Synopsis

Problem Statement

Let **A[ i…n ]** be an array of **n** distinct real numbers. A pair **( A[ i ], A[ j ] )** is said to be an index-value inversion if **A[ i ] = j** and **A[ j ] = i** . Design an algorithm for counting the number of index-value inversions.

Design Technique

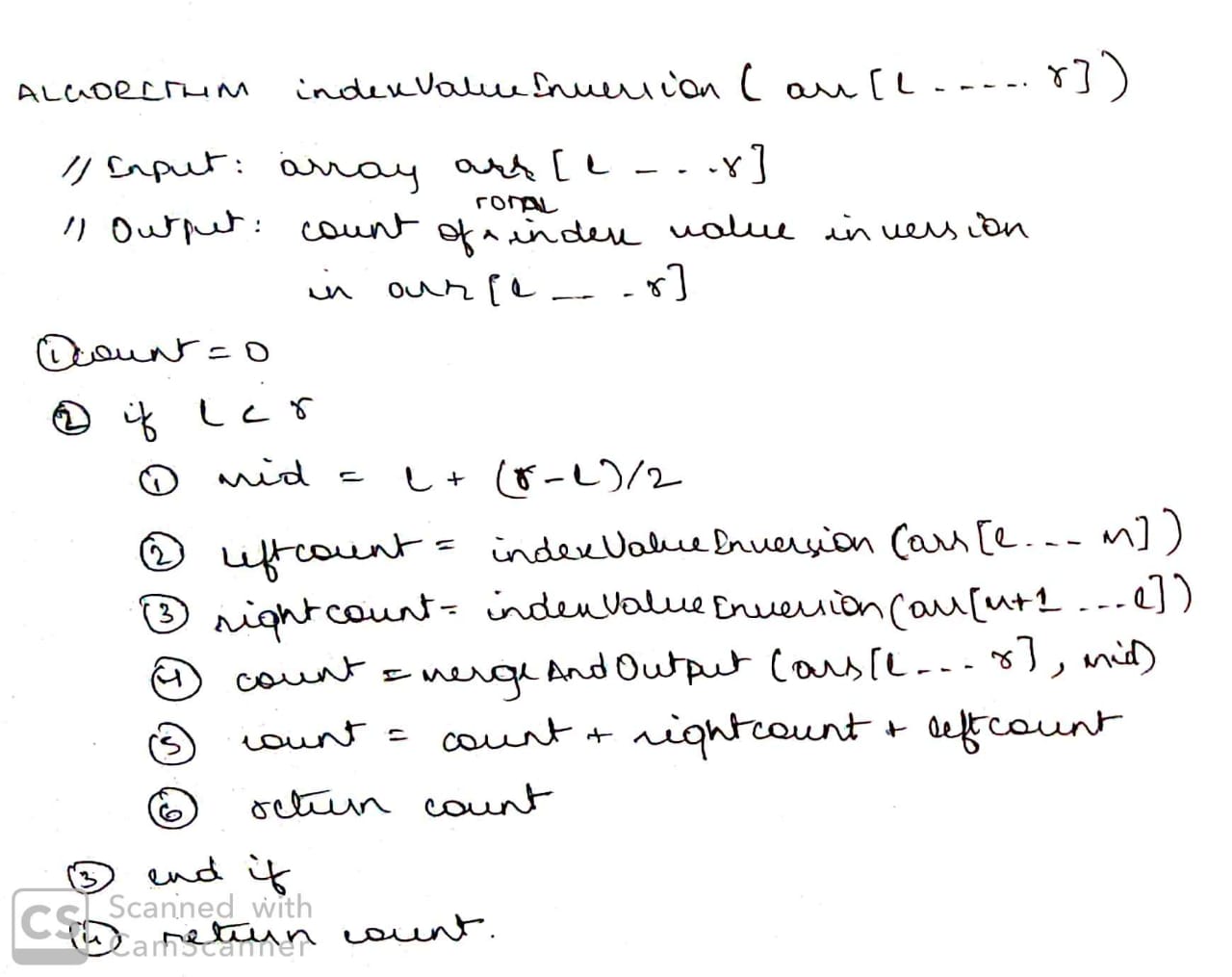
We can obtain an efficient solution for the above problem statement if we use **Divide And Conquer** **Technique**. Here, we divide the input array into two halfs. The number of **index value inversions from** **both the halves are added** to find the total number of index value inversions in the complete array. This procedure is **executed** **recursively**.

We can expect our solution to be similar to that of mergesort.

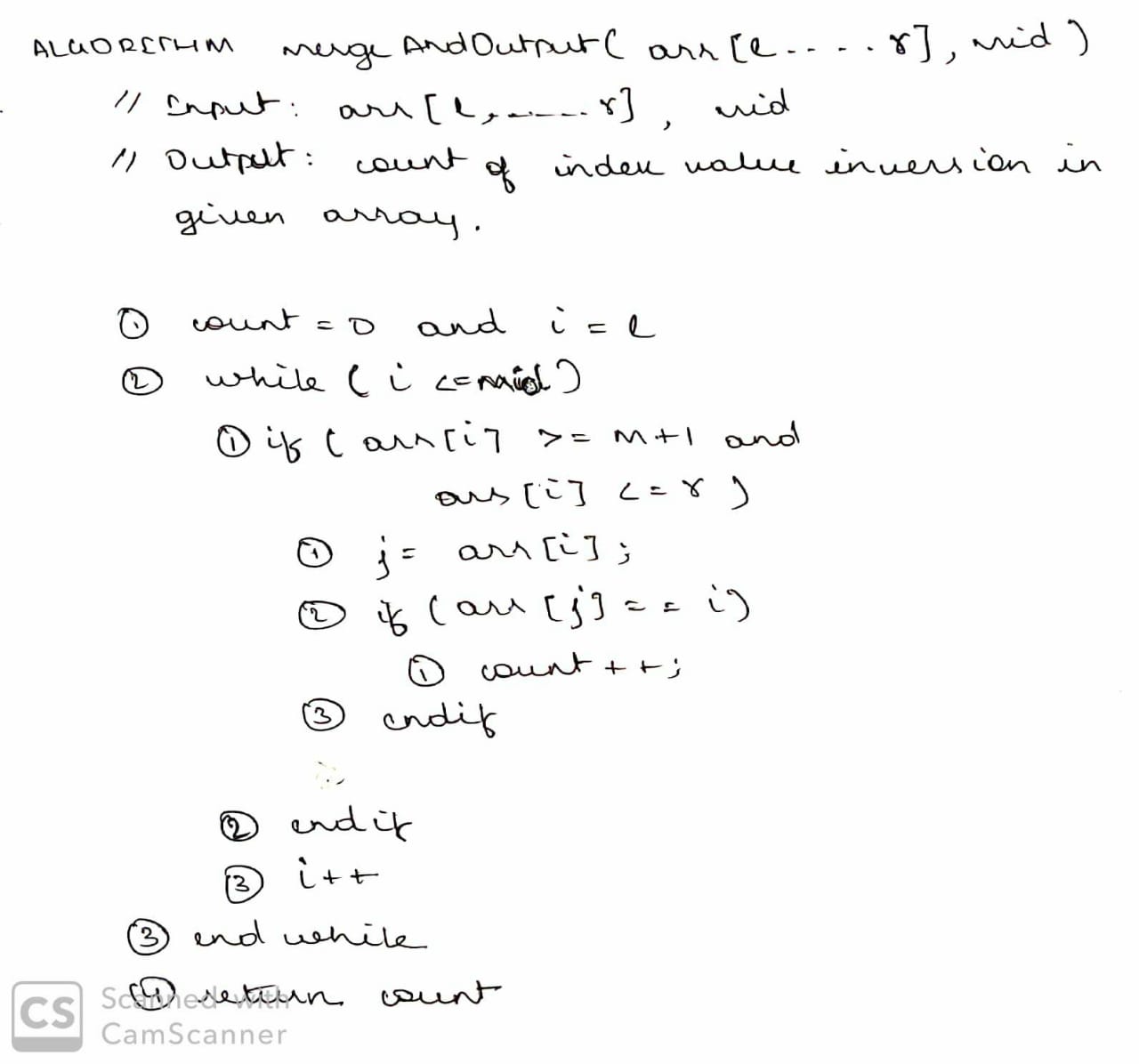
Data Structure

**Array Data Structure** is used to solve the above problem statement. We have used this data structure because :

* It is **easier to code** our algorithm around array data structure.
* It gives an **efficient solution**.
* We face **no notable loss** in time or memory efficiency.

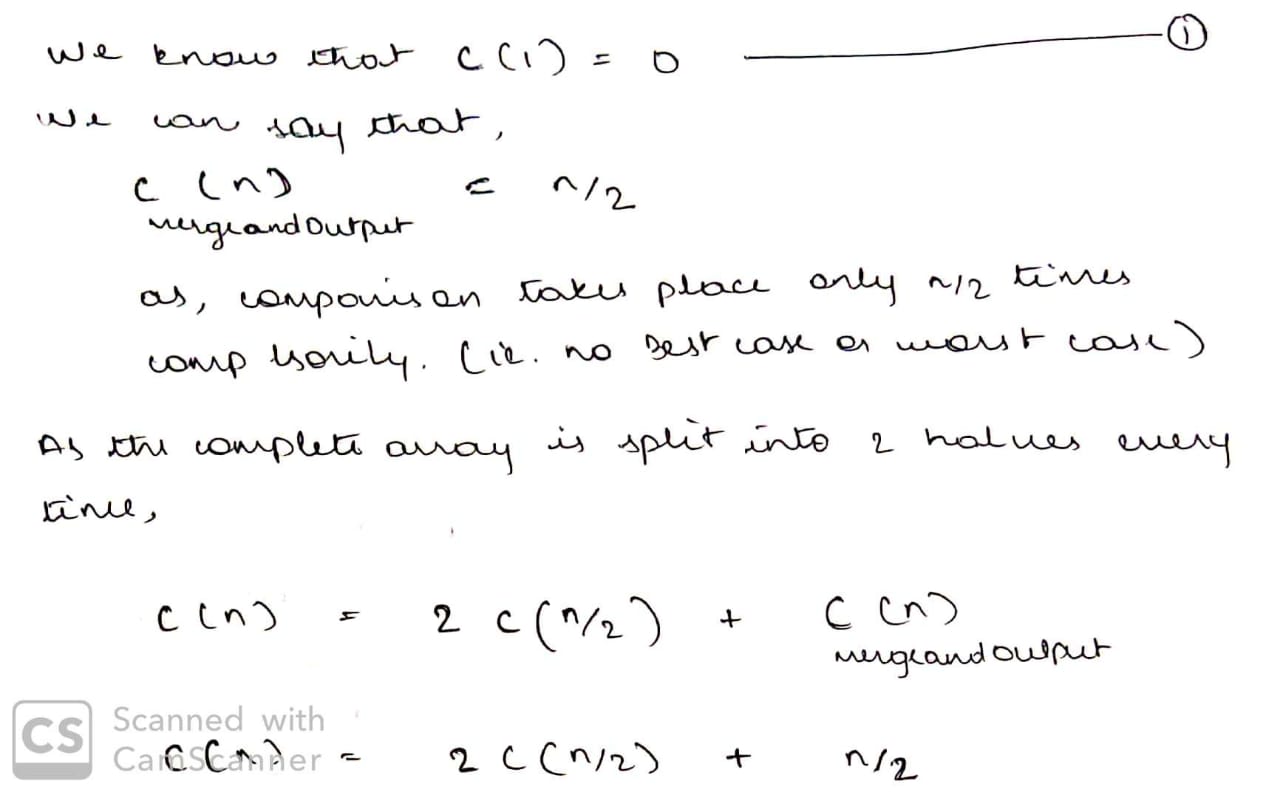
Algorithm

Algorithm

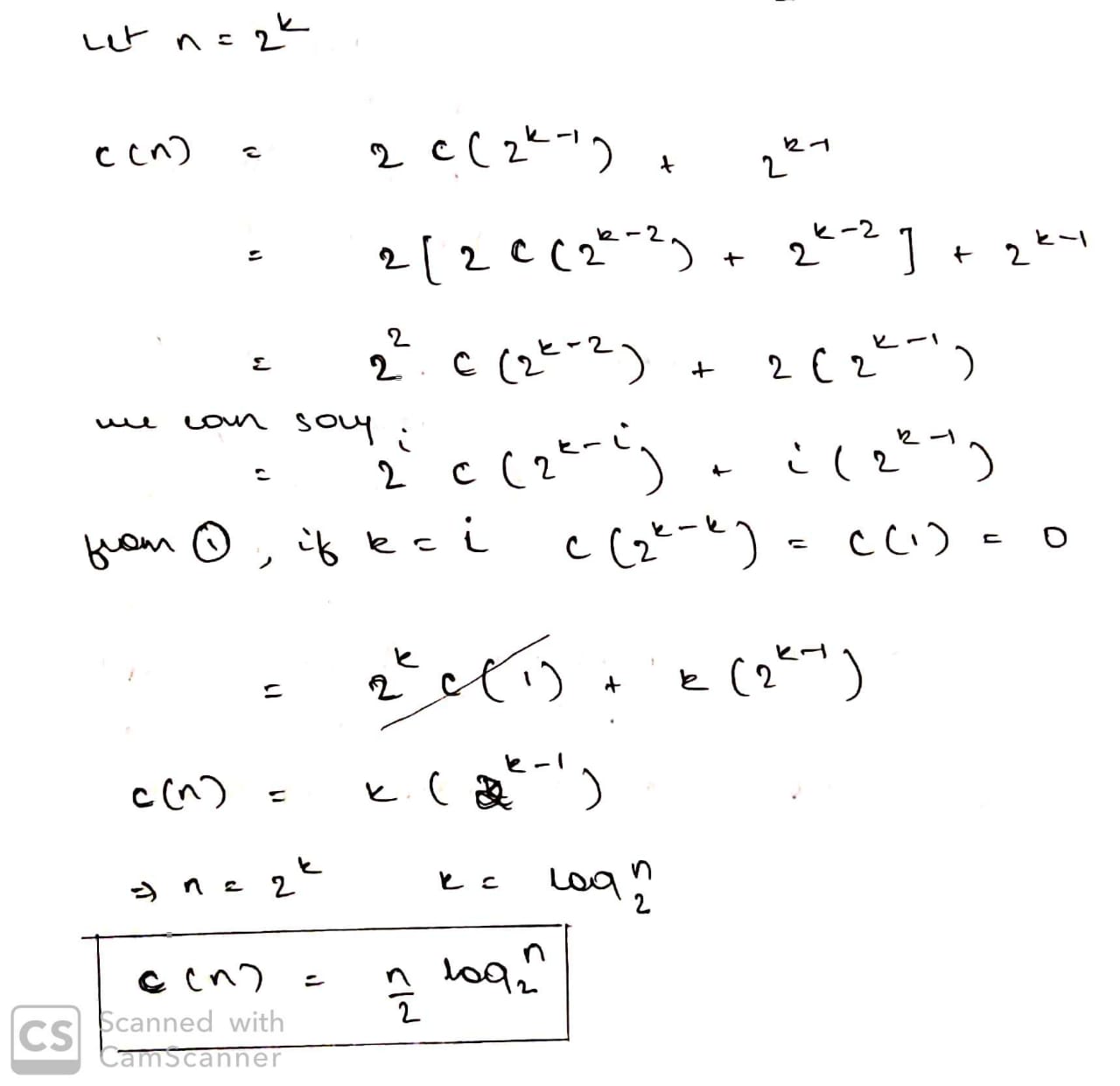


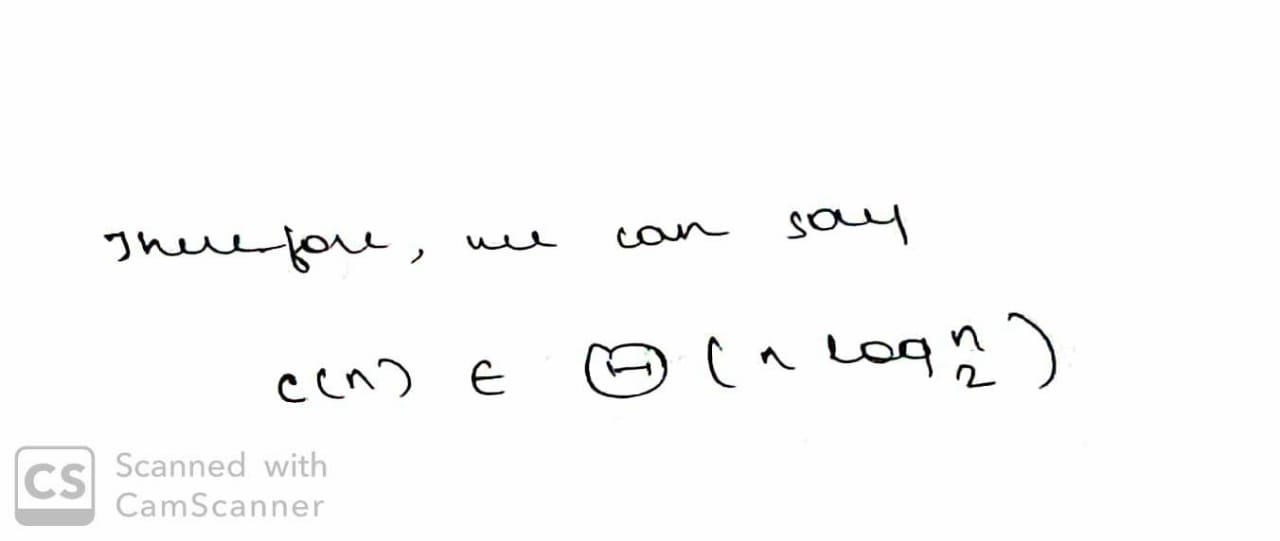
Algorithm Analysis

Our algorithm **mergeAndOutput( A[ l…r ] , mid )** will run for **n/2** times compulsorily. That is, the while loop( present at line location 2 ) while run n/2 times without fail.

Hence, we **need not check our algorithm’s efficiency for best or worst case** as it will run for fixed amount of time everytime.

Continued On The Next Page…

Algorithm Analysis



We have obtained an efficiency class of n\*log(n).

Algorithm Analysis

We can justify our efficiency class by the help of **Master Theorem.**

**function f**(input x size n)

**if**(n < k)

solve x directly and return

**else**

divide x into **a** subproblems of size n/b

call f recursively to solve each subproblem

Combine the results of all sub-problems

So, according to master theorem the runtime of the above algorithm can be expressed as:

**T(n) = aT(n/b) + f(n)**

Where,n = size of the problem  
a = number of subproblems in the recursion and a >= 1  
n/b = size of each subproblem  
f(n) = cost of work done outside the recursive calls like dividing into subproblems and cost of combining them to get the solution.

**Advance version of master theorem** that can be used to determine running time of divide and conquer algorithms if the recurrence is of the following form

**T(n) = aT(n/b) + θ(nk logpn)**

Where, n = size of the problem  
a = number of subproblems in the recursion and a >= 1  
n/b = size of each subproblem  
b > 1, k >= 0 and p is a real number.

**Then**,

1. if a > bk, then T(n) = θ(nlogba)
2. if a = bk, then  
   (a) if p > -1, then T(n) = θ(nlogba logp+1n)  
   (b) if p = -1, then T(n) = θ(nlogba loglogn)  
   (c) if p < -1, then T(n) = θ(nlogba)
3. if a < bk, then  
   (a) if p >= 0, then T(n) = θ(nk logpn)  
   (b) if p < 0, then T(n) = θ(nk)

Algorithm Analysis

**Our Algorithm will follow the reccurence equation,**

**T(n) = 2T(n/2) + O(n)**

a = 2, b = 2, k = 1, p = 0  
bk = 2. So, a = bk and p > -1 [Case 2.(a)]  
then, T(n) = θ(nlogba logp+1n)  
**T(n) = θ( nlogn )**

The efficiency classes obtained by analysis and obtained by Advanced Master Theorem are same. Our Analysis is justified.

Our Algorithm has an efficiency class of **nlogn.**

Source CodeSource Code



Output



Output



References

<https://www.geeksforgeeks.org/merge-sort/>

Project Repository

<https://github.com/aksharsramesh/ADAproject>