Count smaller elements on right side

Write a function to count number of smaller elements on right of each element in an array. Given an unsorted array arr[] of distinct integers, construct another array countSmaller[] such that countSmaller[i] contains count of smaller elements on right side of each element arr[i] in array.

Examples:

```
Input:   arr[] =  {12, 1, 2, 3, 0, 11, 4}
Output:  countSmaller[]  =  {6, 1, 1, 1, 0, 1, 0}

(Corner Cases)
Input:   arr[] =  {5, 4, 3, 2, 1}
Output:  countSmaller[]  =  {4, 3, 2, 1, 0}

Input:   arr[] =  {1, 2, 3, 4, 5}
Output:  countSmaller[]  =  {0, 0, 0, 0, 0}
```

Method 1 (Simple)

Use two loops. The outer loop picks all elements from left to right. The inner loop iterates through all the elements on right side of the picked element and updates countSmaller[].

```
void constructLowerArray (int *arr[], int *countSmaller, int n)
{
  int i, j;

  // initialize all the counts in countSmaller array as 0
  for  (i = 0; i < n; i++)
     countSmaller[i] = 0;

  for (i = 0; i < n; i++)
  {
    for (j = i+1; j < n; j++)
    {
       if (arr[j] < arr[i])
         countSmaller[i]++;
    }
  }
}

/* Utility function that prints out an array on a line */
void printArray(int arr[], int size)
```

```
{
  int i;
  for (i=0; i < size; i++)
    printf("%d ", arr[i]);

  printf("\n");
}

// Driver program to test above functions
int main()
{
  int arr[] = {12, 10, 5, 4, 2, 20, 6, 1, 0, 2};
  int n = sizeof(arr)/sizeof(arr[0]);
  int *low = (int *)malloc(sizeof(int)*n);
  constructLowerArray(arr, low, n);
  printArray(low, n);
  return 0;
}
```

Run on IDE

Time Complexity: O(n^2)

Auxiliary Space: O(1)

Method 2 (Use Self Balancing BST)

A Self Balancing Binary Search Tree (AVL, Red Black,.. etc) can be used to get the solution in O(nLogn) time complexity. We can augment these trees so that every node N contains size the subtree rooted with N. We have used AVL tree in the following implementation.

We traverse the array from right to left and insert all elements one by one in an AVL tree. While inserting a new key in an AVL tree, we first compare the key with root. If key is greater than root, then it is greater than all the nodes in left subtree of root. So we add the size of left subtree to the count of smaller element for the key being inserted. We recursively follow the same approach for all nodes down the root.

Following is C implementation.

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

// An AVL tree node
struct node
{
    int key;
    struct node *left;
    struct node *right;
    int height;
    int size; // size of the tree rooted with this node
};

// A utility function to get maximum of two integers
int max(int a, int b);

// A utility function to get height of the tree rooted with N
int height(struct node *N)
{
    if (N == NULL)
```

```
        return 0;
    return N->height;
}

// A utility function to size of the tree of rooted with N
int size(struct node *N)
{
    if (N == NULL)
        return 0;
    return N->size;
}

// A utility function to get maximum of two integers
int max(int a, int b)
{
    return (a > b)? a : b;
}

/* Helper function that allocates a new node with the given key and
    NULL left and right pointers. */
struct node* newNode(int key)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->key   = key;
    node->left   = NULL;
    node->right  = NULL;
    node->height = 1;  // new node is initially added at leaf
    node->size = 1;
    return(node);
}

// A utility function to right rotate subtree rooted with y
struct node *rightRotate(struct node *y)
{
    struct node *x = y->left;
    struct node *T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right))+1;
    x->height = max(height(x->left), height(x->right))+1;

    // Update sizes
    y->size = size(y->left) + size(y->right) + 1;
    x->size = size(x->left) + size(x->right) + 1;

    // Return new root
    return x;
}

// A utility function to left rotate subtree rooted with x
struct node *leftRotate(struct node *x)
{
    struct node *y = x->right;
    struct node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    //  Update heights
```

```
    x->height = max(height(x->left), height(x->right))+1;
    y->height = max(height(y->left), height(y->right))+1;

    // Update sizes
    x->size = size(x->left) + size(x->right) + 1;
    y->size = size(y->left) + size(y->right) + 1;

    // Return new root
    return y;
}

// Get Balance factor of node N
int getBalance(struct node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Inserts a new key to the tree rotted with node. Also, updates *count
// to contain count of smaller elements for the new key
struct node* insert(struct node* node, int key, int *count)
{
    /* 1.  Perform the normal BST rotation */
    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left  = insert(node->left, key, count);
    else
    {
        node->right = insert(node->right, key, count);

        // UPDATE COUNT OF SMALLER ELEMENTS FOR KEY
        *count = *count + size(node->left) + 1;
    }


    /* 2. Update height and size of this ancestor node */
    node->height = max(height(node->left), height(node->right)) + 1;
    node->size   = size(node->left) + size(node->right) + 1;

    /* 3. Get the balance factor of this ancestor node to check whether
       this node became unbalanced */
    int balance = getBalance(node);

    // If this node becomes unbalanced, then there are 4 cases

    // Left Left Case
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    // Right Right Case
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    // Left Right Case
    if (balance > 1 && key > node->left->key)
    {
        node->left =  leftRotate(node->left);
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && key < node->right->key)
```

```
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    /* return the (unchanged) node pointer */
    return node;
}

// The following function updates the countSmaller array to contain count of
// smaller elements on right side.
void constructLowerArray (int arr[], int countSmaller[], int n)
{
  int i, j;
  struct node *root = NULL;

  // initialize all the counts in countSmaller array as 0
  for  (i = 0; i < n; i++)
     countSmaller[i] = 0;

  // Starting from rightmost element, insert all elements one by one in
  // an AVL tree and get the count of smaller elements
  for (i = n-1; i >= 0; i--)
  {
     root = insert(root, arr[i], &countSmaller[i]);
  }
}

/* Utility function that prints out an array on a line */
void printArray(int arr[], int size)
{
  int i;
  printf("\n");
  for (i=0; i < size; i++)
    printf("%d ", arr[i]);
}

// Driver program to test above functions
int main()
{
  int arr[] = {10, 6, 15, 20, 30, 5, 7};
  int n = sizeof(arr)/sizeof(arr[0]);

  int *low = (int *)malloc(sizeof(int)*n);

  constructLowerArray(arr, low, n);

  printf("Following is the constructed smaller count array");
  printArray(low, n);
  return 0;
}
```

Run on IDE

Output:

```
Following is the constructed smaller count array
3 1 2 2 2 0 0
```

Time Complexity: O(nLogn)

Auxiliary Space: O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

117 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3.2 Average Difficulty : **3.2/5.0**
Based on **5** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

117 Comments GeeksforGeeks Login

Recommend 1 Share Sort by Newest

Join the discussion…

amit • 11 days ago

java implementation for 2nd approach.

Instead of AVL tree map uses red black.

```
package arrays.page.three;

import java.util.Arrays;

import java.util.SortedMap;

import java.util.TreeMap;

/**

*

* http://www.geeksforgeeks.org/c...

*

* @author amishash
```

see more

• Reply • Share ›

kris • 2 months ago

instead of saving the size of the tree rooted @ N can we save the size of the lst roooted @ N

• Reply • Share ›

asen • 3 months ago

http://ideone.com/qEpxwb nlogn using set. thanks kushal. please comment for errors. 1. distance function is O(n) so it boils down to O(n^2) :/. Thanks @popeye

• Reply • Share ›

popeye ➚ asen • 3 months ago

distance function for sets is O(n), so your algorithm boils down to O(n^2). We should be careful before using STL libraries

1 • Reply • Share ›

asen ➚ popeye • 3 months ago

Thank you so much for pointing this out :)

• Reply • Share ›

Ravi • 4 months ago

1) Initialise:
n = a.length - 1;
bst with bst.put(a[n],null)
count[] array to hold result
2) for i = n-1 to 1

count[i] = bst.rank(a[i])
bst.insert(a[i])

• Reply • Share ›

QILI • 5 months ago

@geeksforgeeks

Hello, there is one possible bug in

/* 2. Update height and size of this ancestor node */
node->height = max(height(node->left), height(node->right)) + 1;
node->size = size(node->left) + size(node->right) + 1;

Should we first make balance for the node, before we calculate the height? And also for size, even though here it won't affect the result I think.

• Reply • Share ›

QILI ➚ QILI • 5 months ago

Sorry, actually, it has no problem, because in rotation functions, the height and size have been updated. So it does not matter to put these two lines before the possible rotations or not.

• Reply • Share ›

Dexter • 5 months ago

1. Create an empty array of size n, call it sorted array.
2. Scan from right. right=n-1
3. parse input array from right to left And each new element can be put into sorted array. We can do Binary search to find the right place.
4. Any given time, number of elements on the left hand side are smaller elements.

Does this work ?

• Reply • Share ›

Dexter ➚ Dexter • 5 months ago

Time: o(nlogn)
Space; o(n)

▲ | ▼ • Reply • Share ›

Jegeyom ➚ Dexter • 3 months ago

The problem here is that inserting an element into an array is not constant time.

▲ | ▼ • Reply • Share ›

coolk • 5 months ago

Java implementation. Please note, I am not using AVL tree. So worst case performance could be n^2.
Corrections or suggestions are welcome!

http://ideone.com/QtAjCa

▲ | ▼ • Reply • Share ›

Holden • 5 months ago

Can somebody please share the code for Method 2 in Java?

▲ | ▼ • Reply • Share ›

Holden • 5 months ago

Why from right to left?

> We traverse the array from right to left and
> insert all elements one by one in an AVL tree.

▲ | ▼ • Reply • Share ›

Holden • 5 months ago

There are some small typos:

```
// Inserts a new key to the tree rotted with node. ----->
// Inserts a new key to the tree rooted with node.
```

```
/* 1.  Perform the normal BST rotation */   ----->
/* 1.  Perform the normal BST insertion */
```

▲ | ▼ • Reply • Share ›

Raj • 6 months ago

A C++ implementation using Fenwick Tree.
Time complexity O(nlogn).
Correct me if i am wrong.

http://ideone.com/0aTtPb

3 • Reply • Share ›

Suresh Reddy • 6 months ago

Can we use insertion sort kind of algorithm here?

- Start with last before element.

- In every step, we deal with two sub arrays.
- First is left array, which we are processing. Second is right array, which we already processed and whose elements are sorted

When ever you are processing an element, the elements to the right of it already sorted. So try to find the position of it, in the array and its count will be, the number of elements to the left of it in the right sub array.
This way we will be sorting the array and we are able to get the count of smaller elements to the right of it.
Insert the element at the target position, by moving other elements by one position left to them.

Space Complexity: O(1)
Time Complexity: O(n^2). But on average it will be O(n)

• Reply • Share ›

Holden ➚ Suresh Reddy • 5 months ago

Insertion sort is O(n^2) in average case; only in best case is O(n), which is not important, since we are interested in worst case analysis of algorithms.
Your solution has no advantages over "first method" proposed here, using 2 loops, and 'using 2 loops' method is way simpler than yours :)

• Reply • Share ›

Suresh Reddy ➚ Holden • 5 months ago

Thanks Holden for reading my solution and giving the cons of it. Thanks :)

1 • Reply • Share ›

Qiaomu Yao • 6 months ago

It can be easily done with segment tree (node value is the count of each number and do the range search) by O(N) time complexity.

The link below is finding the smaller number before, but its the same method with this problem.

http://www.jiuzhang.com/soluti...

• Reply • Share ›

QILI ➚ Qiaomu Yao • 5 months ago

In my opinion, this is not O(n) time algorithm, but O(nlogn). Moreover, each time you need update the segment tree before next time of query of smaller values on the right side of the value. For segment tree, each update value is logn time, so the total time requirement is O(nlogn).

Secondly, extra space complexity depends on the range of the value, not depends on the input array size of this problem. It should be bigger than the algorithm stated in the post too.

Here, I give the C++ version of the segment tree version for this problem. If any problem, please correct it!

http://ideone.com/rfyR1p

So, I prefer the algorithm using the BST

▲ ▼ • Reply • Share ›

Holden ➚ Qiaomu Yao • 5 months ago

How do you know it is O(n)?
Wish explanations were in English instead of Chinese.

▲ ▼ • Reply • Share ›

Avata This comment was deleted.

s.s ➚ Guest • 6 months ago

wrong method ... index of 3 is 3 but element lesser than it on its right is just one n thats 0.

▲ ▼ • Reply • Share ›

nikhil jain • 7 months ago

is it necessary to balance a BST in this question?
I think there is no need of doing balancing.
correct me if i am wrong

1 ▲ ▼ • Reply • Share ›

Anup Rai ➚ nikhil jain • 7 months ago

Yes it is if we do not do it then it will take O(n^2) rather then O(nlon(n))
take the case
5 4 3 2 1
insert tree 1 (compared 0)
then 2 (Compared 1)
then 3 (compered 1,2)

then 4 (compered 1,2,3)
then 5 (compered 1,2,3,4)
in short the time complexity will go O(n^2) if the sequence is in descending order or if most of the larger digits come earlier the smaller digit.

2 ^ | v • Reply • Share ›

Holden ➚ Anup Rai • 5 months ago

Could you please why time complexity of the proposed method using 'self-balancing BST' is O(nlogn)?

^ | v • Reply • Share ›

Naveen Anto • 8 months ago

This provides O(n) solution with O(n) auxillary space

```
void find_smaller_right(int a[], int n)
{
int right[n],i,max = a[n-1],min = a[n-1];
right[n-1] = 0;
for(i = n-2; i >=0 ; i--)
{
if(a[i] < min)
{
min = a[i];
right[i] = 0;
}
else if(a[i] > max)
{
max = a[i];
right[i] = n-i-1;
}
else
right[i] = 1;
}
print(right,n);
}
```

^ | v • Reply • Share ›

TulsiRam ➚ Naveen Anto • 8 months ago

Mere dost. . Not right. I too had thought te same:

Take this array:

9 8 2 10 11 3

right[i] = 1; //////This is the problem

right[i] = 1; ////// This is the problem

1 • Reply • Share ›

rishab • 9 months ago

The person who wrote this problem is going through the bad phase of his life. But, fortunately he won some cash in his last programming event.
Now to make his girlfriend feel special, he wants to buy her some chocolates. As mentioned, he is not having good time so he want to spend as less as possible.
Keeping that in mind, he decided to play a game with her. The rule of game is as follows:
1) There are N chocolates represented by type 1..N
2) He will arrange them in a row in some random order
3) Now she (his girlfriend ofcourse) has to pick an index say i, then she will get all the chocolates at index j such that j>i and type of chocolate at j is strictly less than type of chocolate at index i.

He believes that his girlfriend is not that clever and will surely not choose the most optimal index, but he wants to know that if by any chance she picked the optimal index then how many chocolates will he have to buy.

Input:
First line contain N. then next line contain N space separated integers.

output :

see more

1 • Reply • Share ›

Holden ➚ rishab • 5 months ago

This question can be reduced to the question stated above

(count-smaller-elements-on-right-side).

We should calculate the 'countSmaller array', then return the maximum number in this array, which is the result.

```
             array:  7  6  10  5  2  8  1  9  3  4


countSmaller array:  6  5   7   4  1  3  0  2  0  0
```

Maximum is '7', which is the answer :)
So, the solution will be like above question, and can be done in O(nlogn) just like above.
P.S: I am a girl, so don't be that sure that your GF is not that clever to find out which index is the best!

• Reply • Share ›

rishab rishab • 9 months ago
can anyone solve this in o(n) ?
• Reply • Share ›

Holden rishab • 5 months ago
I highly doubt on O(n) solution; there might be, I think, O(nlogn) solution for this one.
• Reply • Share ›

vipul rishab • 7 months ago
number of chocolates subtracted by the index of the maximum number is the answer.
• Reply • Share ›

Holden vipul • 5 months ago
@vipul: What about this one?
7 6 9 5 2 8 1 10 3 4
if she picks 'i=3' which is '9', she will get '6' chocolates; but if as you said, we subtract (number of chocolates) from (index of max number), i.e.: 10 - 8 = 2, she will get only '2' chocolates.
• Reply • Share ›

BSP • 10 months ago
Can we sort the original array and compare such that for each value
if index in original and sorted array is same, then it output value is 1
else subtract the index of sorted array with the original array and
last position value in original array value is 0 and for the first
position in sorted array is 0.

Index : 0, 1, 2, 3, 4, 5, 6

Original Array : 10, 6, 15, 20, 30, 5, 7

Sorted : 5, 6, 7, 10, 15, 20, 30
Output : 3, 1, 2, 2, 2, 0, 0
• Reply • Share ›

Bharat BSP • 9 months ago
Hi BSP,

I have been trying to implement this solution for some time but couldn't able to figure out what you meant by --
'and last position value in original array value is 0 and for the first position in sorted array is 0'

array is 0 .

Please help me understand this problem by considering this example 2 14 10 11 16 7 17 21.

Your reply will be very helpful to me in understanding.

Thanks.

1 • Reply • Share ›

BSP ➚ Bharat • 9 months ago

Hi Bharat,
Thanks for pointing it out. I got to know that, it will not work.
It works for some cases but not all. Its not working for the case which you asked and also when input is already sorted.

1 • Reply • Share ›

Parul Garg • a year ago

Create Binary Search Tree
Smaller elements on Right Side = No of Elements in its left Subtree.

• Reply • Share ›

Holden ➚ Parul Garg • 5 months ago

Completely wrong.
Just draw BST of this array:
{12, 1, 2, 3, 0, 11, 4}
lots of numbers won't have left subtree, e.g. 2, 3, 11, 4!

• Reply • Share ›

Anonymous • a year ago

This can also be done with BIT and counting inversion count for each index. Here is the link http://pavelsimo.blogspot.in/2...

1 • Reply • Share ›

Holden ➚ Anonymous • 5 months ago

What will be the time complexity? better than O(nlogn)?

• Reply • Share ›

Ankit Chaudhary • a year ago

This can be solved by using Binary Index Tree with value compression in nlogn

• Reply • Share ›

TulsiRam ➚ Ankit Chaudhary • 8 months ago

Could u please explain how ?

• Reply • Share ›

Raj ➚ TulsiRam • 6 months ago

http://ideone.com/0aTtPb

• Reply • Share ›

rverma • a year ago

This is Simple Stack based Approach having
O(n) time Complexity and O(n) Auxillery Space
http://ideone.com/N2GleR
Approach is
start from right most element
if stack1 empty so push currentElement into first stack and put its top1 into result[i]
else
if stack1 not empty and stack2 is empty (if current Element is >= stack1[top1] so push currentelement into stack1 and put its top1 into result[i])
else if(currentelement is < stack1[top1] so run loop till we get smaller element on stack1[top1] or stack1 is empty and also push all poped element from stack1 into stack2. then we push currentElement into stack1 and put its top1 into result[i]

if stack1 not empty and stack2 not empty(if current element is >=stack1[top1] and <stack2[top2] so="" push="" currentelement="" into="" first="" stack="" and="" put="" its="" top1="" into="" result[i])="" else="" if(="" currentelement="" is="" <="" stack1[top1]="" so="" run="" loop="" till="" we="" get="" smaller="" element="" on="" stack1[top1]="" or="" stack1="" is="" empty="" and="" also="" push="" all="" removing="" element="" into="" stack2.="" then="" we="" push="" currentelement="" into="" stack1="" and="" put="" its="" top1="" into="" result[i])="" else="" if(currentelement="" is="">stack2[top2] and stack1[top1] so run loop till we get element which is >currentElement in stack2 or stack2 empty and also push all poped element from stack2 into stack1 then we push currentElement into stack1 and
put its top1 into result[i])

output:: is result array

• Reply • Share ›

Nara ➚ rverma • a year ago

Time complexity is O(n^2). Not O(n).

1 • Reply • Share ›

rverma ➚ Nara • a year ago

Nara another internal loop run if current element is less than from stack1 top so remove top1 from stack1 and put it into stack2 similar case form stack2 when current element is greater than stack1 top1 and also stack2 top2 sp remove stack2 top2 and put it into stack1. correct me if i am wrong.

• Reply • Share ›

v_n rverma • 7 months ago

so is the complexity of the stack based solution O(n^2) ? What about the input {10,1,9,2,8,3,7} -- I think complexity will be O(n^2)?

1 • Reply • Share ›

The_Geek • a year ago

Awsum code !

• Reply • Share ›

Load more comments