Counting Number of Inversions in an Array

B.tech Semester 4th

Harshdeep Singh Pruthi
IIT2019105

Jayaram Naik
IIT2019106

Shivansh Gupta
IIT2019107

***Abstract*—In this report we have discussed how to Count Number of Inversions in an Array. We have elaborated on two approaches a Naive approach and a Divide and Conquer approach.**

***Index Terms*—Inversion, Divide and Conquer, Merge Sort**

I. Introduction

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in the reverse order, the inversion count is the maximum. Two elements a[i] and a[j] form an inversion if a[i] $>$ a[j] and i $<$ j .

II. Algorithm Design

A. Naive

We perform a brute force by iterating over all pairs of indices and check whether this pair satisfies the condition if a[i] $>$ a[j] then i $<$ j .

B. Divide and Conquer

Suppose we want to count the number of inversions inside the array A. Let's divide it into two equal Arrays and call the first one L and the second one R. Now an inversion (i, j) can have one of the following types:

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

Consider the first type. The problem turns out to computing for each index $i \in L$, the number of indices $j \in R$ such that $L_i > R_j$.

Now suppose that both arrays L and R are sorted increasingly, then we can use the two-pointers technique to solve it. That's because indices j that satisfy the condition for an index i always form a prefix from the array R, when i gets bigger, the length of the prefix increases.

But, what about the second and third types? The answer is simple: just perform the same algorithm recursively for both L and R independently.

Back to the first type, we supposed that both L and R are sorted increasingly. To achieve that, the algorithm must sort them. In other words, after applying the algorithm to both L and R, they become sorted. To sort A we just merge the two sorted arrays L and R into A.

Identify applicable funding agency here. If none, delete this.

III. Algorithm Analysis(Divide and Conquer)

Algorithm is very similar to the merge sort algorithm. The only difference is that we make another iteration on the elements of the array A when we count the number of inversions of the first type (the first iteration was when we merged the two subarrays).

Therefore, the complexity is the same as the merge sort algorithm, which is $O(n \times log(n))$.