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.

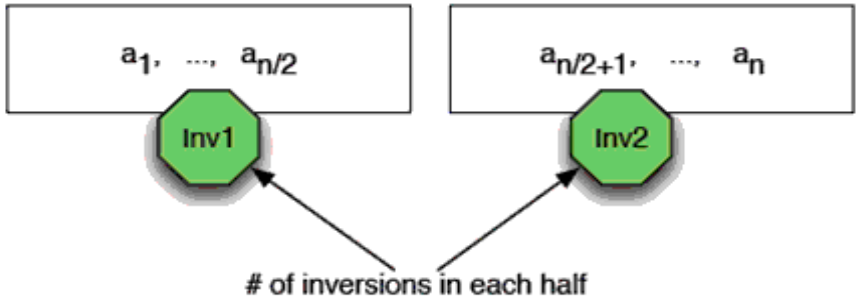
```
int getInvCount(int arr[], int n)
{
  int inv_count = 0;
  int i, j;

  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;
```


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

}

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().

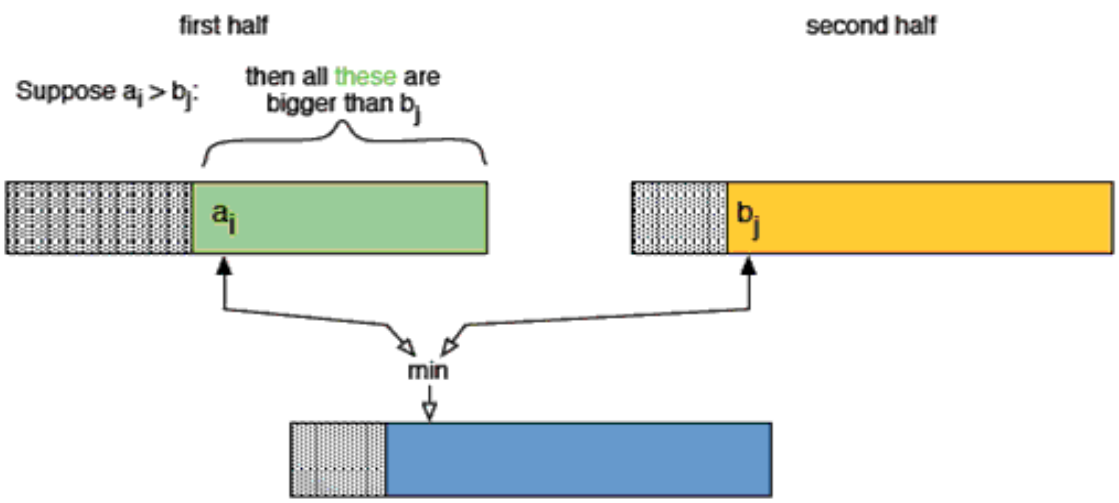


How to get number of inversions in merge()?
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray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

Geometric Algorithms

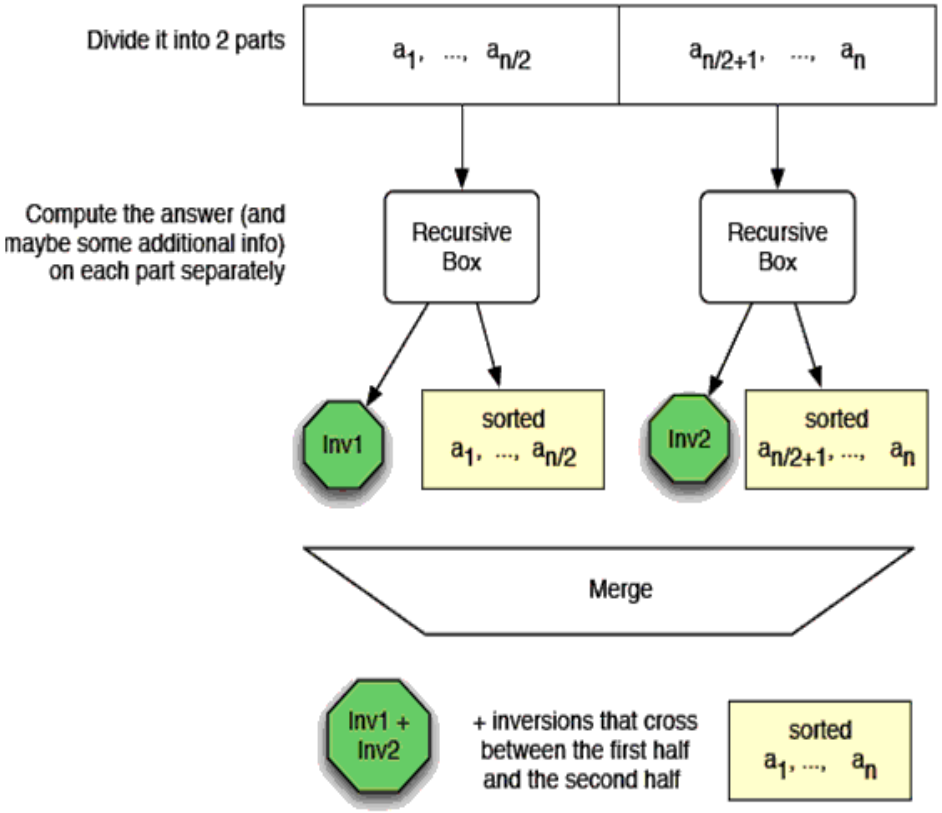
Popular Posts

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



The complete picture:

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



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
```


Recent Comments

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 22 minutes ago

```
  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-p
      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
```

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 25 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 50 minutes ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 52 minutes ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

newCoder3006 Code without using while loop. We can do it...

Find subarray with given sum · 1 hour ago

```
   (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 Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

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

60 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

Paramvir Singh • 2 months ago

everybody who is facing with the doubt that inversions should be mid-1+j is co
function argument passed is mid+1 not mid. Hope this helps.

• Reply • Share ›

Rahul • 5 months ago

Its fine! Sorry

1 • Reply • Share ›

Rahul • 5 months ago

It should be j=mid+1

• Reply • Share ›

Rahul • 5 months ago

I am in a doubt..there should be mid-i+1 inversions.

• Reply • Share ›

Rahul → Rahul • 5 months ago

or j should point to mid+1????

• Reply • Share ›

alam01 • 5 months ago

If we just need the inversion count then what is the need of array 'temp'?
Do we need it?

• Reply • Share ›

Akshay Srinivas • 5 months ago

i wrote following algorithm, let me know if its good one
#include<stdio.h>

```
static int start_address = 0;
static int size = 0;
int inc = 0;
int inversion(int *arr, int num, int n)
{
if(n == 0) {
if(num > arr[0]) {
return 1;
}
return 0;
}
inc += inversion(arr, num, n/2);
if(n %2 != 0 && n > 1) {
if(num > arr[n-1]) {
inc++;
}
```

see more

Reply • Share ›

feroz • 6 months ago

how can i do method one in 2d array c#

Reply • Share ›

tczf1128 • 6 months ago

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

Reply • Share ›

tczf1128 → tczf1128 • 6 months ago

you are right.sorry

Reply • Share ›

We can solve this in O(n) using a stack.

• Reply • Share ›

Upen ➦ Murali • 6 months ago

we can solve it by using stack but can't in O(n) time it will cost us O(n^
really think we can solve it in O(n) using stack

1 • Reply • Share ›

kd111 • 7 months ago

//Simple modification to mergeSort algorithm

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

int count = 0;

void mergeAndCount(int *A , int low , int mid , int high){
int n1 = mid - low + 1;
int n2 = high - low;
int *L = (int *) malloc(sizeof(int)*n1);
int *R = (int *) malloc(sizeof(int)*n2);
int i , j , k;
for(i = 0 ; i < n1 ; i++)
L[i] = A[low + i];
for(j = 0 ; j < n2 ; j++)
R[j] = A[mid + 1 + j];
i = 0;
i = 0;
```

see more

1 • Reply • Share ›

@geeksforgeeks although not optimal..is the following progm correct to count sort..just to clarify the concept..

```
#include<stdio.h>

int bubble(int arr[],int n)
{
int i, j,temp,k,inv_count=0;

for(i = 0; i <= n - 2; i++)
for(j = 0; j <= n-i-2; j++)
if(arr[j] > arr[j+1])
{
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
inv_count++;
}
return inv_count;
```

see more

• Reply • Share ›

Guest • 8 months ago

@GeeksforGeeks although not optimal..is the following progm correct to cou bubble sort..just to clarify the concept..

```
#include<stdio.h>

int bubble(int arr[],int n)
{
int i, j,temp,k,inv_count=0;
```

```
{for(j = 0; j <= n-i-2; j++)
{if(arr[j] > arr[j+1])
{
printf("\n%d\t%d\n",arr[j],arr[j+1]);
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
inv_count++;
// for(k=0;k<n;k++) printf("%d",arr[k]);="" }="" }="" printf("\n\n");="" for(k="0;k&lt;
printf("%d\t",arr[k]);printf("\n\n");="" }="" return="" inv_count;="" }="" void="" ma
myarray[]="{50,40,30,20,10};" count_inv="bubble(myarray,5);" for(k="0;k&lt;5;k
%d\n",myarray[k]);="" printf("the="" number="" of="" inversions="" is="" %d",cc
```

• Reply • Share ›

Mohit Garg • 10 months ago

I think there exists a simpler solution
sort the array
e.g. 4,5,6,1,2,3 becomes 1,2,3,4,5,6
find the displacement for a given element e.g. 4 which was initially at 0 is now

Total number of inversions should be sum of all the displacement towards righ
Only 4,5,6 are displaced right, total = 3+3+3 = 9

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

piyush ➦ Mohit Garg • 9 months ago

Try using the same technique on 4,5,6,1,3,2:
Your answer would still be 3+3+3=9, however the correct answer is 1(

Its the not displacement towards right that counts, but the relative displ

to 2 is swapped => add 1, and so on .

1 • Reply • Share ›

crazy • 11 months ago

```
 #include<stdio.h>
#define INF 19999999
long long total;
void merge(int a[],int p,int q,int r)
{
    int n1,n2,i,k,j;
    n1=(q-p)+1;
    n2=(r-q);
    int left[n1+2],right[n2+2];
    for(i=1;i<=n1;i++)
     left[i]=a[p+i-1];
    for(i=1;i<=n2;i++)
     right[i]=a[q+i];
    left[n1+1]=right[n2+1]=INF;
    i=j=1;
    for(k=p;k<=r;k++)
    {
        if(left[i]<=right[j])
```

see more

4 • Reply • Share ›

Salman Cheema • a year ago

OOOOOOOO

• Reply • Share ›

Venkatesh B • a year ago

for the algorithm given by geeks for geeks, for this input 4,5,6,1,2,3 number of

• Reply • Share ›

Venkatesh Fan → Venkatesh B • 4 months ago

Are u the famous Venkatesh B?

• Reply • Share ›

ljk → Venkatesh B • 7 months ago

Is this venkatesh basker?

• Reply • Share ›

Swapnil R Mehta → Venkatesh B • a year ago

Yes its correct.
As inversions: (4,1),(4,2),(4,3),(5,1),(5,2),(5,3),(6,1)(6,2),(6,3).

• Reply • Share ›

shivi • a year ago

```
 #include <algorithm>
#include <cstdio>
#include<shiviheaders.h>
#include <cstring>
using namespace std;

typedef long long llong;
const int MAXN = 500020;
llong tree[MAXN], A[MAXN], B[MAXN];

llong read(int idx)
{
    llong sum = 0;
    while (idx > 0)
    {
        sum += tree[idx];
```

```
        idx -= (idx & -idx);
    }
```

see more

Reply • Share ›

ajiteshpathak • a year ago

Not sure if any of the above methods have similar implementation but here is r
where I holds the iterator for every element in the array and J just iterates all el

```
int inversionCount(int *arr, int n)
{
int i = 0, j = 1;
int count = 0;

while (i < n - 1)
{
if (arr[j] > arr[i] && j > i)
{
// Already sorted
j++;
}
else if (arr[j] < arr[i] && j > i)
{
printf(" (%d, %d) ", arr[i], arr[j]);
```

see more

1 Reply • Share ›

Nilesh J Choudhary • a year ago

nice

Reply • Share ›

lotus • a year ago

Why can't we just store sorted array and count how many numbers in the orig
position.

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

dew ➦ lotus • a year ago

@GeeksforGeeks Please let us know if there is any mistake in this logi
elements are sorted. then finding the no of elements that are not in thei

• Reply • Share ›

Priso ➦ dew • a year ago

Consider an sorted array (1,2,3,4) where inversion is 0
now lets swap 1 and 4 so the array is (4,2,3,1)
and the numbers which are not in their expected positions = 2 (

But the number of inversions = 5 i.e., {(4,2),(4,3),(4,1),(2,1),(3,1

```
/* Paste your code here (You may delete these lines if
```

• Reply • Share ›

mukesh gupta • a year ago

```
void inversion(int a[ ],int n)
{
if(n>1){
int b[n/2],c[n-n/2],i,j=0,k,m;
for(i=0;i<n/2;i++)
b[i]=a[i];
```

```
{ c[j]=a[i];
j++;}
inversion(b,n/2);
inversion(c,n-n/2);
i=0;
j=0;
k=0;
while(i<n/2 && j<(n-n/2))
{ if(b[i]<c[j])
{ a[k]=b[i];
i++;}
```

see more

Reply • Share ›

mukesh gupta • a year ago

```
void inversion(int a[ ],int n)
{
if(n>1){
int b[n/2],c[n-n/2],i,j=0,k,m;
for(i=0;i<n/2;i++)
b[i]=a[i];
for(i=n/2;i<n;i++)
{ c[j]=a[i];
j++;}
inversion(b,n/2);
%2
```

Reply • Share ›

bartender • a year ago

If mid = (left+right)/2

As all elements from A[i] to A[mid] are less than A[j]
which is mid+1-i.

But the code works out as we pass mid+1 into the merge function.So, here m
code, pointing to the first element in the second sub-array.

• Reply • Share ›

coderAce ➦ bartender • a year ago

You're right. @Moderators, you should highlight this in the main article.

• Reply • Share ›

Ankit Malhotra • a year ago

Earlier code was using extra memory. However using array rotation, this can b

```
#include <iostream>
using namespace::std;
typedef unsigned long counter;
typedef long long element;

inline void rotate (element term[], counter count, counter jump)
{
  element temp;
  counter gcd = count, k = jump, i, position;
  for (; k != 0; i = k, k = gcd % k, gcd = i);
  for (i = 0; i != gcd; ++i)
  {
    temp = term [i];
    position = k = i;
    do
```

see more

• Reply • Share ›

Ankit Malhotra ➦ Ankit Malhotra • a year ago

As we are using unsigned for counter which is generally what size_typ
need to be done.

```
k -= jump;
if (k < 0) k += count;
```

should be

```
k += jump;
if (k >= count) k -= count;
```

```
rotate (term + left, range + movecount, movecount);
```

should be

```
rotate (term + left, range + movecount, range);
```

• Reply • Share ›

Arun • a year ago

You need to change

inv_count += mid - i to
inv_count += (mid-left+1-i);

else, the result coming is not correct.

Thanks,

• Reply • Share ›

bartender ➦ Arun • a year ago

We are passing mid+1 into merge routine,so everything works out.Rea
details.

• Reply • Share ›

GeeksforGeeks ➦ Arun • a year ago

@Arun: Please take a closer look at the code. The value of mid is (left
case for which the given code doesn't produce the correct result.

1 • Reply • Share ›

Ankit Malhotra • a year ago

Simplified the code to a single merge sort function which returns the inversion
reduces looping when consecutive elements on the right side need me be mo

```
 using namespace::std;
typedef unsigned long counter;
typedef long long element;
counter mergesort (element term[], counter count)
{
  if (count < 2) return 0;
  counter mid = count/2,
  inversions = mergesort (term, mid);
  inversions += mergesort (term + mid, count - mid);
  counter prefix = 0, suffix = mid, range, movecount, bound, j;
  element * ptr = NULL;
  while (true)
  {
    for (; prefix != suffix && term[prefix] <= term[suffix]; prefix++
    range = suffix - prefix;
```

see more

• Reply • Share ›

Ankit Malhotra → Ankit Malhotra • a year ago

if (suffix == count) break;
should also have else statement as follows
else prefix += movecount;

• Reply • Share ›

Ankit Malhotra → Ankit Malhotra • a year ago

delete ptr should be replaced with delete []ptr

• Reply • Share ›

pankaj • a year ago

balanced BST

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

hi • 2 years ago

This problem can also be solved using BIT and Segment Tree

• Reply • Share ›

abzx12@gmail.com → hi • a year ago

Can you suggest the way we can use BIT to solve it?

```
/* Paste your code here (You may delete these lines if not wri
```

• Reply • Share ›

Rahul → abzx12@gmail.com • 11 months ago

There you go

#include

```
#include
#include

using namespace std;

typedef long long llong;

const int MAXN = 500020;
llong tree[MAXN], A[MAXN], B[MAXN];

llong read(int idx){
llong sum = 0;
while (idx > 0){
sum += tree[idx];
idx -= (idx & -idx);
}
```

see more

• Reply • Share ›

Venkatesh • 2 years ago

If we use modified insertion sort alg, we see best running time. for partially sor
time.

```
int arr[], array_size, inv_count;

for (int i = 0; i 0; j--)
if (a[j] < a[j-1]))
inv_count ++;
else
break;
```

• Reply • Share ›

jordi • 3 years ago

theres a linear solution that im looking for..

• Reply • Share ›

Algoseekar ➦ jordi • 3 years ago

@jordi..i dont think we can do it linearly..??

• Reply • Share ›

amit • 3 years ago

nice question !

• Reply • Share ›

Naman Goel • 3 years ago

We can also use binary search tree method for this
here is the code-

```
#include<iostream>
using namespace std;

struct node{
        int data;
        node *left;
        node *right;
        int rc;
};

node *root = NULL;

int get_inv(int val)
{
```

see more

Reply • Share ›

Load more comments

Subscribe Add Disqus to your site