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.

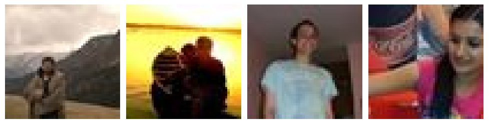
Examples:

```
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
```

Google™ Custom Search

GeeksforGeeks

53,519 people like GeeksforGeeks.



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

```
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.

```cpp
#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;
```

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

```
}

// 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 x 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.

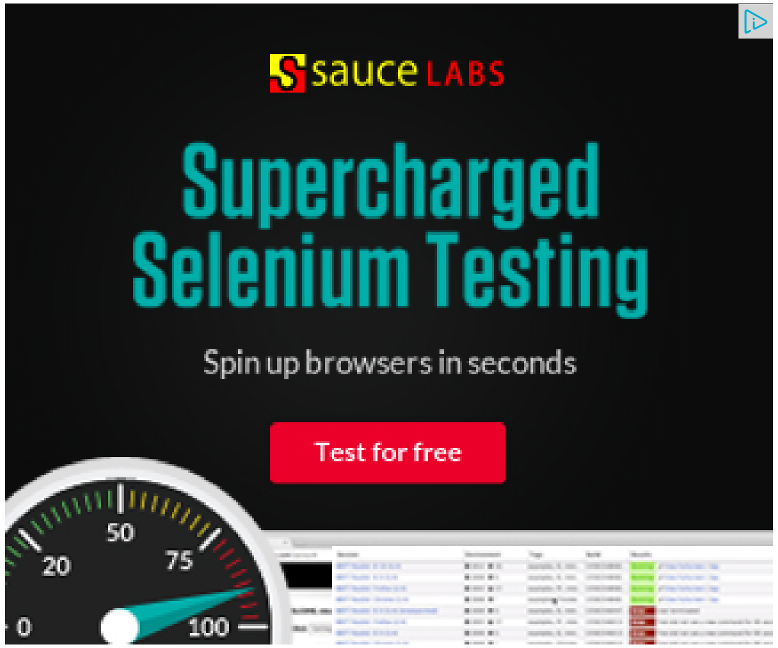
Following is C++ implementation of the above approach. Thanks to Richard Zhang for suggesting this solution.

```
// 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 Tpoics:

- 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
- Sort n numbers in range from 0 to n^2 – 1 in linear time

10 Tweet 3 0

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

41 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

Nitin Sharma • 2 days ago
I think complexity of my code is nlogn for this problem. If any one finds it wrong
my code
http://ideone.com/c29tNr
• Reply • Share ›

peki • 4 days 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
be the minimum value of array segment [i,j], and Max[i][j] will be maximum valu
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)
1 • Reply • Share ›

CodeGuest ➜ peki • 12 hours 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 ›

nani • 5 days ago

1- Define a 1 dimensional array A of 10 integers.
DisplayArray is a function that accepts an array and displays the array elemen
3- ArraySum is a function that returns the sum of the array elements.
4- displayRever is a function that accepts an array and displays the array elements in reverse order.

Reply • Share ›

guest • 6 days ago

I have a small doubt. Is it necessary to always start removing from left side firs
100}; , you only need to remove 100 from right side it that will be the minimum
solution proposed will also remove 9 and 5 from left end.

I know the questions says "Elements should be removed either end of the arra
from left hand side?

Reply • Share ›

Abhinay Madhasu • 7 days ago

hai rahul. you can more optimise the code by putting the following code

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

outside the 2nd for loop. it will give the same result and reduce the no of instru

Reply • Share ›

Ritesh · 7 days ago

In the 4th example provided,
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

why isn't the answer '2'? by removing 20 and 1????

· Reply · Share ›

kaushik Lele ➦ Ritesh · 5 days ago

How can it be 2? You can not directly remove "1". You have to start trin
reach "1". So you have to trim 20 & 4 and then 1. So three removals.
Or other way 20, 3, 1 or 20, 4, 1. Thus every way needs three remaval

· Reply · Share ›

Vipul Bansal ➦ Ritesh · 7 days ago

Yep you are absolutely correct!
Check my solution in comments. Its gives exactly what you desire.

· Reply · Share ›

kapil · 8 days ago

Simple O(n) solution. start from the middle of array , move both side ,keep tra
violated then stop moving that side but keep moving other side.
plz comment if code fails any condition..

```
#include<stdio.h>
#define SIZE 9
int main()
{
// int a[]={4,5,6,7};
int a[]={4,5,100,9,10,11,12,15,200};\
trim(a);
}
void trim(int a[])
{
int mid1=SIZE/2-1;
int mid2=mid1+1;
int flag1=0;
int flag2=0;
int i;
```

see more

Reply · Share ›

kaushik Lele ➦ kapil · 5 days ago

It gives wrong results for int a[]={201,100,9,101,11,12,15,4,5,200};
It returns {11,12,15,4,5} which violates condition

Reply · Share ›

kaushik Lele ➦ kapil · 5 days ago

Check for this array {201,4,5,100,9,101,11,12,15,200};
Program prints final array as "9". Which requires nine removals

Where as it should return {4,5} which needs just eight removals.

As mentioned in question we should achieve task with minimum remo

Reply · Share ›

AlienOnEarth · 8 days ago

Thank you Richard for o(n^2) Algorithm. My code using same algorithm is as b

```
void findSubArray(int arr[], int n)

{
```

int i,j;

int min,max;

int curlen=0, maxlen=0; // curlen = size of current window, maxlen = size of m

int start =0;

for(i=0;i<n;i++) {="" min="INT_MAX;" max="INT_MIN;" curlen="0;" for(j="i;j&lt;n
for="" current="" window="" if(min="">arr[j])

min = arr[j];

if(max<arr[j]) max="arr[j];" if(2*min="" <="" max)="" {="" break;="" }="" else=""
{="" start="i;" maxlen="curlen;" }="" }="" }="" }="" printf("\noutput:="" %d\n",n-m

Reply • Share ›

mitesh tambi • 8 days ago

int minE(int a, int b) {return (a < b)? a : b;}
int minRemovalsDP(int arr[], int n)
{
int min[n][n];
int max[n][n];
int Min, Max;
for(int i=0; i < n; i++)
{
Min=Max=min[i][i]=max[i][i]=arr[i];
for(int j=i+1; j < n ;j++)
{

if(Min > arr[j])
{
Min=arr[j];

```
}
min[i][j]=Min;
```

see more

Reply • Share ›

Harjit Singh • 9 days ago

This can be done in O(n logn). Here is the psedocode.

1) Sort the array (nlogn)

2) Take two pointers , start at the beginning of array and end at the end of arra

```
start = 0;

end= n-1;
```

3) Find out the min difference after consider removing either side of the eleme

```
if(2*A[start]> A[end]
break;
else

if((A[end-1] - 2*A[start] < A[end]-2*A[start+1]))

end--;
```

see more

Reply • Share ›

kaushik Lele ➦ Harjit Singh • 5 days ago

Sorting the array changes all the positions of numbers. We need to ma

Reply • Share ›

AlienOnEarth Harjit Singh • 8 days ago

Can not sort the array. Check comment provided by Rahul.

1 • Reply • Share ›

Yitao Li • 9 days ago

```
#include <stdio.h>

#define MAX_N 128
#define LOG_MAX_N 16

#define MIN(a, b) (((a) < (b)) ? (a) : (b))
#define MAX(a, b) (((a) > (b)) ? (a) : (b))

/*
* remove minimum elements from either side of an array such that 2 * min be
* (assuming at least one positive element exists in the input array)
*
* solution using sparse table RMQ :
*
*/

unsigned int msb(unsigned int x) {
static const unsigned int bval[] = {0, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4};
unsigned int r = 0;
```

see more

Reply • Share ›

Yitao Li Yitao Li • 9 days ago

and time complexity would be O(n^2 + n log n) -> O(n^2)

Sukoi • 9 days ago

how about finding the min and max value in all the intervals and store them in t
the intervals with maximum length that satisfies the 2*min > max, here the i,j v
interval .....
suggestions are appreciated ....

• Reply • Share ›

Harsh • 9 days ago

How about this O(n) soln :

For above input
arr[] = {4, 5, 100, 9, 10, 11, 12, 15, 200}

make 2 arrays - min array and max array:
Take middle index and find cumulative min and max while moving outwards to
In this case ,start from arr[4], and proceed towards left and right extremes sto

minArr[]={4,5,9,9,10,10,10,10,10}
maxArr[]={100,100,100,10,10,11,12,15,200}

Now we need to remove 4 elements from original array arr[] (4, 5, 100, 200)
so that 2*min(in minArr[] from both ends) becomes more than max(in maxArr[
).

• Reply • Share ›

RK • 9 days ago

Can somebody shed some light at the below statement please?
table[i][j] = (2*mn > mx)? 0: min(table[i][j-1]+1, table[i+1][j]+1);

What I understand, we are trying to fill table[i][j] by using value at table[i+1][j]. E

Am I missing something?

Reply • Share ›

sushant singh ➦ RK • 9 days ago

It is computed, try to do a dry run of the program. First everything lengt
Then we do length 2's. So, [01] [12] [23].
Then length 3. [02] [13] [24]

So while doing [13] we need to know [1,2] and [2,3] both are length 2 a
calculated.

Hope it helps

Reply • Share ›

Ankit Vani • 9 days ago

Simple O(n^2) solution: http://ideone.com/NdytMa

Reply • Share ›

GOPI GOPINATH ➦ Ankit Vani • 9 days ago

can u please explain it in words....Thanks in advance

Reply • Share ›

GOPI GOPINATH ➦ Ankit Vani • 9 days ago

its nice

Reply • Share ›

Guest ➦ GOPI GOPINATH • 9 days ago

can u please explain it in words....Thanks in advance.

Reply • Share ›

Vipul Bansal • 9 days ago

Heres my solution:

1. Find the max value in the array and divide it by 2.

2. Look if any value in array is greater than value obtained in 1.
3. If yes then that is your desired array else cancel that max value.
4. And if any value in array is less than max/2 cancel it too.

• Reply • Share ›

Coder011 Vipul Bansal • 9 days ago

int a[]={1,1,1,1,1,1,1,1,101,200,1}; According 2 ur approach, you would answer is {1,1,1,1,1,1,1,1} , and no. of elements removed=3.

• Reply • Share ›

Vipul Bansal Coder011 • 9 days ago

One question: In an array of identical values how will you deterr
Whereas in problem statement you are required to use min and

• Reply • Share ›

Coder011 Vipul Bansal • 9 days ago

min=max, otherwise it should be mentioned in the ques
distinct values, if not so , then we have to assume min a

• Reply • Share ›

Vipul Bansal Coder011 • 9 days ago

But is it not evident from the distinct values they have us
I am not denying what you say. Instead I am working on
believe your min=max assumption is irrelevant!

Anybody else here wants to clarify??

• Reply • Share ›

Vipul Bansal Coder011 • 9 days ago

Oh sorry I did not read 'Number of removals should be minimur
Hmm..I'll have to apply a different approach now.
Thanks for replying :)

• Reply • Share ›

Coder011 • 9 days ago

O(n^2) greedy approach. Start from every element in the array, incrementing c
2*curmin="">curmax. curmin and curmax need to updated at every step.
Update start and end variables (used to print the trimmed array) , if the curleng
seen so far.

Code: http://ideone.com/nBDmgn

• Reply • Share ›

Richard Zhang • 10 days ago

Shouldn't it be simply O(n^2)? Did I miss anything? Here is my code:

public class Trim {

public static void main(String[] args) {

int[] ary = new int[] {4, 5, 100, 9, 10, 11, 12, 15, 200};

printArray(ary);

printArray(trim(ary));

ary = new int[] {4, 7, 5, 6};

printArray(ary);

printArray(trim(ary));

ary = new int[] {20, 7, 5, 6};

printArray(ary);

see more

1 ^ | v • Reply • Share ›

GeeksforGeeks Mod ➦ Richard Zhang • 10 days ago

Thanks for suggesting a more efficient approach. It looks good. We wc
add it to the original post.

^ | v • Reply • Share ›

Aditya Joshi • 10 days ago

Instead of the O(n^3) approach, you could first sort the array in O(nlogn) and u
demonstrated here: https://ideone.com/o0fsb0

That is O(n^2)

1 ^ | v • Reply • Share ›

Rahul ➦ Aditya Joshi • 10 days ago

Adiya, your approach doesn't seem to work for

{4, 5, 100, 9, 10, 11, 12, 15, 200}.

The expected output is 4, but your approach seems to be producing 3.

^ | v • Reply • Share ›

Aditya Joshi ➦ Rahul • 10 days ago

It does produce 4. You have to sort the array first. Check https:

^ | v • Reply • Share ›

Rahul ➦ Aditya Joshi • 10 days ago

May be correct result in this case, But sorting the array
always as you loose the position information. For exam
{8, 9, 100, 10, 11, 12, 15, 200} would become same afte
array and 4 in second array.

1 ^ | v • Reply • Share ›

Aditya Joshi ➦ Rahul • 10 days ago

Ok. You can't modify the array but I took the liberty of dc
mentioned in the description that you can't. Thanks!

^ | ˅ • Reply • Share ›

Subscribe   Add Disqus to your site