GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- 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

Remove minimum elements from either side such that 2*min becomes more than max

Given an unsorted array, trim the array such that twice of minimum is greater than maximum in the trimmed array. Elements should be removed either end of the array.

Number of removals should be minimum.

Examples:

```
arr[] = {4, 5, 100, 9, 10, 11, 12, 15, 200}
Output: 4
We need to remove 4 elements (4, 5, 100, 200)
so that 2*min becomes more than max.
```

```
arr[] = {4, 7, 5, 6}
Output: 0
We don't need to remove any element as
4*2 > 7 (Note that min = 4, max = 8)

arr[] = {20, 7, 5, 6}
Output: 1
We need to remove 20 so that 2*min becomes
more than max

arr[] = {20, 4, 1, 3}
Output: 3
We need to remove any three elements from ends
like 20, 4, 1 or 4, 1, 3 or 20, 3, 1 or 20, 4, 1
```

Naive Solution:
A naive solution is to try every possible case using recurrence. Following is the naive recursive algorithm. Note that the algorithm only returns minimum numbers of removals to be made, it doesn't print the trimmed array. It can be easily modified to print the trimmed array as well.

```
// Returns minimum number of removals to be made in
// arr[l..h]
minRemovals(int arr[], int l, int h)
1) Find min and max in arr[l..h]
2) If 2*min > max, then return 0.
3) Else return minimum of "minRemovals(arr, l+1, h) + 1"
   and "minRemovals(arr, l, h-1) + 1"
```

Following is C++ implementation of above algorithm.

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

// A utility function to find minimum of two numbers
int min(int a, int b) {return (a < b)? a : b;}

// A utility function to find minimum in arr[l..h]
int min(int arr[], int l, int h)
{
    int mn = arr[l];
    for (int i=l+1; i<=h; i++)
       if (mn > arr[i])
         mn = arr[i];
    return mn;
}

// A utility function to find maximum in arr[l..h]
int max(int arr[], int l, int h)
{
    int mx = arr[l];
    for (int i=l+1; i<=h; i++)
       if (mx < arr[i])
         mx = arr[i];
    return mx;
```

```
}

// Returns the minimum number of removals from either end
// in arr[l..h] so that 2*min becomes greater than max.
int minRemovals(int arr[], int l, int h)
{
    // If there is 1 or less elements, return 0
    // For a single element, 2*min > max
    // (Assumption: All elements are positive in arr[])
    if (l >= h) return 0;

    // 1) Find minimum and maximum in arr[l..h]
    int mn = min(arr, l, h);
    int mx = max(arr, l, h);

    //If the property is followed, no removals needed
    if (2*mn > mx)
       return 0;

    // Otherwise remove a character from left end and recur,
    // then remove a character from right end and recur, take
    // the minimum of two is returned
    return min(minRemovals(arr, l+1, h),
               minRemovals(arr, l, h-1)) + 1;
}

// Driver program to test above functions
int main()
{
  int arr[] = {4, 5, 100, 9, 10, 11, 12, 15, 200};
  int n = sizeof(arr)/sizeof(arr[0]);
  cout << minRemovals(arr, 0, n-1);
  return 0;
}
```

Output:

```
4
```

Time complexity: Time complexity of the above function can be written as following

```
T(n) = 2T(n-1) + O(n)
```

An upper bound on solution of above recurrence would be $O(n \times 2^n)$.

Dynamic Programming:
The above recursive code exhibits many overlapping subproblems. For example minRemovals(arr, l+1, h-1) is evaluated twice. So Dynamic Programming is the choice to optimize the solution. Following is Dynamic Programming based solution.

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

```
// A utility function to find minimum of two numbers
int min(int a, int b) {return (a < b)? a : b;}

// A utility function to find minimum in arr[l..h]
int min(int arr[], int l, int h)
{
    int mn = arr[l];
    for (int i=l+1; i<=h; i++)
       if (mn > arr[i])
         mn = arr[i];
    return mn;
}

// A utility function to find maximum in arr[l..h]
int max(int arr[], int l, int h)
{
    int mx = arr[l];
    for (int i=l+1; i<=h; i++)
       if (mx < arr[i])
         mx = arr[i];
    return mx;
}

// Returns the minimum number of removals from either end
// in arr[l..h] so that 2*min becomes greater than max.
int minRemovalsDP(int arr[], int n)
{
    // Create a table to store solutions of subproblems
    int table[n][n], gap, i, j, mn, mx;

    // Fill table using above recursive formula. Note that the table
    // is filled in diagonal fashion (similar to http://goo.gl/PQqoS),
    // from diagonal elements to table[0][n-1] which is the result.
    for (gap = 0; gap < n; ++gap)
    {
        for (i = 0, j = gap; j < n; ++i, ++j)
        {
            mn = min(arr, i, j);
            mx = max(arr, i, j);
            table[i][j] = (2*mn > mx)? 0: min(table[i][j-1]+1,
                                              table[i+1][j]+1);
        }
    }
    return table[0][n-1];
}

// Driver program to test above functions
int main()
{
  int arr[] = {20, 4, 1, 3};
  int n = sizeof(arr)/sizeof(arr[0]);
  cout << minRemovalsDP(arr, n);
  return 0;
```

```
}
```

Time Complexity: $O(n^3)$ where n is the number of elements in arr[].

Further Optimizations:
The above code can be optimized in many ways.
1) We can avoid calculation of min() and/or max() when min and/or max is/are not changed by removing corner elements.

2) We can pre-process the array and build segment tree in O(n) time. After the segment tree is built, we can query range minimum and maximum in O(Logn) time. The overall time complexity is reduced to $O(n^2Logn)$ time.

A O(n^2) Solution
The idea is to find the maximum sized subarray such that 2*min > max. We run two nested loops, the outer loop chooses a starting point and the inner loop chooses ending point for the current starting point. We keep track of longest subarray with the given property.

Following is C++ implementation of the above approach. Thanks to Richard Zhang for suggesting this solution.

```
// A O(n*n) solution to find the minimum of elements to
// be removed
#include <iostream>
#include <climits>
using namespace std;

// Returns the minimum number of removals from either end
// in arr[l..h] so that 2*min becomes greater than max.
int minRemovalsDP(int arr[], int n)
{
    // Initialize starting and ending indexes of the maximum
    // sized subarray with property 2*min > max
    int longest_start = -1, longest_end = 0;

    // Choose different elements as starting point
    for (int start=0; start<n; start++)
    {
        // Initialize min and max for the current start
        int min = INT_MAX, max = INT_MIN;

        // Choose different ending points for current start
        for (int end = start; end < n; end ++)
        {
            // Update min and max if necessary
            int val = arr[end];
            if (val < min) min = val;
            if (val > max) max = val;

            // If the property is violated, then no
            // point to continue for a bigger array
            if (2 * min <= max) break;
```

```
            // Update longest_start and longest_end if needed
            if (end - start > longest_end - longest_start ||
                longest_start == -1)
            {
                longest_start = start;
                longest_end = end;
            }
        }
    }

    // If not even a single element follow the property,
    // then return n
    if (longest_start == -1) return n;

    // Return the number of elements to be removed
    return (n - (longest_end - longest_start + 1));
}

// Driver program to test above functions
int main()
{
    int arr[] = {4, 5, 100, 9, 10, 11, 12, 15, 200};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << minRemovalsDP(arr, n);
    return 0;
}
```

This article is contributed by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

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: Dynamic Programming

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

Join the discussion…

Guest • 2 months ago

There is nlogn solution using sorting-

code-

```
#include <iostream>

#include <algorithm>

using namespace std;

// A utility function to find minimum of two numbers

int min(int a, int b) {return (a < b)? a : b;}

// A utility function to find minimum in arr[l..h]

int min(int arr[], int l, int h)

{

int mn = 99999;
```

see more

• Reply • Share ›

Abhinav Chauhan • 3 months ago

@GeeksforGeeks
O(n^2 log n) solution.

1. Build two BIT, one for minimum and for maximum in an interval.
2. Check each pair for possible solution.

Each pair can be checked in O(log n) time.

• Reply • Share ›

prashant jha • 3 months ago

here is my dp solution with optimisation using segment tree
https://ideone.com/WM0fyD

• Reply • Share ›

Moiz Ali • 4 months ago

one more O(nlogn) solution for this problem

http://ideone.com/iWtSao

• Reply • Share ›

rverma • 5 months ago

This is O(n) Solution for this Problem.
http://ideone.com/x6tnUU

• Reply • Share ›

Chao Zhou • 6 months ago

There is an O(n*log(n)) solution for this problem.

The main idea is based on the observation that you can not increase min or decrease max by adding more number, but you can only do it by cut off number.

Thus we can use a sliding window go through the entire array to keep track of the longest satisfied sub-array in O(n), and build a sparse table to query min and max in range(i, j) in O(1).

Since building a sparse table will take time O(n*log(n)), the entire algorithm will have the time complexity of O(n*log(n)).

Here is the code in python:
http://ideone.com/NaZX87

1 • Reply • Share ›

Guest • 7 months ago

In 3rd example,

_____

arr[] = {20, 4, 1, 3}
Output: 3
We need to remove any three elements from ends
like 20, 4, 1 or 4, 1, 3 or 20, 3, 1 or 20, 4, 1

Output should be 2.

Remove 20 and then 1.

min = 3 , max =4. & 2*min > max.

@GeeksforGeeks

2 • Reply • Share ›

Neyaz Ahmad Guest • 7 months ago

We only trim from Start or End.

• Reply • Share ›

Norbert Madarász • 9 months ago

Here is an O(n*log(n)) solution.
Using a min and a max sparse table
and binary searching on subarray length:
http://ideone.com/GcMcjX

Both segment tree and sparse table need O(n*log(n))
memory and build time, but sparse table has better query
performance, O(1) vs segment tree's O(log(n)).
In return segment table supports dynamic updates (in
this specific problem this feature is not needed).

4 ^ | v • Reply • Share ›

Norbert Madarász • 9 months ago

Here is an O(n*log(n)*log(n)) solution.
Using a min and a max segment tree
binary search on subarray length:
http://ideone.com/rpsZPS

^ | v • Reply • Share ›

Pushpendra Patel • 9 months ago

This problem can be solved in O(n) or O(nlogn) (depending upon queue implementation) . This problem can be stated as finding maximum size sub-array such that sub-array holds given condition. Iterate through the array . keep pushing element in queue at back until condition breaks, then keep poping element at front until condition satisfies . element is array index. Note that an element is touched maximum twice, once for push, then for pop. Hence problem can be solved in O(n) , if queue support specified operations in constant time.
http://ideone.com/G0j0kH

2 ^ | v • Reply • Share ›

Guest ➚ Pushpendra Patel • 5 months ago

Instead of taking queue if we take min heap and max heap with elements with their index, than it can be solved in O(nlogn).

^ | v • Reply • Share ›

Pushpendra Patel ➚ Guest • 5 months ago

consider example, {3,2,5,3,4,5} . whatever min/max heap you will use, it will fail. Simple reason, min/max heap don't maintain relative order of elements , in which they were inserted.

^ | v • Reply • Share ›

Mohammad Moiez Gohar ➚ Pushpendra Patel • 7 months ago

how do u implement min, max in O(1) time in the queue?

• Reply • Share ›

Pushpendra Patel ➚ Mohammad Moiez Gohar • 5 months ago

try to implement a queue, which will only tell what is max/min of current elements in queue. This is similar to the question in which we are asked to implement a stack which gives min also in O(1) time. for queue amortized complexity will be O(1). (although single operation may take (n) time, but tight bound is O(1)). I will soon update proper implementation.

• Reply • Share ›

youtt • 9 months ago

This problem can be done in n.logn.

for check if maximum length of final array, we need not check from 1 upto N.
just make a binary search here. for eg.
Use this if we can find a final array of length x, then we can find atleast one final array such that its length is <= x .

• Reply • Share ›

Anurag Singh ➚ youtt • 9 months ago

Please explain more. Array is not sorted. How will you use binary search.

• Reply • Share ›

Hamoon • 10 months ago

Using the segment tree and the algorithm of Richard Zhang
above one can get an O(nlogn) algorithm. See below for the code.
Note that the number of call made to get the min or max in segment tree is O(n), as in each iteration of the for loop either l or r gets incremented.

```
/* package whatever; // don't place package name! */

import java.util.*;

import java.lang.*;

import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */

class Ideone

{

public static void main (String[] args) throws java.lang.Exception
```

see more

• Reply • Share ›

rishg ➚ Hamoon • 9 months ago

This problem can be done in nlogn.
for check if maximum length of final array, we need not check from 1 upto N.
just make a binary search here. for eg.
Use this if we can find a final array of length x, then we can find atleast one final array such that its length is <=x.

• Reply • Share ›

kasif • 10 months ago

in the optimized version of O(n^2) , min and max are inititalized to INT_MIN and INT_MAX. what are INT_MIN and INT_MAX??

• Reply • Share ›

Arpit ➚ kasif • 9 months ago

INT_MIN and INT_MAX are the minmum and maximum values that an integer variable can hold defined under climits or limits.h

for more details refer http://www.cplusplus.com/refer...

• Reply • Share ›

Guest • 10 months ago

the above solution is wrong for the following array {4, 5, 100, 9, 10, 11, 12, 15, 101,102,103,104,105,106,107,108,109,110,111,112}

it should return 7 whereas it is returning 8. Please admin see to it. Sort the following array and first 7 elements are removed to satisfy the condition that 2*min>max.

• Reply • Share ›

krishna ➚ Guest • 10 months ago

i think you can't sort these numbers, read problem statement if you sort numbers that means you are changing their relative position, so it should be 8 not 7

• Reply • Share ›

Guest • 10 months ago

I have solved the above one in Time Complexity:O(n) and space Complexity : O(1).

The Code is in C and well commented . Compiled and Run and gave correct results for all the sample inputs given in this web page.

http://ideone.com/0aZ7s2

Compile using GCC.
PS: Not done using Dynamic Programming

• Reply • Share ›

mittal Guest • 10 months ago

that sol is not working for this input

4 5 9 10 11 12 17 20

• Reply • Share ›

kAnN mittal • 10 months ago

Yeah You are Right :) This algo will not work in case of two overlapping sequences . Will need at least O(n2) to do it correctly . Sorry for the wrong answer provided :)

• Reply • Share ›

smonhiat • 10 months ago

in dp wht does this statemnt returns

table[i][j] = (2*mn > mx)? 0: min(table[i][j-1]+1,table[i+1][j]+1); when j=0

plz help me through this

• Reply • Share ›

np smonhiat • 10 months ago

for the first time when j=0 always 2*mn>mx will be true so it will return 0 because we have only 1 element to consider so 2*mn>mx will always be true

• Reply • Share ›

Jun • 10 months ago

@GeeksforGeeks

Please guide, how to print the trimmed array in case of method1, the recurrence is creating issues in the printing

• Reply • Share ›

Vineel Kumar Reddy Kovvuri • 10 months ago

http://ideone.com/LE5XUs

Looks like this can be solved in O(n) skeptic

The idea here is to grab a window who's length is maximum and with the constraint mentioned in the problem.

But by carefully sliding the window and handling the overlapping window cases. Looks like it can be solved in O(n)

step1: let the current window be at 0 both i = 0, j = 0 and also let min, max be at index 0

step2: check if new element will fall in to following there categories.

step3: new <= arr[min] // could be a new minimum to expand the window

step 3a: if twice of new minimum is greater than current max //{7,3, [21],|19|,34,31,600,500,320,260,270};

then we can update the current min

see more

• Reply • Share ›

Preethi • a year ago

Sort the array initially,
1.Take the first element ie. i.
2.Multiply i with 2 compare it with largest element
3.If not greater than compare with second largest no
4.if not > then simply delete i and first largest element.
4.continue the first step til you got

• Reply • Share ›

Jun ➚ Preethi • 10 months ago

Hey preethi,
U r missing out a point here, we are allowed to trim the array from only the either sides.
Your approach sorts the array first, which itself tampers the original ordering of the elements.
Take for instance , the array

4,5,100,9,17,10,250,11,12,15,200
according to ur algo, the output should be
10,11,12,15,100
but these are not continuous elements of the original array, means we deleted some intermediate elements as well, which is not allowed
So, according to me this approach is not correct.
Anyhow, in cases when we need to find a subarray satisfying any condition, sorting is a BAD idea, since it disturbs the original array.
Its sheer coincidence that might give correct answer sometimes.
:)

• Reply • Share ›

Preethi ➚ Jun • 10 months ago

Thanks for notifying my mistake

• Reply • Share ›

Jun ➚ Preethi • 10 months ago

Welcome dear

• Reply • Share ›

AlienOnEarth • a year ago

Another algorithm can be:

1.) start from middle element of the array
2.) expand left side as much as possible, when condition does not satisfy stop
3.) expand right side as much as possible, when condition does not satisfy stop

Please let me know if there is any problems with this algo

• Reply • Share ›

AlienOnEarth ➚ AlienOnEarth • a year ago

I think the above algorithm is wrong as it assumes that the middle elements will come in the window. But that may not be the case:

{4,5,300,6,2,3}

• Reply • Share ›

kAnN ➚ AlienOnEarth • 10 months ago

You are in the right path . I am right now coding for a O(n) algo which is similar to yours . Wait for it :)

• Reply • Share ›

tweety • a year ago

what changes we need to make in naive solution to print the trimmed array.... plz somebody tell mee...??

1 • Reply • Share ›

Jun ➚ tweety • 10 months ago

I doubt if there is a way out.I tried , but recursion somehow doesn't allow to give precise results:(:(

• Reply • Share ›

Joy • a year ago

I posted my code http://ideone.com/5COs8l (with running input output)

I follow the approach.

For each element in array
Treat it as expected_min
scroll through right and left to check for the condition

O(n^2) Solution in python.

• Reply • Share ›

silu • a year ago

#include<bits stdc++.h="">

using namespace std;

int a[1010],n,ans = 99999999;

int main()

{

cin>>n;

for(int i=0;i<n;++i) cin="">>a[i];

for(int i=0;i<n;++i) {="" int="" pos="i,mn" ==="" a[pos],mx="a[pos],temp" ==="" 1;="" for(int="" l="pos-1,r" ==="" pos+1,cl="0,cr" ==="" 0;l="">=0 || r< n;--l,++r)

{

if(cl and cr) break;

see more

1 • Reply • Share ›

Minhaz Palasara • a year ago

O(n2) written in java. Please reply in case if it fails

/**
* @author Minhaz
* Find the minimum number of removals in order to get the 2.min > max
*
* from
* ----------
*To |
* |
* |
*/

public class DP1 {
/**Building the Minimum and Maximum value from one point in array to other
* and hence building the validity matrix
* */

void getValidArrayRange(int[] data){

see more

• Reply • Share ›

sidceo • a year ago

We can solve the above problem with O(n) time complexity using dynamic programming :
#include<stdio.h>

int S[100][100];

int arr[100];

int minm(int a,int b){

return (a<b)?a:b; }="" int="" fun(int="" l,int="" h){="" int="" min="9999;" int="" max="0;" int="" i;="" if(s[l][h]!="0){" return="" s[l][h];="" }="" for(i="l;i<h;i++){" if(arr[i]<min){="" min="arr[i];" }="" if(arr[i]="">max){

max=arr[i];

}

}

if(2*min>max){

see more

• Reply • Share ›

bgautam • a year ago

One doubt! Can we sort the array and do a O(n^2) check in the array for the max condition. While checking the array we can take alternatively first and last element from the sorted array! Is this approach correct!

• Reply • Share ›

Alien ➔ bgautam • a year ago

Sorting will not work in this case

• Reply • Share ›

Guest • a year ago

Instead of running the loops for the entire size of array everytime, once a small size is computed you need to run inner loop less than the size obtained since by checking for larger lengths is just waste of time!
Thoughts?!

• Reply • Share ›

Nitin Sharma • a year ago

I think complexity of my code is nlogn for this problem. If any one finds it wrong then please comment here . Here is my code
http://ideone.com/c29tNr

• Reply • Share ›

peki • a year ago

The preprocessing step in solution 2 can be done the following way:

We can make matrices Min[n][n], Max[n][n] that will keep min and max of array segments [i,j]. Precisely Min[i][j] will be the minimum value of array segment [i,j], and Max[i][j] will be maximum value of array segment[i,j]. This precomputaion takes O(n^2), but we can change the function calls

mn = min(arr, i, j);
mx = max(arr, i, j);

with mn = Min[i][j], mx = Min[i][j] which is, of course, O(1).

Total time complexity is O(n^2) + O(n^2) = O(n^2), so we can achieve O(n^2) with dynamic programming.

1 • Reply • Share ›

Joy ➚ peki • a year ago

creative approach. but it can be done in plain n^2 without the matrix
http://ideone.com/5COs8l

• Reply • Share ›

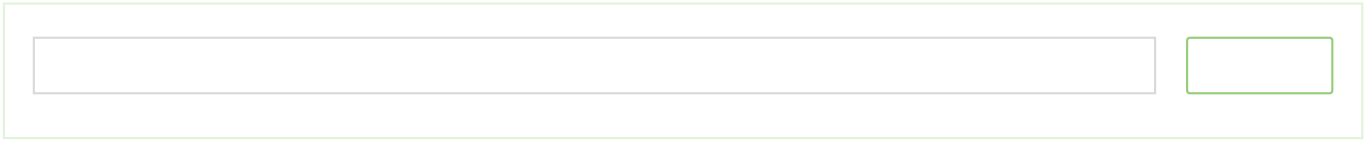
CodeGuest ➚ peki • a year ago

Min[i][j] and Max[i][j] can be updated inside same loop using

Min[i][j] = getMin(Min[i][j-1],Min[i+1][j]) and similarly for Max,
just initialize all Min[i][i] = a[i] and Max[i][i] = a[i] so total it will be O(n^2)

• Reply • Share ›

Load more comments



- 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

- lt_k

i need help for coding this function in java...

Java Programming Language · 2 hours ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago