid, invCount = 0

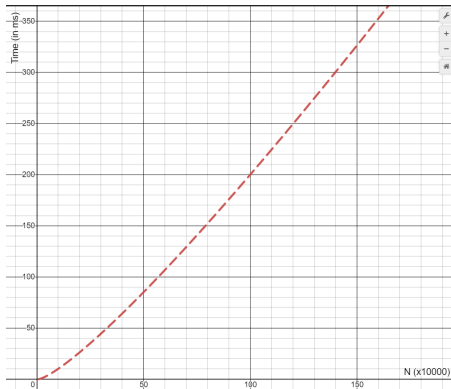
```
if end > beg then  
  mid = (beg + end) / 2  
  invCount +=  
    enhancedMergeSort(arr, left, mid)  
    invCount +=  
    enhancedMergeSort(arr, mid + 1, right)  
    invCount += mergeTwoAr-  
      rays(arr, left, mid + 1, right)  
end if  
return invCount
```


Time Complexity

In the **Divide and Conquer** algorithm , the input array is divided into two halves each time it is processed. As such it can be expressed with following recurrence relation,

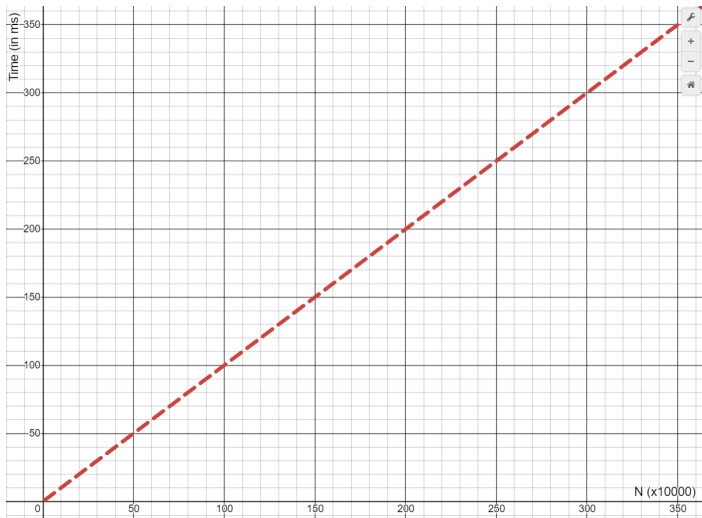
$$T(n) = 2T(n/2) + \theta(n).$$


$O(N \log(N))$



Space Complexity

$O(N)$



-  Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein
-  Introduction to Design and Analysis of Algorithms by Anany Levitin
-  Algorithms by Robert Sedgewick and Kevin Wayne