



AfterAcademy



OFFER

Android Online Course by MindOrks

Start your career in Android Development. Learn by doing real projects.

[ENROLL NOW](#)

Admin AfterAcademy
30 Apr 2020

Inversion Count in an Array-Interview Problem

Interview question

Inversion count in an array



Asked in



Difficulty: Medium

Asked in: Amazon, Google

Understanding the Problem:

Given an array of integers $A[]$, if $i < j$ and $A[i] > A[j]$ then the pair (i, j) is called the inversion of an array $A[]$. You need to write a program to find the total counts of inversion in an array $A[]$.

The inversion count of an array suggests how close the array is from being sorted. If the array is already sorted the inversion count in this case will be 0. Obviously, the inversion count will be maximum when the array is reversely sorted(descending order).

For Example:

Input: $A[] = \{3, 2, 1\}$

Output: 3

Explanation: The inversion pairs in the given array are $(3,2)$, $(3,1)$ and $(2,1)$.

Input: $A[] = \{4, 1, 3, 2\}$

Output: 4

Explanation: There are four inversion pairs in the given array, $(4,1)$, $(4,2)$, $(4,3)$ and $(3,2)$.

Possible questions to ask the interviewer:

- Are there any duplicate elements in the array? (**Ans:** The array has only

unique elements)

You can try solving this problem [here](#).

Solutions

We are going to see four different solutions to this problem. The first one will be an obvious, brute force solution while in the second solution we will work on optimisation and come up with an efficient solution. The third solution will be based on using an ordered set and the fourth one will use the AVL tree to solve this problem.

- Brute Force Solution
- Optimised Solution Based on Divide and Conquer
- Solution Using Ordered Set
- Solution Using AVL Tree

Brute Force Solution

Solution idea

The idea is very simple, we will start from the initial index and will traverse the entire array. While traversing the array for each index, we will find the number of elements which are smaller than it and are present in the right side of that element. That is for index i , we will look for smaller elements present from the index $(i+1)$. With every such encounter of smaller elements, we will increase our counter. We will follow this approach for every element and at last, the value of the counter will be our inversion count.

Solution steps

- We will start from the initial index and keep traversing till the end.
- For every element present at index i , we will find the number of elements which are smaller than this element and are present on the right side of it.
- We will find the count of such smaller elements for every index and will add it to the counter.
- We will print the value of the counter, which will be our inversion count.

Pseudo-code

```
int inversionCount(int A[], int size)
{
    int inversion_count = 0
    for(i = 0 to size-2; i=i+1)
    {
        for(int j = i+1 to size-1; j=j+1)
        {
            if(A[j] < A[i])
                inversion_count = inversion_count+1
        }
    }
    return inversion_count
}
```

Complexity Analysis

Time Complexity: $O(N^2)$

Space Complexity: $O(1)$

Critical ideas to think!

- Can you think of reducing the time complexity? What if we divide the array into smaller halves and find the inversion count for each half? How can we find the total inversion counts in that case?

Optimised Solution Based on Divide and Conquer

Solution idea

This solution idea is based on the divide and conquer algorithm. Here, we will be dividing our array into two halves similar as in the merge sort algorithm. For each half, we will be getting the inversion counts using recursion. The total counts of inversion will be the sum of inversions in the first half, second half as well as the inversion counts during the process of merging.

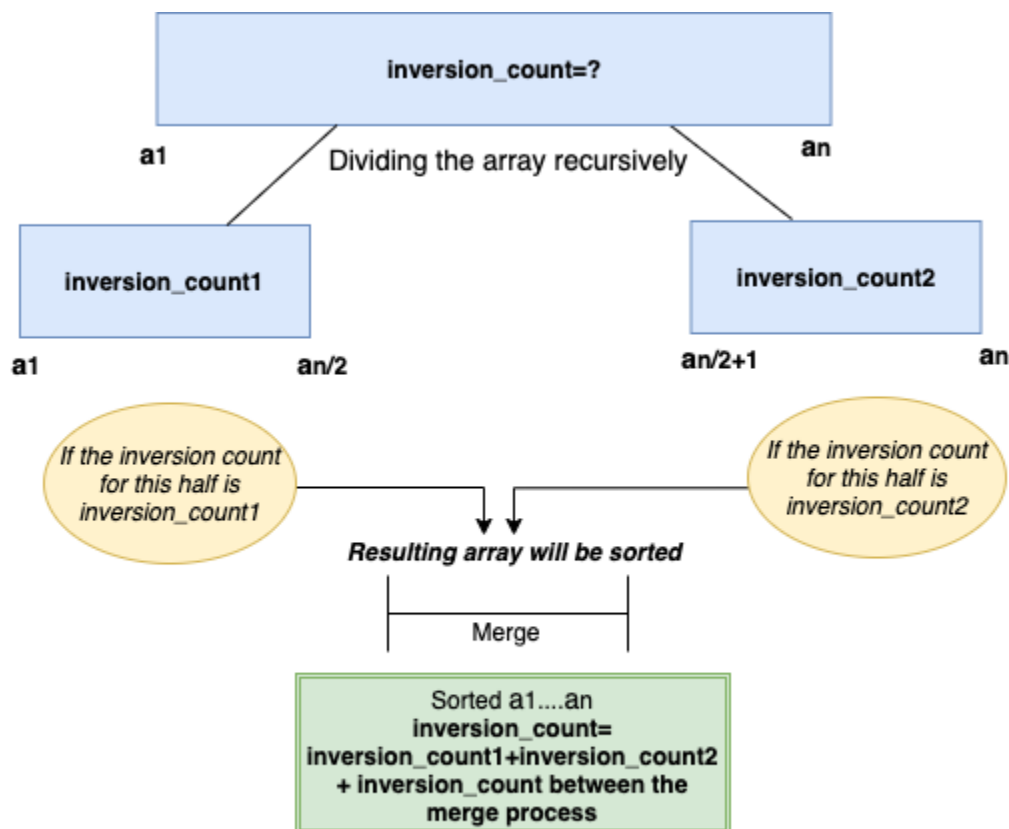
During the merge process, the inversion count is found by comparing elements of both the halves. We will use two pointers that will compare the indexes of both the array halves.

Getting inversion count during merge?

Suppose if the two halves of the array are like this: `firstHalf[] = {2, 4, 7}` and `secondHalf[] = {1, 3, 6}`. Now, the i th element (where $i = 0$) is greater than the corresponding j th element (j is also 0). Obviously, j th element will be smaller than all the rest elements of `firstHalf` array as this array is sorted. That is if $1 < 2$, 1 will be smaller than 4 and 7 also.

Now, the inversion count, in this case, will be 3. Here the value of mid will

be 3. So $(\text{mid}-i)$ i.e. $(3-0)$ will also give 3. So we always increment the inversion count by $(\text{mid}-i)$ in such cases. This is how it works.



Understanding Inversion Count using Divide and Conquer Approach

Solution steps

- First, we will split our array into two halves recursively similar as in the case of merge sort.
- The recursion will continue until the base condition that is only one element is left.
- We will count the number of inversions in the first half, second half (again recursively) as well as the number of inversions during the merge process.
- The main trick is to find the number of inversions during the merge, for

this, we will maintain two pointers, *i* and *j*. *i* will point to the starting element of the left half and *j* will point to the starting element of the second half. We will compare the elements at both the positions. If *i*th element is smaller than *j*th element just add it to the new sorted list. Else, increment the count of inversions by (*mid*-*i*).

Pseudo-code

```
int merge(int originalArr[], int temp[], int start, int mid, int end)
{
    int i = start;
    int j = mid;
    int k = start;
    int inversion_count = 0;
    while(i <= mid-1 && j<= end)
    {
        if(originalArr[i]< originalArr[j])
            temp[k++] = originalArr[i++]
        else{
            inversion_count += mid-i;
            temp[k++] = originalArr[j++]
        }
    }
    while(i <= mid-1)
        temp[k++] = originalArr[i++]
    while(j <= end)
        temp[k++] = originalArr[j++]
    for(int i = start; i <= end; i++)
        originalArr[i] = temp[i]
    return inversion_count;
}

int mergeSort(int originalArr[], int temp[], int start, int end)
{
    int inversion_count = 0;
    if(start < end) {
        int mid = (start+end)/2;
        inversion_count += mergeSort(originalArr, temp, start, mid);
```

```
        inversion_count += mergeSort(originalArr, temp, mid+1, end);
        inversion_count += merge(originalArr, temp, start, mid+1, end);
    }
    return inversion_count;
}

int inversionCount(int A[], int size)
{
    int temp[size]
    return mergeSort(A, temp, 0, size-1)
}
```

Complexity Analysis

Time Complexity: $O(N \log N)$

Space Complexity: $O(N)$ (Why?)



NEW

Android App Development Online Course by MindOrks

Start your career in Android Development. Learn by doing real projects.

[CHECK NOW](#)

Solution Using Ordered Set

Solution idea

Here we will be using an ordered set where all the elements are stored in a sorted manner. The idea is, we will insert the first element into the set and then for every next element to be inserted we will find the first element's index that is larger than the current element(which is to be inserted). By getting the index of that just greater element we can find the distance between the two indices and add them to our `inversion_count` variable(that will store total inversion counts). We will keep doing this for all the elements. The value of `inversion_count` , at last, will be our final

inversion count.

Solution steps

- We will create an ordered set say S and initialise a variable `inversion_count` to 0, which will store our `inversion_count`.
- We will insert the first element i.e. 0th indexed element into the set.
- For every element from index i to $(\text{size}-1)$, where `size` is the actual length of the array, we will keep finding the smallest index from the beginning whose value is greater than i th element(upper bound).
- Once we get that index say j , we will take the difference $(j-i)$ and will add it to our `inversion_count` variable.
- We will keep doing this and at last, the value of `inversion_count` will be our desired output.

Pseudo-code

```
int getUpperBound( ordered_set S, int elem)
{
    //returns the upper bound for an element elem
}
int inversionCount(int A[], int size)
{
    int ordered_set S
    S.insert(A[0])    //Inserting first element into the set

    inversion_count = 0
    for(int i = 1 to size-1)
    {
        S.insert(A[i])
        int firstGreaterElemIndex = getUpperBound(S, A[i])
        inversion_count = inversion_count + (firstGreaterElemIndex-i)
    }
}
```

```
    }  
    return inversion_count  
}
```

Complexity Analysis

Time Complexity: $O(N^2)$

Space Complexity: $O(N)$

Critical ideas to think!

- Why are we using ordered set here? Which data structure is used for implementing an ordered set?

Solution Using AVL Tree

Solution idea

The idea in this solution is to use a self-balancing Binary Search Tree(BST) like AVL or Red-Black Tree with augmentation to keep the size of the subtree in every node. With every insertion of a new node, we will traverse the array from start to end and keep adding the elements to the tree. Now if $a > b$ where a appears before b in the array then obviously a will be in the right subtree. Whenever we will encounter any such node whose value is smaller than the root node's value, we will add this to the left node and increment the value of the inversion count with the number of elements present in the right subtree plus one for the root node. We are adding the size of right subtree as all elements of the right subtree will form a pair $(elem, b)$ for element b where $elem > b$.

Solution steps

- Create an AVL Tree augmenting the size of the subtree in all nodes.
- Start traversing the array A[] from the beginning.
- Follow the normal procedure of insertion and insert every element in the AVL tree.
- The number of elements which are greater than the current element can be found by adding the size of the nodes present in the right subtree. Obviously, if there are greater elements already present in the array then, we can be sure that it's index is less than the current element's index.
- Whenever we are adding node such that the value of the node is less than the root and it is being inserted in the left subtree, we will increment the `inversion_count` by size of right subtree plus one.
- Output the value of the `inversion_count` when the array is fully traversed.

Pseudo-code

```
class avlNode
{
    int value
    avlNode left
    avlNode right
    int height
    int size //size of the subtree
}
inversion_count = 0 //global variable
class avlTree
{
    //root element of the tree
```

```
avlNode root
// constructor of the class
avlTree()
//creates a new node with value val
newNode(val)
//return height of the node
getHeight()
//returns size of the subtree
getSize()
insertNode(root, val)
{
    if(root is NULL)
        newNode(val)

    if(root.val > val)
    {
        insertNode(root.left, val)
        //inversion_count needs to be updated
        inversion_count = inversion_count + getSize(root.right) + 1

    }
    else
    {
        insertNode(root.right, val)
    }
    //update height of the node
    root.height = max(getHeight(root.left), getHeight(root.right)) + 1
    //update size of the node
    root.size = getSize(root.left) + getSize(root.right) + size + 1
    //balance the tree if there is any imbalance
    balance()
}
//insert node with value val to the tree
insert(val)
{
    insertNode(root, val)
}
//balances the tree using rotations
balance()
//left rotation of the tree
```

```
rotateLeft()  
//right rotation of the tree  
rotateRight()  
  
}  
int inversionCount(int A[], int size)  
{  
    for(int i = 0 to size-1)  
    {  
        avlTree avl_tree  
        avl_tree.insert(A[i], inversion_count)  
    }  
  
    return inversion_count  
  
}
```

Complexity Analysis

Time Complexity: $O(N \log N)$

Space Complexity: $O(N)$

Critical ideas to think!

- Can you think of other data structures that can be used to solve this problem? How good you are with trees? Have you heard about Binary Indexed Trees(BIT)? Comment below.

Comparison Table for Different Solutions

Approach	Time Complexity	Space Complexity
Brute Force	$O(N^2)$	$O(1)$
Using Divide and	$O(N \log N)$	$O(N)$

Using Divide and Conquer	$O(N \log N)$	$O(N^2)$
Using Ordered Set	$O(N^2)$	$O(N)$
Using AVL Tree	$O(N \log N)$	$O(N)$

Suggested Problems to Solve

 SHARE ON FACEBOOK

 SHARE ON TWITTER


- Merge Sort with constant space complexity for merge

 SHARE ON LINKEDIN

 SHARE ON TELEGRAM

- Count of greater elements for each element in the given array.

 SHARE ON REDDIT

 SHARE ON WHATSAPP

- Sort an array using Merge Sort.

Happy Coding!

Team AfterAcademy!



Average of levels of Binary tree

afteracademy.com

Average of Levels in Binary Tree



Admin AfterAcademy
4 Nov 2020
Given a binary tree, write a program to return the average value of the nodes on each level

Interview question

LRU Cache implementation

k1 k2 k3 k4

N1

N2

N3

N4

Asked in    

afteracademy.com

LRU Cache Implementation

Design and implement a data structure for LRU Recently Used (LRU) cache. Your data structure must support two operations:



Admin AfterAcademy
12 Oct 2020


in the form of an array. The range of the node's value is in the range of 32-bit signed integer.

get(key) and put(). The problem expects a constant time solution

Interview question

Letter Combinations of a Phone Number

1	2	3
4	5	6
7	8	9
*	0	#

Asked in 

afteracademy.com

Letter Combinations of a Phone Number

Given a string str, containing digits from 2 - 9 inclusive, write a program to return all the possible letter combinations that the number could represent. This is a popular coding interview question based on backtracking and recursive approach.



Admin AfterAcademy
11 Oct 2020

1	4
2	3
3	2
4	1

Reverse a Stack Using Recursion

afteracademy.com

Reverse a Stack Using Recursion

Given a stack of integers st, write a program to reverse the stack using recursion. This problem will clear the concept of recursion.



Admin AfterAcademy
5 Oct 2020

Interview question

Interleaving String

ABMN
AMBN
AMNB

Asked in   

afteracademy.com




Admin AfterAcademy
17 Jul 2020
Given three strings S1, S2 and S3, write a

Interview question

Surrounded regions

X X O O X O
O X O X X X

Asked in 

afteracademy.com



Admin AfterAcademy
22 Sep 2020
Given a 2-D matrix board where every

program which checks whether S3 is an interleaving of S1 and S2. The problem is a typical dynamic programming problem.

element is either 'O' or 'X', write a program to replace 'O' with 'X' if surrounded by 'X'. It is a famous problem based on the concept of flood fill algorithms

Our Learners Work At



AfterAcademy

Stay up to date. Follow us on



© Copyright 2019

MindOrks Nextgen Private Limited
Gurgaon, Haryana, India
+91-8287460223

About Us

MindOrks
Amit Shekhar
Janishar Ali

Quick Links

[Contact Us](#)
[Privacy Policy](#)
[Terms And Conditions](#)
[Cookie Policy](#)

Free Resources

[Publication](#)
[Medium](#)
[Video Lessons](#)
[Open Source](#)