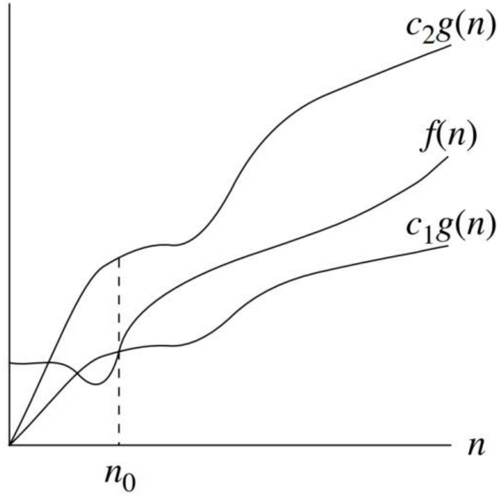
Algorithm Design & Analysis (CSE222)

Lecture-3

Recap

Growth Functions:



$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

f(n) = Θ(g(n)) f(n) = o(g(n)) f(n) = ω(g(n))

- Recurrences & Masters' Theorem
- Graph Notations
- BFS

Outline

- Revisit BFS: Bipartite Testing
- Divide & Conquer
- Counting Inversions

Revisit BFS

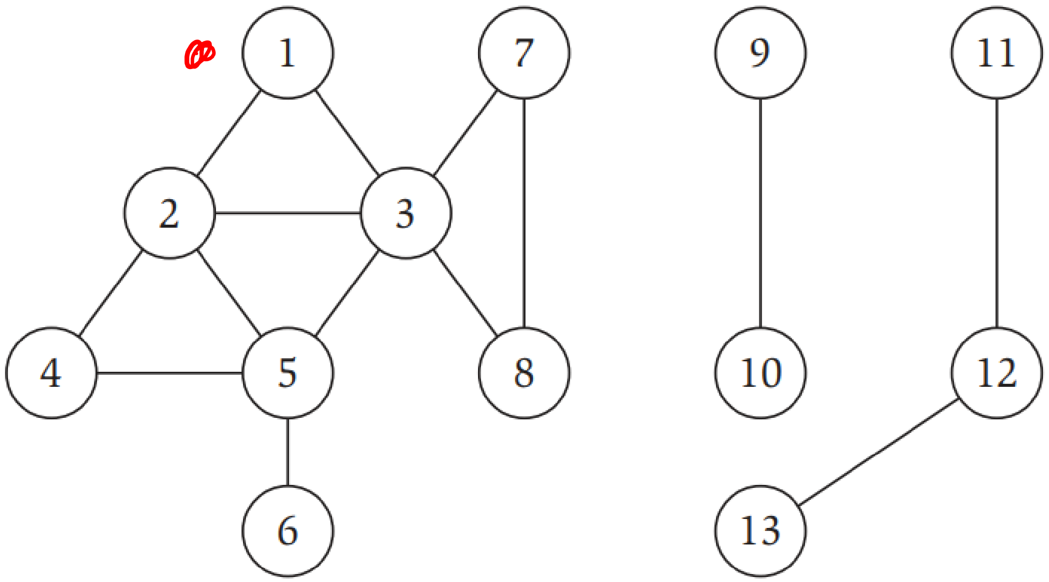
Breadth First Search (BFS): Starting from a vertex it traverses the graph layer by layer.

$L_0 = 1$

$L_1 = 2, 3.$

$L_2 = 4, 5, 8, 7$

$L_3 = 6$



Revisit BFS

Applications

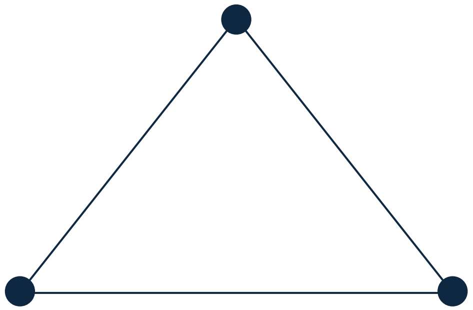
- Finds if two vertices are connected.
- Finds the shortest path between two connected vertices.

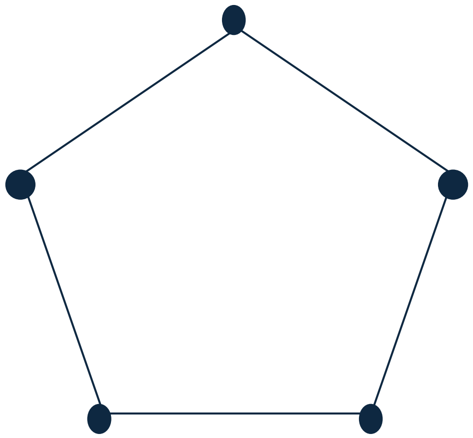
For $j \geq 1$, the layer L_j produced by BFS consists of all the nodes that are exactly at distance j from s (start node).

Running time of BFS? $O(|E| + |V|)$

Testing Bipartiteness using BFS

Is it a bipartite graph?





Testing Bipartiteness using BFS

A graph G having odd length cycle cannot be a bipartite graph.

A bipartite graph cannot contain an odd length cycle.

Testing Bipartiteness using BFS

Start from any vertex and colour it red.

Go to its neighbours and colour them blue.

Go to their neighbours and colour them red.

Continue until all the vertices are coloured.

If there is an edge whose vertex pairs are of same colour then the graph is not bipartite.

Think: Let T be a BFS tree, let u and v be nodes in T belonging to layers L_i and L_j respectively, and let (u, v) be an edge of G. Then i and j differ by at most 1.

Testing Bipartiteness using BFS

The bipartite testing algorithm is identical to BFS.

Colour all the vertices in the even numbered layers by red.

Colour all the vertices in the odd numbered layers by blue.

Claim of the Algorithm

Let G be a connected graph and let layers L_1, L_2, … be produced by the BFS starting at vertex s.

1. If there is no edge joining two vertices of same layer then every edge has a vertex coloured red and another vertex coloured blue. Hence, G is a bipartite graph.
2. If there is an edge joining two vertices of the same layer then G contains odd length cycle. Hence, G is not a bipartite graph.

Analysis

1. If there is no edge joining two vertices of same layer then every edge has a vertex coloured red and another vertex coloured blue. Hence, G is a bipartite graph.

Recall: Let T be a BFS tree, let u and v be nodes in T belonging to layers L_i and L_j respectively, and let (u, v) be an edge of G. Then i and j differ by at most 1.

- This implies edges in G are either in same layer or adjacent layers.
- No edges in same layer (by assumption)
- So every edge joins two vertices in adjacent layers.
- So, in our colouring every edge has two vertices from different colours.

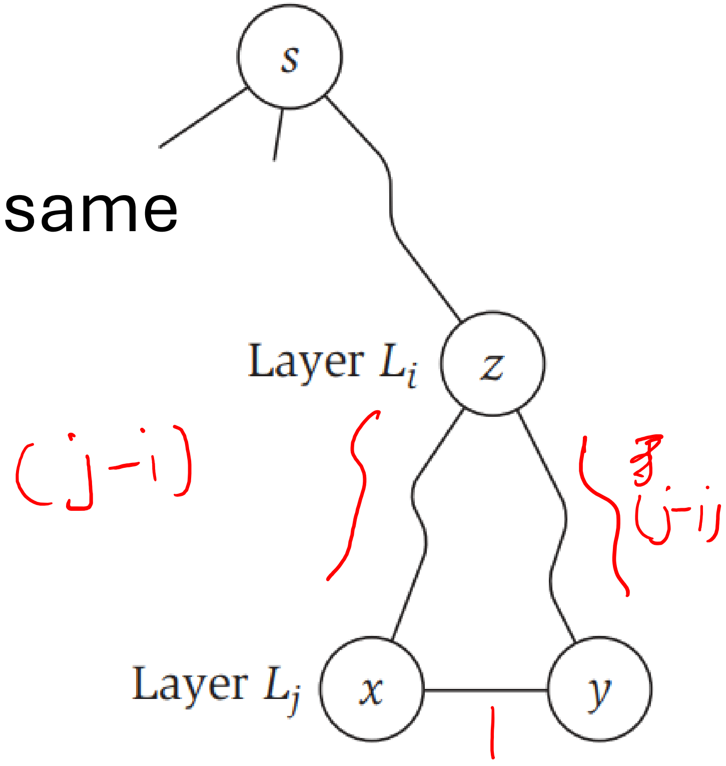
Analysis

2. If there is an edge joining two vertices of the same layer then G contains odd length cycle. Hence, G is not a bipartite graph.

As there is an edge between two vertices in the same layer. So consider this situation.

Now compute the length of the cycle (z,x,y,z).

$2(j-i)+1$



Outline

- Revisit BFS: Bipartite Testing
- Divide & Conquer
- Counting Inversions

Divide & Conquer

$T(n) = 3T(n/3) + n$

$T(1) = O(1)$

Divide & Conquer

$T(n) = 3T(n/3) + n$

$T(1) = O(1)$

- Divide the input into small pieces.
- Solve subproblems on these pieces separately by recursions.
  - Subproblems are the same problem solved on a smaller input size
- Combine results into overall solution.

Merge Sort

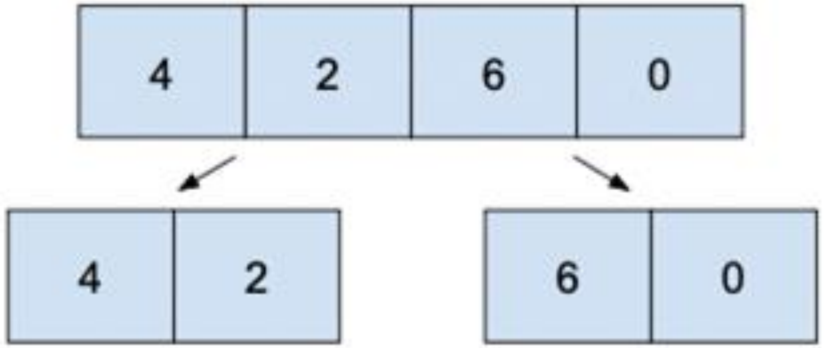
Input: Array $\boldsymbol{A}$ of n numbers.
Output: Sorted array of n numbers.

- Divide array into $\boldsymbol{A}_{\textbf{Left}}$ = $\boldsymbol{A}$**[1,2, …, n/2]** and $\boldsymbol{A}_{\textbf{Right}}$.= $\boldsymbol{A}$**[n/2+1, …, n]** (Divide).
- Recursively sort $\boldsymbol{A}_{\textbf{Left}}$ and $\boldsymbol{A}_{\textbf{Right}}$.(Conquer).
- Merge two sorted arrays from $\boldsymbol{A}_{\textbf{Left}}$ and $\boldsymbol{A}_{\textbf{Right}}$ into one sorted array (Combine).

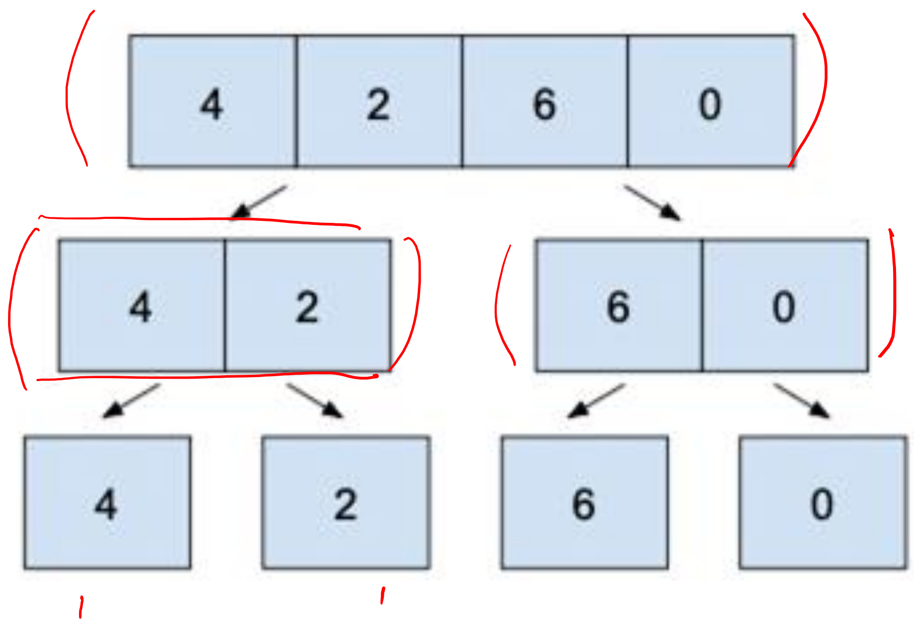
Merge Sort

4	2	6	0

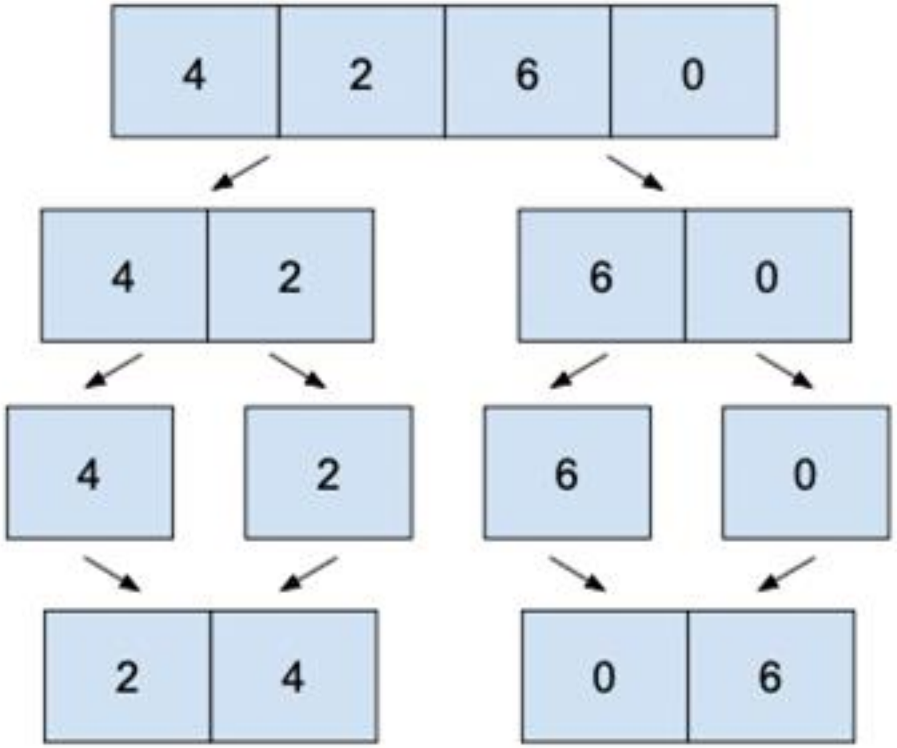
Merge Sort



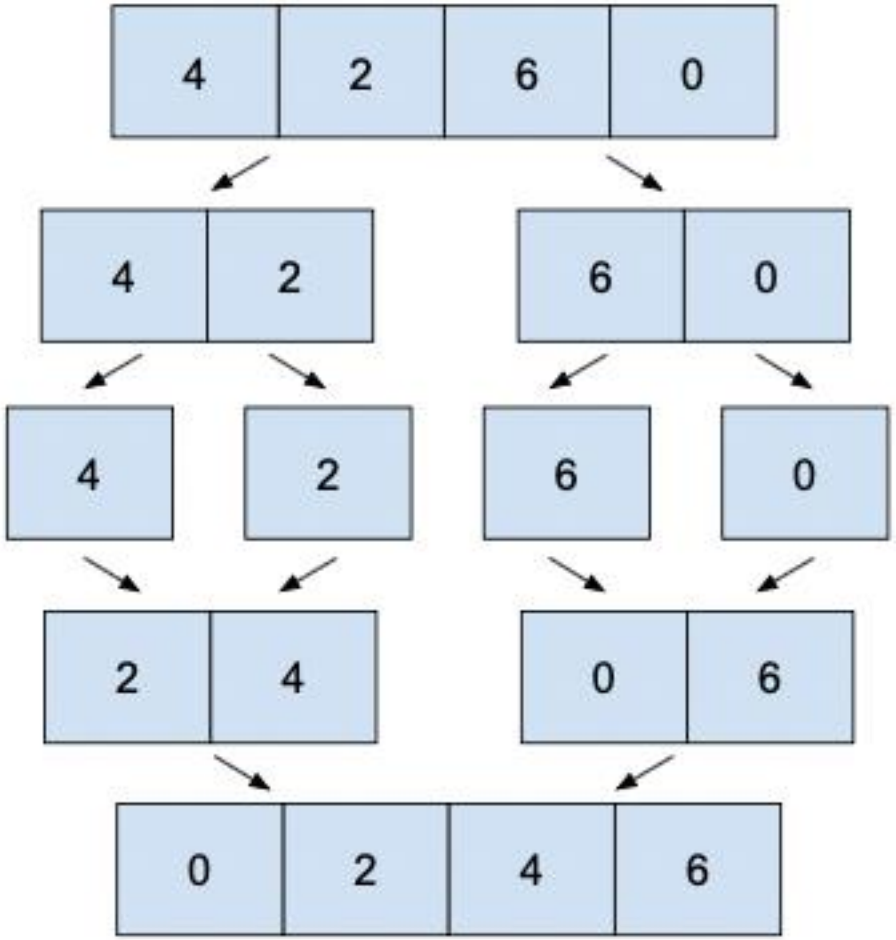
Merge Sort



Merge Sort



Merge Sort



Merge Sort

$\textsc{Merge-Sort}(A, \ell, r)$

$\quad$ **if** $\ell < r$

$\qquad m = \lceil (\ell + r)/2 \rceil$

$\qquad \textsc{Merge-Sort}(A, \ell, m)$

$\qquad \textsc{Merge-Sort}(A, m + 1, r)$

$\qquad \textsc{Merge}(A, \ell, m, r)$

Outline

- Revisit BFS: Bipartite Testing
- Divide & Conquer
- Counting Inversions

Measure Similarity

$A = (1\ \ 3\ \ 2\ \ 6)$

$A(1)\ \ A(2)\ \ A(3)\ \ A(4)$

- Let there are ***n*** songs.
- You rank them as ***1***, ***2***, ..., ***n***
- Your friend ranks them as $\boldsymbol{a_1}$, $\boldsymbol{a_2}$, ..., $\boldsymbol{a_n}$
- How to measure similarity?
- Compute the number of *out of order* pairs.
- A pair is out of order if $\boldsymbol{i} < \boldsymbol{j}$ but $\boldsymbol{a_i} > \boldsymbol{a_j}$ (a.k.a. *Inversion*).

1 3 2 6

1 2 3 6

Questions

- What is the number of inversions in a sorted (ascending order) array?
- What is the maximum possible inversions in a array of length n? Example?

Counting Inversions

Problem: In an array of ***n*** numbers count the number of inversions.

A Naive algorithm.

Input: Array ***A*** of ***n*** numbers.
Output: Number of inversion in ***A***.

SIMPLE-COUNTIVERSIONS(A, n)

$Count = 0$
for every $(i, j) \in n^2 - n$
  if $\left(i > j \text{ and } A[i] < A[j]\right)$
    $Count = Count + 1$

Running time?

Count Inversions in Sorted Arrays

$C = \cancel{0}\ 3 -\ 2+1 = \cancel{2}\ \cancel{4}\ 5$

Left

1	5	9
1	2	3

2	4	7
1	2	3

Right

1	2	4	5	7	9

Count Inversions in Sorted Arrays

Input: Two Sorted arrays
Output: Count Inversion

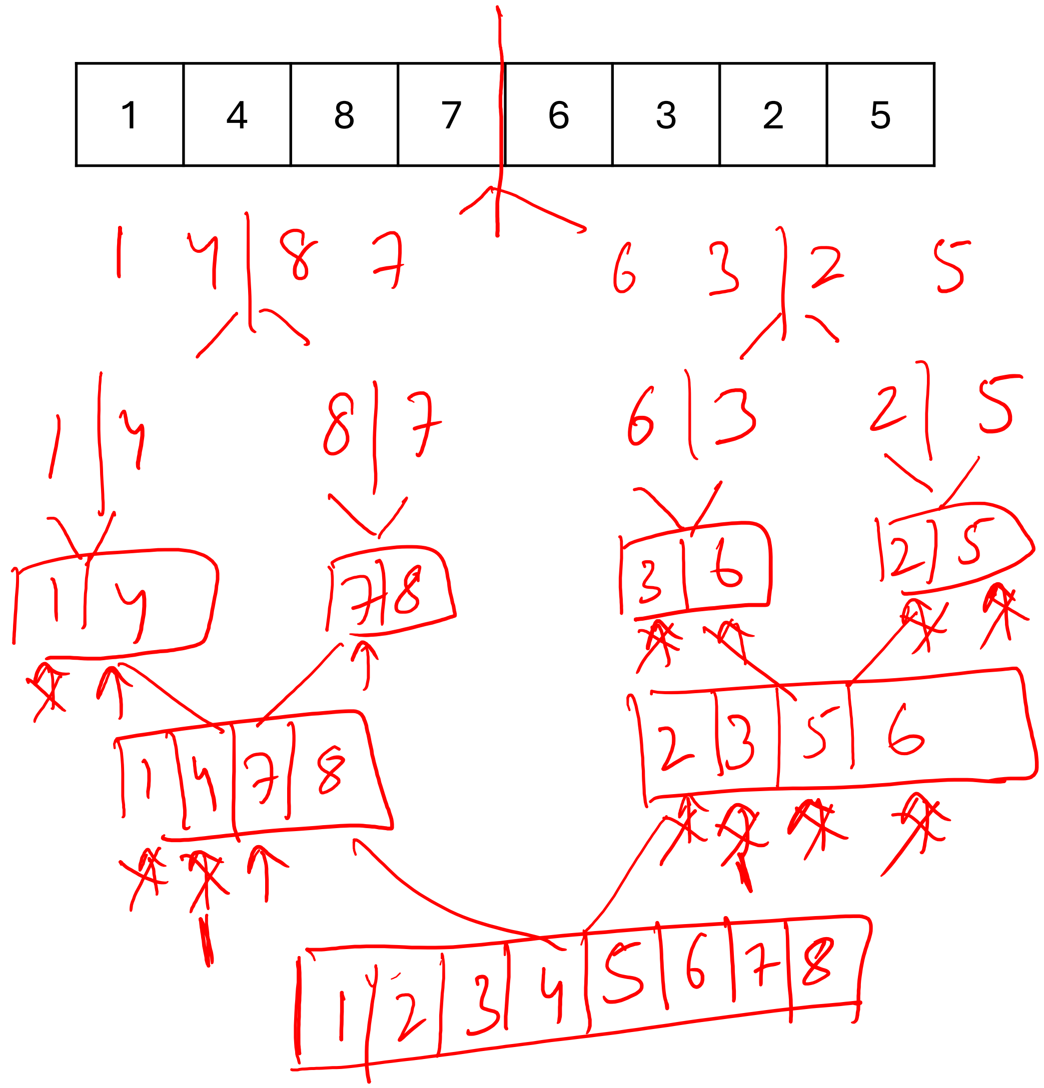
MERGE-COUNT(L, R)

Initialize $pt_\ell = 1; pt_r = 1; i = 1$ and $Count = 0$
Initialize an empty array A of size $|L| + |R|$.
while ($pt_\ell \leq |L|$ and $pt_r \leq |R|$)
    if ($L[pt_\ell] \leq R[pt_r]$)
        $A[i] = L[pt_\ell]$ and $pt_\ell = pt_\ell + 1$
    else
        $A[i] = R[pt_r]$; $pt_r = pt_r + 1$ and $Count = Count + |L| - pt_\ell + \cancel{1}$
    $i = i + 1$
if ($pt_\ell > |L|$)
    Append remaining elements of R into A
if ($pt_r > |R|$)
    Append remaining elements of L into A
Return ($Count, A$)

Running Time? $O(n)$

Counting Inversion Using DC

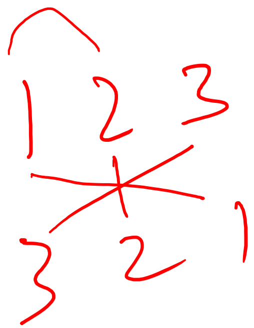
$C = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{4}\ \cancel{5}\ \cancel{8}\ \cancel{11}\ \cancel{13}\ 15$



$|L| - 2 + 1$

Counting Inversion Using DC

Input: Unsorted array
Output: Count Inversions



Counting Inversion Using DC

Input: Unsorted array
Output: Count Inversio

COUNT-INVERSION(A)

if $|A| = 1$
else
    $m = \left\lceil \frac{|A|}{2} \right\rceil$
    $A_{Left} = A[1, \ldots, m]$ and $A_{Right} = A[m+1, \ldots, |A|]$
    $(C_{Left}, A_{Left}) =$ COUNT-INVERSION(A_{Left})
    $(C_{Right}, A_{Right}) =$ COUNT-INVERSION(A_{Right})
    $(C_{Split}, A) =$ MERGE-COUNT(A_{Left}, A_{Right})
    $Count = C_{Left} + C_{Right} + C_{Split}$
Return $(Count, A)$

Is it correct? Lets analyze!

Reference

Slides

Algorithms Design by Kleinberg & Tardos - Chp 3.4 & 5.3