Merge Overlapping Intervals

Given a set of time intervals in any order, merge all overlapping intervals into one and output the result which should have only mutually exclusive intervals. Let the intervals be represented as pairs of integers for simplicity.
For example, let the given set of intervals be {{1,3}, {2,4}, {5,7}, {6,8} }. The intervals {1,3} and {2,4} overlap with each other, so they should be merged and become {1, 4}. Similarly {5, 7} and {6, 8} should be merged and become {5, 8}

We strongly recommend that you click here and practice it, before moving on to the solution.

Write a function which produces the set of merged intervals for the given set of intervals.

A **simple approach** is to start from the first interval and compare it with all other intervals for overlapping, if it overlaps with any other interval, then remove the other interval from list and merge the other into the first interval. Repeat the same steps for remaining intervals after first. This approach cannot be implemented in better than O(n^2) time.

An **efficient approach** is to first sort the intervals according to starting time. Once we have the sorted intervals, we can combine all intervals in a linear traversal. The idea is, in sorted array of intervals, if interval[i] doesn't overlap with interval[i-1], then interval[i+1] cannot overlap with interval[i-1] because starting time of interval[i+1] must be greater than or equal to interval[i]. Following is the detailed step by step algorithm.

```
1. Sort the intervals based on increasing order of
    starting time.
2. Push the first interval on to a stack.
3. For each interval do the following
   a. If the current interval does not overlap with the stack
       top, push it.
   b. If the current interval overlaps with stack top and ending
       time of current interval is more than that of stack top,
        update stack top with the ending  time of current interval.
  4. At the end stack contains the merged intervals.
```

Below is a C++ implementation of the above approach.

```
// A C++ program for merging overlapping intervals
#include<bits/stdc++.h>
using namespace std;

// An interval has start time and end time
struct Interval
{
    int start, end;
};

// Compares two intervals according to their staring time.
// This is needed for sorting the intervals using library
// function std::sort(). See http://goo.gl/iGspV
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start);
}

// The main function that takes a set of intervals, merges
// overlapping intervals and prints the result
void mergeIntervals(Interval arr[], int n)
{
    // Test if the given set has at least one interval
    if (n <= 0)
        return;

    // Create an empty stack of intervals
    stack<Interval> s;

    // sort the intervals in increasing order of start time
    sort(arr, arr+n, compareInterval);

    // push the first interval to stack
    s.push(arr[0]);

    // Start from the next interval and merge if necessary
    for (int i = 1 ; i < n; i++)
    {
        // get interval from stack top
        Interval top = s.top();

        // if current interval is not overlapping with stack top,
        // push it to the stack
        if (top.end < arr[i].start)
            s.push(arr[i]);

        // Otherwise update the ending time of top if ending of current
        // interval is more
        else if (top.end < arr[i].end)
        {
            top.end = arr[i].end;
            s.pop();
            s.push(top);
        }
    }

    // Print contents of stack
    cout << "\n The Merged Intervals are: ";
    while (!s.empty())
    {
        Interval t = s.top();
        cout << "[" << t.start << "," << t.end << "] ";
        s.pop();
    }
    return;
}

// Driver program
int main()
{
    Interval arr[] =  { {6,8}, {1,9}, {2,4}, {4,7} };
    int n = sizeof(arr)/sizeof(arr[0]);
    mergeIntervals(arr, n);
    return 0;
}
```

Run on IDE

Output:

```
The Merged Intervals are: [1,9]
```

Time complexity of the method is O(nLogn) which is for sorting. Once the array of intervals is sorted, merging takes linear time.

A O(n Log n) and O(1) Extra Space Solution

The above solution requires O(n) extra space for stack. We can avoid use of extra space by doing merge operations in-place. Below are detailed steps.

```
1) Sort all intervals in decreasing order of start time.
2) Traverse sorted intervals starting from first interval,
   do following for every interval.
      a) If current interval is not first interval and it
         overlaps with previous interval, then merge it with
         previous interval. Keep doing it while the interval
         overlaps with the previous one.
      b) Else add current interval to output list of intervals.
```

Note that if intervals are sorted by decreasing order of start times, we can quickly check if intervals overlap or not by comparing start time of previous interval with end time of current interval.

Below is C++ implementation of above algorithm.

```
// C++ program to merge overlapping Intervals in
// O(n Log n) time and O(1) extra space.
#include<bits/stdc++.h>
using namespace std;

// An Interval
struct Interval
{
    int s, e;
};

// Function used in sort
bool mycomp(Interval a, Interval b)
{   return a.s > b.s; }

void mergeIntervals(Interval arr[], int n)
{
    // Sort Intervals in decreasing order of
    // start time
    sort(arr, arr+n, mycomp);
    int index = 0; // Stores index of last element
    // in output array (modified arr[])

    // Traverse all input Intervals
    for (int i=0; i<n; i++)
    {
        // If this is not first Interval and overlaps
        // with the previous one
        if (index != 0 && arr[index-1].s <= arr[i].e)
        {
            while (index != 0 && arr[index-1].s <= arr[i].e)
            {
                // Merge previous and current Intervals
                arr[index-1].e = max(arr[index-1].e, arr[i].e);
                arr[index-1].s = min(arr[index-1].s, arr[i].s);
                index--;
            }
        }
        else // Doesn't overlap with previous, add to
            // solution
            arr[index] = arr[i];

        index++;
    }

    // Now arr[0..index-1] stores the merged Intervals
    cout << "\n The Merged Intervals are: ";
    for (int i = 0; i < index; i++)
        cout << "[" << arr[i].s << ", " << arr[i].e << "] ";
}

// Driver program
int main()
{
    Interval arr[] =  { {6,8}, {1,9}, {2,4}, {4,7} };
    int n = sizeof(arr)/sizeof(arr[0]);
    mergeIntervals(arr, n);
    return 0;
}
```

Run on IDE

Output:

```
The Merged Intervals are: [1,9]
```

Thanks to Gaurav Ahirwar for suggesting this method.

This article is compiled by Ravi Chandra Enaganti. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

104 Comments Category: Arrays Stack Tags: stack

Related Posts:

- Replace every element with the least greater element on its right
- Remove duplicates from an array of small primes
- Find Surpasser Count of each element in array

- Print all subarrays with 0 sum
- Find frequency of each element in a limited range array in less than O(n) time
- Minimum sum of two numbers formed from digits of an array
- Maximum value K such that array has at-least K elements that are >= K
- Find number of subarrays with even sum

(Login to Rate and Mark)

3.4 Average Difficulty : **3.4/5.0**
Based on **46** 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.

104 Comments **GeeksforGeeks** 1 **Login**

Recommend 3 **Share** **Sort by Newest**

Join the discussion…

Ratnesh Singh · 11 days ago
simple code using c++ stl

http://ideone.com/CP8rlo

· Reply · Share ›

Gourav Goswami · 15 days ago
```
#include<bits stdc++.h="">
using namespace std;

struct Interval
{
int start, end;
};

Interval fun(Interval A[],int left,int right)
{
if(left==right) return A[left];

int mid=left+(right-left)/2;

Interval L= fun(A,left,mid);
Interval R = fun(A,mid+1,right);

Interval temp;
temp.start=min(L.start, R.start);
```

see more

Reply • Share ›

Anmol Varshney • a month ago

Much easier implementation based on sorting in increasing order of start time: https://ideone.com/Jb6CLb

Reply • Share ›

ABC • a month ago

https://ideone.com/23wSdn

Reply • Share ›

TheManHasNoName ➦ ABC • a month ago

nice n clean :)

Reply • Share ›

ABC ➦ ABC • a month ago

O(1) extra space solution and intervals are in ascending order according to starting time.

Reply • Share ›

Anmol Varshney ➦ ABC • a month ago

Much easier to understand than your version: https://ideone.com/Jb6CLb

Reply • Share ›

AllergicToBitches • 2 months ago

No need to pop and push again to stack. We can use below line -

```
s.top().end = arr[i].end;

instead of

/*top.end = arr[i].end;
s.pop();
s.push(top);*/
```

Reply • Share ›

Ritvik Raj ➦ AllergicToBitches • a month ago

Right. Can you explain why we need to sort the array in descending order in the second method? I think the ascending order will work fine.

Reply • Share ›

AllergicToBitches ➦ Ritvik Raj • a month ago

I think as we are using stack, it will print in LIFO order. So we start from the end, so merged intervals in the starting will be printed first.

1 ^ | ˅ • Reply • Share ›

bhavik gujarati • 2 months ago

C solution using both the approaches:

http://ideone.com/B3z053

ideone.com/UwEgAy

Java solution using approach-2:

http://ideone.com/ioH2YW

1 ^ | ˅ • Reply • Share ›

bhavik gujarati • 2 months ago

Can we use ascending order according to start time in solution with O(1) approach?

^ | ˅ • Reply • Share ›

Mr. Chugh • 2 months ago

using c++ stl (pair & stack) : https://ideone.com/TzGlWf

^ | ˅ • Reply • Share ›

Raghav Agarwal • 2 months ago

```
JAVA->
class interval {
int st, en;
interval(int s, int e) {
this.st = s;
this.en = e;
}
}
public class intervalmerge {
public static void main(String args[]) throws IOException {
ArrayList<interval> arr = new ArrayList<interval>();
arr.add(new interval(6, 8));
arr.add(new interval(2, 4));
arr.add(new interval(1, 9));
arr.add(new interval(4, 7));
Collections.sort(arr, new Comparator<interval>() {
public int compare(interval o1, interval o2) {
return o1.st-o2.st;
```

see more

^ | ˅ • Reply • Share ›

Joshua Greenfield • 3 months ago

Here's a solution in python. It's a pretty concise solution. This was inspired by

Here's a solution in python. It's a pretty concise solution. This was inspired by Gaurav Ahirwar. It also does the merging of the array in place so space complexity is O(1). Time complexity is also O(n*log(n)).

http://code.geeksforgeeks.org/...

^ | v • Reply • Share ›

Mr. Chugh • 4 months ago

Does second method require the pairs to be sorted according to first elements particularly in decreasing order only? Willn't it work even if we sort them in increasing order of start time with slight modifications in conditions to check the overlapping?

^ | v • Reply • Share ›

wilson • 5 months ago

sort based on second element and can be easily solve :)

1 ^ | v • Reply • Share ›

Pranay • 5 months ago

@GeeksForGeeks..
Second method fails for test cases {6,8}, {1,9}, {2,4}

The merging might be needed to carry backward

^ | v • Reply • Share ›

GeeksforGeeks Mod → Pranay • 5 months ago

Thanks for pointing this out. We have updated the second method.

^ | v • Reply • Share ›

rockeblaze • 6 months ago

If the input is {6,8}, {1,9}, {2,3}, {4,7} shouldn't it still return {1,9}?

^ | v • Reply • Share ›

Truong Khanh Nguyen • 6 months ago

My solutions in java here http://www.capacode.com/array/.... Discuss 2 solutions, one using stack and one sort

^ | v • Reply • Share ›

CoderHax • 6 months ago

a simple c++ O(n) solution http://code.geeksforgeeks.org/...

^ | v • Reply • Share ›

Jack Snoeyink → CoderHax • 6 months ago

Not O(n) unless by n you mean the total number of all integers in all intervals.

^ | v • Reply • Share ›

GanesH AvacharE • 7 months ago

Hello,
I have written this code using Binary tree

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

#define MAX 1024

typedef struct tree {
int interval[2];
struct tree *left, *right;
}tree;

tree *newNode(int *interval) {
tree *t;

t = (tree *)malloc(sizeof (tree));
t->interval[0] = interval[0];
t->interval[1] = interval[1];
```

see more

^ | v • Reply • Share ›

GanesH AvacharE ➦ GanesH AvacharE • 7 months ago

ignore bstInsertion function not of use

^ | v • Reply • Share ›

GanesH AvacharE ➦ GanesH AvacharE • 7 months ago

and It is not working for For eg, (4, 5), (2,3), (1, 6)
What all changes can be helpful to run this code for such i/ps?

^ | v • Reply • Share ›

GuruSimhe • 8 months ago

Second solution doesn't work in case two or more mutually exclusive intervels are followed by a super interval, encompassing all the previous ones. For eg, (4, 5), (2,3), (1, 6)

1 ^ | v • Reply • Share ›

Siddharth Gupta • a year ago

solved using sorting and interval overlap test. Did not use stack.

http://code.geeksforgeeks.org/...

Note: The code for some reason doesn't run using the GeeksForGeeks IDE :(..
copy paste locally and run...

^ | v • Reply • Share ›

QILI • a year ago

A google interview question

Reply • Share ›

TulsiRam • a year ago

I think we can do the same. if we sort on the basis of finish time

Reply • Share ›

TulsiRam • a year ago

Good question

Reply • Share ›

Karan Kapoor • a year ago

I wish I read it earlier :/

1 • Reply • Share ›

Satish Srinivas • a year ago

Above implementation using c++ stl map,stack.

http://ideone.com/lO3ugJ

Reply • Share ›

Jeff • a year ago

Prints in the order of arrival

http://ideone.com/h5IFYZ

Reply • Share ›

Dman • a year ago

Simple implementation without using stack.

Implemented by sorting according to end time.

https://ideone.com/5e3Dw8

2 • Reply • Share ›

Sunny Jain • a year ago

Without using Stack O(nlogn)

http://ideone.com/Ety7WQ

Reply • Share ›

RacUnite • a year ago

we can sort intervals in increasing order of finishing time O(nlog(n)) .Then while traversing the interval we can compare start time(i+1) "<"finish time(i) if true then overlap.

as we do in ALLOCATION ALGORITHM

Reply • Share ›

Unique ➦ RacUnite • a year ago

Your solution will fail for following case:
{2,3},{4,5},{1,6}

2 ^ | v • Reply • Share ›

limitless ➦ Unique • 9 months ago

{ {2,3}, {4,5}, {6,7}, {8,9}, {1, 10} }The Merged Intervals are: [8, 9] [6, 7] [4, 5] [1, 10]

```
vector<interval> merge(vector<interval>& intervals) {
int n = intervals.size();
if(n<=1)
return intervals;
sort(intervals.begin(), intervals.end(), Solution::mycomp);
vector<interval> res;
res.push_back(intervals[0]);
for(int i=1;i<n;i++) {="" if(res.back().end="">= intervals[i].start)
res.back().end = max(res.back().end, intervals[i].end);
else
res.push_back(intervals[i]);
}
return res;
}
```

^ | v • Reply • Share ›

Akhil ➦ Unique • a year ago

if we use stack, and merge till the finish time is less than start time(i+1), then i think, it works...

^ | v • Reply • Share ›

RacUnite ➦ Unique • a year ago

thank you !

^ | v • Reply • Share ›

radioactive_platypus • a year ago

Any shortcomings of the following solution or any test cases that I have not accounted for?

http://ideone.com/jgMDNj

Time - O(nlogn) with constant space.

Much Appreciated.

^ | v • Reply • Share ›

Mr. Lazy • a year ago

Without using stack: http://ideone.com/cpXid4
TC : O(nlogn)

Auxiliary Space : O(1)

7 ^ | v • Reply • Share ›

GuruSimhe ➦ Mr. Lazy • 8 months ago

Try with (1,6) (2,3) (4,5). Answer should be (1,6) but your's give (1,6) (4,5)

^ | v • Reply • Share ›

dasun ➦ Mr. Lazy • 10 months ago

Auxiliary space is at least O(log(n)) for sorting stack space

^ | v • Reply • Share ›

Mr. Lazy ➦ dasun • 10 months ago

No problem.. can use quicksort (qsort)

^ | v • Reply • Share ›

Aditya Verma ➦ Mr. Lazy • a year ago

It is correct.. you inserted the test case.. but didn't change the line

vector<interval> intervals(intvls, intvls+4);

it should be

vector<interval> intervals(intvls, intvls+5);

1 ^ | v • Reply • Share ›

Mr. Lazy ➦ Aditya Verma • a year ago

LOL! .. My bad.. updated comment. Thanks for pointing out! :D

^ | v • Reply • Share ›

anish_ratnawat • a year ago

No need to use stack.Below is a code for my solution.Intervals will be pass in arraylist

public class Solution {

public ArrayList<interval> merge(ArrayList<interval> intervals) {

ArrayList<interval> res=new ArrayList<interval>();

Collections.sort(intervals, new Comparator<interval>(){

public int compare(Interval i1,Interval i2){

return i1.start-i2.start;

}

});

Interval newInterval=intervals.get(0);

for(int i=1;i<intervals.size();i++){ interval="" t1="intervals.get(i);" if(newinterval.end<t1.start){="" res.add(newinterval);="" newinterval="t1;" }else{="" newinterval.start="Math.min(newInterval.start,t1.start);" newinterval.end="Math.max(newInterval.end,t1.end);" }="" }="" res.add(newinterval);="" return="" res;="" }="" }="">

Reply • Share ›

Bhaskar Bagchi • a year ago

This can be used as an elegant solution http://stackoverflow.com/a/454...

Reply • Share ›