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]
```