- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Count Inversions in an array

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

Example:
The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

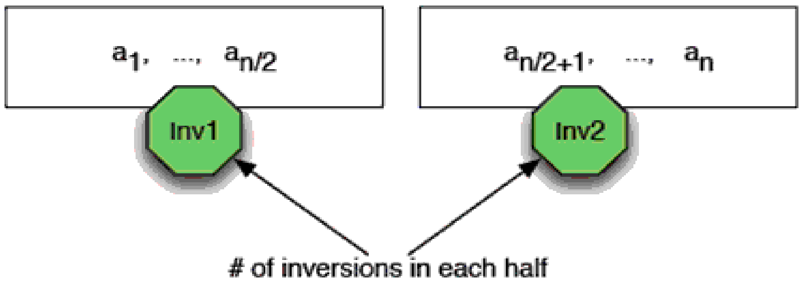
METHOD 1 (Simple)
For each element, count number of elements which are on right side of it and are smaller than it.

```
int getInvCount(int arr[], int n)
```

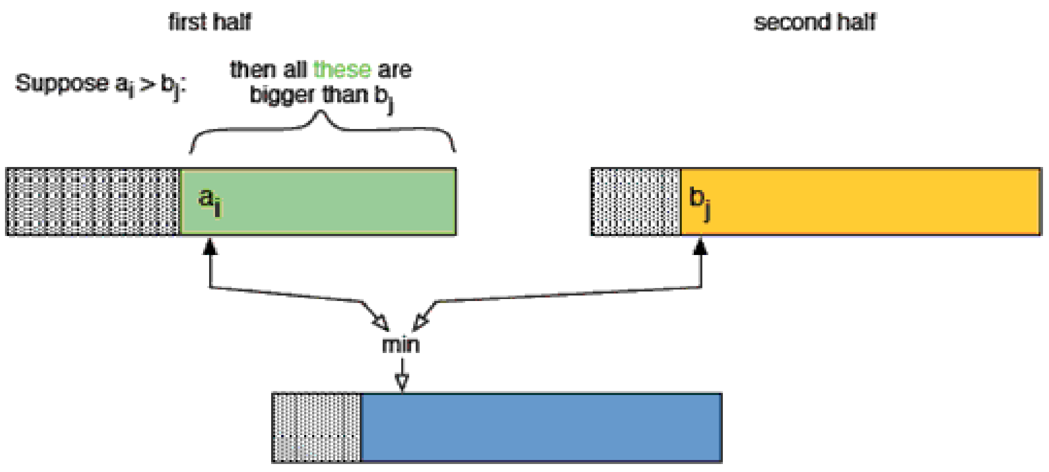
```
{
  int inv_count = 0;
  int i, j;

  for(i = 0; i < n - 1; i++)
    for(j = i+1; j < n; j++)
      if(arr[i] > arr[j])
        inv_count++;

  return inv_count;
}

/* Driver progra to test above functions */
int main(int argv, char** args)
{
  int arr[] = {1, 20, 6, 4, 5};
  printf(" Number of inversions are %d \n", getInvCount(arr, 5));
  getchar();
  return 0;
}
```

Time Complexity: O(n^2)

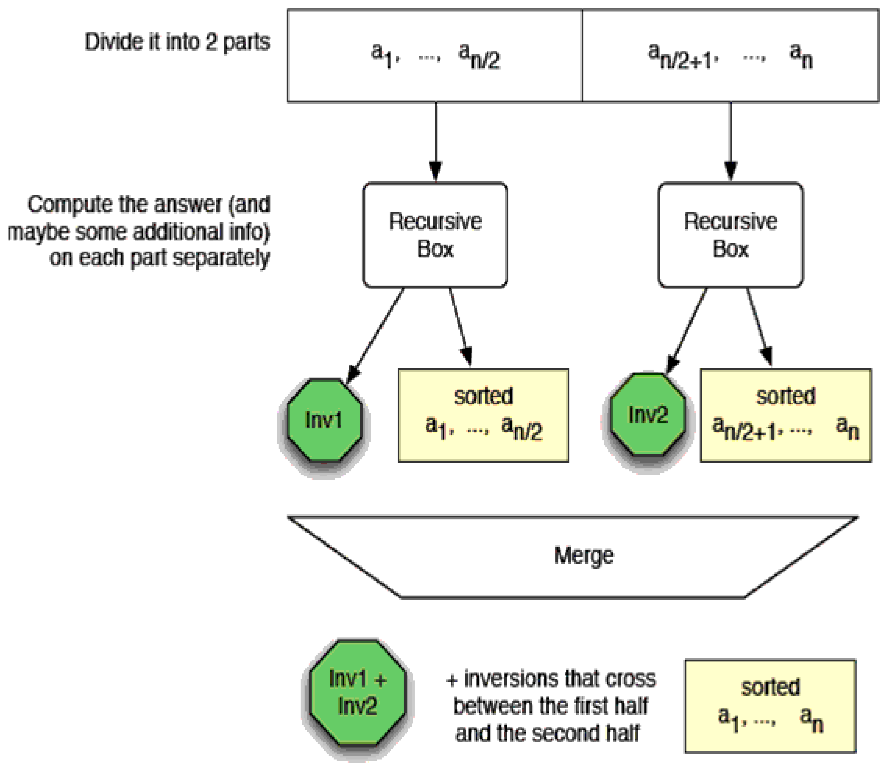
METHOD 2(Enhance Merge Sort)
Suppose we know the number of inversions in the left half and right half of the array (let be inv1 and inv2), what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is – the inversions we have to count during the merge step. Therefore, to get number of inversions, we need to add number of inversions in left subarray, right subarray and merge().



How to get number of inversions in merge()?
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]



The complete picture:


Divide it into 2 parts
$a_1, \ldots, a_{n/2}$
$a_{n/2+1}, \ldots, a_n$
Compute the answer (and maybe some additional info) on each part separately
Recursive Box
Recursive Box
Inv1
sorted $a_1, \ldots, a_{n/2}$
Inv2
sorted $a_{n/2+1}, \ldots, a_n$
Merge
Inv1 + Inv2
+ inversions that cross between the first half and the second half
sorted $a_1, \ldots, a_n$

Implementation:

```
#include <stdio.h>
#include <stdlib.h>

int  _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);

/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int *temp = (int *)malloc(sizeof(int)*array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}

/* An auxiliary recursive function that sorts the input array and
  returns the number of inversions in the array. */
int _mergeSort(int arr[], int temp[], int left, int right)
{
  int mid, inv_count = 0;
  if (right > left)
  {
    /* Divide the array into two parts and call _mergeSortAndCountInv()
       for each of the parts */
    mid = (right + left)/2;

    /* Inversion count will be sum of inversions in left-part, right-part
      and number of inversions in merging */
    inv_count  = _mergeSort(arr, temp, left, mid);
    inv_count += _mergeSort(arr, temp, mid+1, right);

    /*Merge the two parts*/
    inv_count += merge(arr, temp, left, mid+1, right);
  }
  return inv_count;
}

/* This funt merges two sorted arrays and returns inversion count in
   the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
{
  int i, j, k;
  int inv_count = 0;

  i = left; /* i is index for left subarray*/
  j = mid;  /* i is index for right subarray*/
  k = left; /* i is index for resultant merged subarray*/
  while ((i <= mid - 1) && (j <= right))
  {
    if (arr[i] <= arr[j])
    {
      temp[k++] = arr[i++];
    }
    else
    {
      temp[k++] = arr[j++];

     /*this is tricky -- see above explanation/diagram for merge()*/
      inv_count = inv_count + (mid - i);
    }
  }

  /* Copy the remaining elements of left subarray
   (if there are any) to temp*/
  while (i <= mid - 1)
    temp[k++] = arr[i++];

  /* Copy the remaining elements of right subarray
   (if there are any) to temp*/
  while (j <= right)
    temp[k++] = arr[j++];
  /*Copy back the merged elements to original array*/
  for (i=left; i <= right; i++)
    arr[i] = temp[i];

  return inv_count;
}

/* Driver progra to test above functions */
int main(int argv, char** args)
{
  int arr[] = {1, 20, 6, 4, 5};
  printf(" Number of inversions are %d \n", mergeSort(arr, 5));
  getchar();
  return 0;
}
```

Note that above code modifies (or sorts) the input array. If we want to count only inversions then we need to create a copy of original array and call mergeSort() on copy.

Time Complexity: O(nlogn)
Algorithmic Paradigm: Divide and Conquer

References:
http://www.cs.umd.edu/class/fall2009/cmsc451/lectures/Lec08-inversions.pdf
http://www.cp.eng.chula.ac.th/~piak/teaching/algo/algo2008/count-inv.htm

Please write comments if you find any bug in the above program/algorithm or other ways to solve the same problem.

Related Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Tags: Divide and Conquer

Tweet g+1 6

Writing code in comment? Please use **ideone.com** and share the link here.

112 Comments GeeksforGeeks 1 Login

Recommend 2 Share Sort by Newest

Join the discussion

pauline • 21 days ago

how can i Write a function which shifts the elements of an array (of integers) by k positions. For example, if k=5 and the array is [1 2 3 4 5 6 7 8 9] then after we callt the function on the array and k, the array will be [6 7 8 9 1 2 3 4 5].

• Reply • Share ›

Sumeet Varma pauline • 17 days ago

Naively shifting the array always would cause O(n) always.
Here is an O(1) solution per each shift with just addition and modulus.

u don't actually need to shift the array.. just keep a head pointer initially at 0 position. Now say u want to shift everything by 5 position by left,then head = (head + 5) % n because our head would move 5 position by right then. Also think of array as a circular array so that whenever head exceeds n (right end of array) u wrap it around from 0 (left end of array)

also in general if after shifting u want to access say i'th element of new array.
then int element_at_position_i = a[(i+ head) % n]

if u want to loop across the array.just start looping from i = head to i = head + n. and access element as a[i%n]

1 • Reply • Share ›

Ashish Maheshwari • a month ago

In the paragraph..
How to get number of inversions in merge()?
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if "a[i] is greater than a[j]", then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]

There is a typo. it should be "a[i] is greater than b[j]". Please correct it :)

• Reply • Share ›

Mahesh Kasana • a month ago

Inversion finding is very easy by tree implementation..........

first of all make "invers" as a global or class member

and after complete tree creation print "inver".

void array :: inserttreedata(int data)

{

```
tree *p,*q;

p=new tree;

p->data=data;

p->count=1;

p->left=p->right=NULL;

if(root==NULL)
```

see more

^ | v • Reply • Share ›

Ryan Sun • 3 months ago

I think here should be (mid - i + 1), if(arr[j] < arr[i])

```
else
{
temp[k++] = arr[j++];

/*this is tricky -- see above explanation/diagram for merge()*/
inv_count = inv_count + (mid - i);
```

^ | v • Reply • Share ›

mudit ➦ Ryan Sun • 8 days ago

That's perfect. Here mid is actually mid +1. See following line of code:
/*Merge the two parts*/
inv_count += merge(arr, temp, left, mid+1, right);

^ | v • Reply • Share ›

Aadil • 3 months ago

What if I want to count inversions for triplets?

^ | v • Reply • Share ›

Mr.Wonderful • 3 months ago

comments in the code need to be updated.

^ | v • Reply • Share ›

Shiva Attry • 3 months ago

Here's a c++ code to calculate number of inversion, working for several cases, not much sure though. Any suggestion?

http://ideone.com/MWeQpx

• Reply • Share ›

Fernando Ferreira • 3 months ago

This is my implementation in C. It's a very clean code and easier to understand.

http://ideone.com/Z471KB

1 • Reply • Share ›

Aditya Goel • 3 months ago

without using temp (done change in G2G Standard Merge Sort code)

http://ideone.com/XM7xFc

• Reply • Share ›

user • 3 months ago

A python Implementation:
http://ideone.com/XbQmEL

1 • Reply • Share ›

Guest • 3 months ago

whats wrong with input {6, 3, 8, 2, 5, 1, 9}.
Program's output for it is 6, it should be 11.
Please point out the bug

2 • Reply • Share ›

Aditya Goel Guest • 3 months ago

Yes, u r right, something is breaking
It is giving correct o/p in my code-
http://ideone.com/XM7xFc

• Reply • Share ›

TJ • 3 months ago

what should be inversion count for {5,4,2,6,7}, ideally only swapping 5 and 2 will do..so it should be 1 but above methods will return more than 1.

• Reply • Share ›

user TJ • 3 months ago

Thats fine, you can sort this array in one swap, but not using Merge Sort. By doing some paper work you will get the clear picture. For inversions go again with problem definition. In this array, 3 inversions are there: (5,4), (5,2) and (4,2)

• Reply • Share ›

sid • 3 months ago

We can also count no of inversions by using the no of steps which are being executed in the insertion sort like this :-

```
int numInversions(int nArray[], int nLength)
//
int nInversions <- 0
for int i <- 0 to nLength - 1
do
j <- i - 1
while j is greater than or equal to 0 and nArray[j] > nArray[j + 1]
do
swap nArray[j] and nArray[j + 1]
nInversions++

return nInversions
```

Reply • Share ›

Jasdeep Singh sid • 3 months ago

your soln has O(n^2) complexity for n>10^6 it would definitely give tle

Reply • Share ›

neer1304 • 4 months ago

http://ideone.com/ds5zIu

Reply • Share ›

shiva krishna • 4 months ago

One question, isn't the time complexity of method one, O(n!) ?

Reply • Share ›

Ekta Goel shiva krishna • 3 months ago

N0, count the number of comparisons, for the first iteration, (n-1) for second (n-2) for third, (n-3) and so on till i=n-1. This is basically the summation of (n-i-1) for all i from 0 to n-1. This comes out to be O(n^2).

Reply • Share ›

Bill Gates • 4 months ago

Nice post.

Reply • Share ›

Guest • 4 months ago

I understand that the most inversions happen when the array is reversed, but is it possible to determine this number (using some sort of formula)?

Reply • Share ›

Boga Guest • 2 months ago

if the array is reversed, then every number is larger than all the numbers that are to the right side of the number. So basically it becomes

(n-1) + (n-2) + (n-3) + ... + 1 + 0 = n*(n+1)/2 - n = n(n-1)/2

• Reply • Share ›

Peter Fenwick • 5 months ago

BIT

2 • Reply • Share ›

Deepak Rishi • 6 months ago

One can also solve this question by tweaking binary search trees. In each node keep a tab on the number of nodes that are to the right of the node. when you are inserting a node keep in mind the following : if you go left add 1 and the number of nodes to the right of the node from which you went left. If you go right , simply increment the number of nodes to the right of that node by one.
However, if you have a large dataset(especially if your tree is not complete) , you might run into the problem of memory becoming full.

2 • Reply • Share ›

Amit Handa • 6 months ago

Anything wrong with using insertion sort.

```
int findInversions( int[] arr, int n ) {

int invCount = 0;

for( int i = 1; i < n; i++ ) {

int ele = arr[i];

int j;

for( j = i-1; j > -1; j++ ) {

if( arr[j] > ele ) {

arr[i] = arr[j];

invCount++;

} else break;
}

arr[j] = ele;

}
```

return invCount;
}
edit : its n2.

1 ^ | v • Reply • Share ›

Guest • 8 months ago

@GeeksforGeeks the 2nd method is having some problem , check for input {8,7,6,5,4,3,2,1} giving output 10, while it should be 28

3 ^ | v • Reply • Share ›

Anurag Singh ➦ Guest • 6 months ago

You would have done following:
int arr[] = {8,7,6,5,4,3,2,1};
printf(" Number of inversions are %d \n", mergeSort(arr, 5));
This gives answer 10

In mergeSort call, change 5 to 8, as your array size is 8 (Not 5)

Correct line is:
printf(" Number of inversions are %d \n", mergeSort(arr, 8));
This gives answer 28

^ | v • Reply • Share ›

Rahul ➦ Guest • 7 months ago

change your array size to 8 from 5. It will work.

1 ^ | v • Reply • Share ›

MM ➦ Rahul • 7 months ago

So what ?

^ | v • Reply • Share ›

helper bro • 9 months ago

we can use binary search tree for this perpose

2 ^ | v • Reply • Share ›

Kim Jong-il ➦ helper bro • 8 months ago

It would be Better, you have written some line to support your claim.

3 ^ | v • Reply • Share ›

tarun • 9 months ago

@GeeksforGeeks: shouldn't all the element left in the left subarray in merge() be greater than every element in the right sub-array. That should also be added to inv_cnt, like this-
while (i <= mid - 1){
temp[k++] = arr[i++];

if(a[i] > a[j-1]) // This check is to avoid counting equal nos
inv_cnt += right-mid; // because this i is greater than all j
}

Reply • Share ›

jayasurya_j tarun • 3 months ago

I have the same doubt

Reply • Share ›

Guest • 9 months ago

<script src="http://ideone.com/e.js/o3VIsE" type="text/javascript"></script>

Reply • Share ›

Learner • 9 months ago

Inversion count will be sum of inversions in left-part, right-part

and number of inversions in merging

should the line given in code

inv_count = _mergeSort(arr, temp, left, mid);

Actually be

inv_count += _mergeSort(arr, temp, left, mid);

Reply • Share ›

Udit • 10 months ago

because left and right subarrays are sorted, so all the remaining
elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than
a[j]....

at last it should be b[j] instead of a[j]

1 • Reply • Share ›

retard • 10 months ago

try YODANESS @spoj

Reply • Share ›

Ayush Jain • 10 months ago

You can refer to my solution here in C and there is also a Java solution in the link below(by Marek Kirejczyk) which is much simpler and shorter than the above code
The logic is same but code is presented well and efficiently for understanding purpose

http://stackoverflow.com/a/245...

3 • Reply • Share ›

Praveen Reddy • 10 months ago

Actually by using binary search tree we can know the number of inversions inserting an element if the element is less than root then we can easily count the number of nodes which are to the right side of the tree including the root or we can maintain an extra variable in the struct such that how many nodes are on its right and how many nodes are on its left

I think this will give O(logn) solution. found any different cases then comment and let me know please.. :)

• Reply • Share ›

ryan Praveen Reddy • 8 months ago

no u can't ,try with 8,9,10,5,4,1,2,12

1 • Reply • Share ›

Ayush Jain Praveen Reddy • 10 months ago

This will be O(n^2) as in the worst case you have to scan the entire right subtree
For. e.g arr[]={8,7,6,5,4,3,2,1}
When you draw BST you have to scan all n-1 elements to increase their inversions which will result in O(n^2)

• Reply • Share ›

kumar Saurabh • a year ago

We can simply sort the array in O(nlogn) and finally compare the original array with the sorted one. Count the number of places where the elements are different. The count will be inversion count.

• Reply • Share ›

tweety kumar Saurabh • a year ago

no this method doesnt give inversion count. for the same example
2,4,1,3,5 sorted array is
1,2,3,4,5
according to ur method inversion count should be 4 but it is 3

1 • Reply • Share ›

koder tweety • 4 months ago

For each element 'x' in the original array, find it's index in the sorted array, and add the difference of the indices. Note that this way we count twice each pair, so we just have to divide the final number by 2 to get the correct solution.

In the example above:

- 2: old_index = 0, new_index = 1, num_pairs = 1

- 4: old_index = 1, new_index = 3, num_pairs = 2
- 1: old_index = 2, new_index = 0, num_pairs = 2
- 3: old_index = 3, new_index = 2, num_pairs = 1
- 5: old_index = 4, new_index = 4, num_pairs = 0

- total = (1 + 2 + 2 + 1 + 0)/2 = 3

• Reply • Share ›

rishabhjoshi tweety • 10 months ago

as per me the method by kumar saurabh can be corrected ,here it is giving 4 because he is not seeing the condition that i<j so="" here="" we="" should="" see="" that="" too.="" we="" can="" create="" a="" structure="" with="" first="" element="" as="" array="" element="" and="" second="" element="" as="" the="" array="" index="" .="" sort="" it="" as="" per="" elements="" and="" do="" check="" the="" condition="" of="" i<j="" while="" count++="" of="" inversion.="">

• Reply • Share ›

tweety rishabhjoshi • 10 months ago

can u plz explain your algorithm in detail...??
what i have got from the above explanation is that u are first sorting your linked list on the basis of element's value.. after that u have to check index..how you are checking index and what is stored in i and j..???

• Reply • Share ›

Zombie! • a year ago

Clean and elegant :

```
#include<iostream>

using namespace std;

// 1 array and count the no. of inversions
// Approached using mergesort

int merge(int * array,int low,int high,int mid)
{

int * newarr = new int[high-low+1];
int p1 = low;
int p2 = mid+1;
int c=0;
int z=0;

while(p1<=mid && p2<=high)
```

see more

1 · Reply · Share ›

saanvi Zombie! · a year ago

very nice :) thank you

· Reply · Share ›

Load more comments

Subscribe Add Disqus to your site Privacy DISQUS

Google™ Custom Search

-
-
-
-
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms

- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

Recent Comments

- Ashish Aggarwal

  Try Data Structures and Algorithms Made Easy -...

  Algorithms · 17 minutes ago

- Vlad

  Thanks. Very interesting lectures.

  Expected Number of Trials until Success · 1 hour ago

- cfh

  My implementation which prints the index of the...

  Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- Gaurav pruthi

  forgot to see that part ;)

  Bloomberg Interview | Set 1 (Phone Interview) · 1 hour ago

- saeid aslami

  thanks

  Greedy Algorithms | Set 7 (Dijkstra’s shortest path algorithm) · 1 hour ago

- Cracker

  Implementation:...

  Implement Stack using Queues · 2 hours ago

-