Day 08

Thursday, January 7, 2021 7:05 PM

Homework Assignments :

1. Implement Merge Sort and Quicksort.
2. https://leetcode.com/problems/merge-intervals/
3. https://leetcode.com/problems/insert-interval/

4. https://leetcode.com/problems/meeting-rooms/
5. https://leetcode.com/problems/meeting-rooms-ii

6. https://leetcode.com/problems/my-calendar-i/
7. https://leetcode.com/problems/my-calendar-ii/

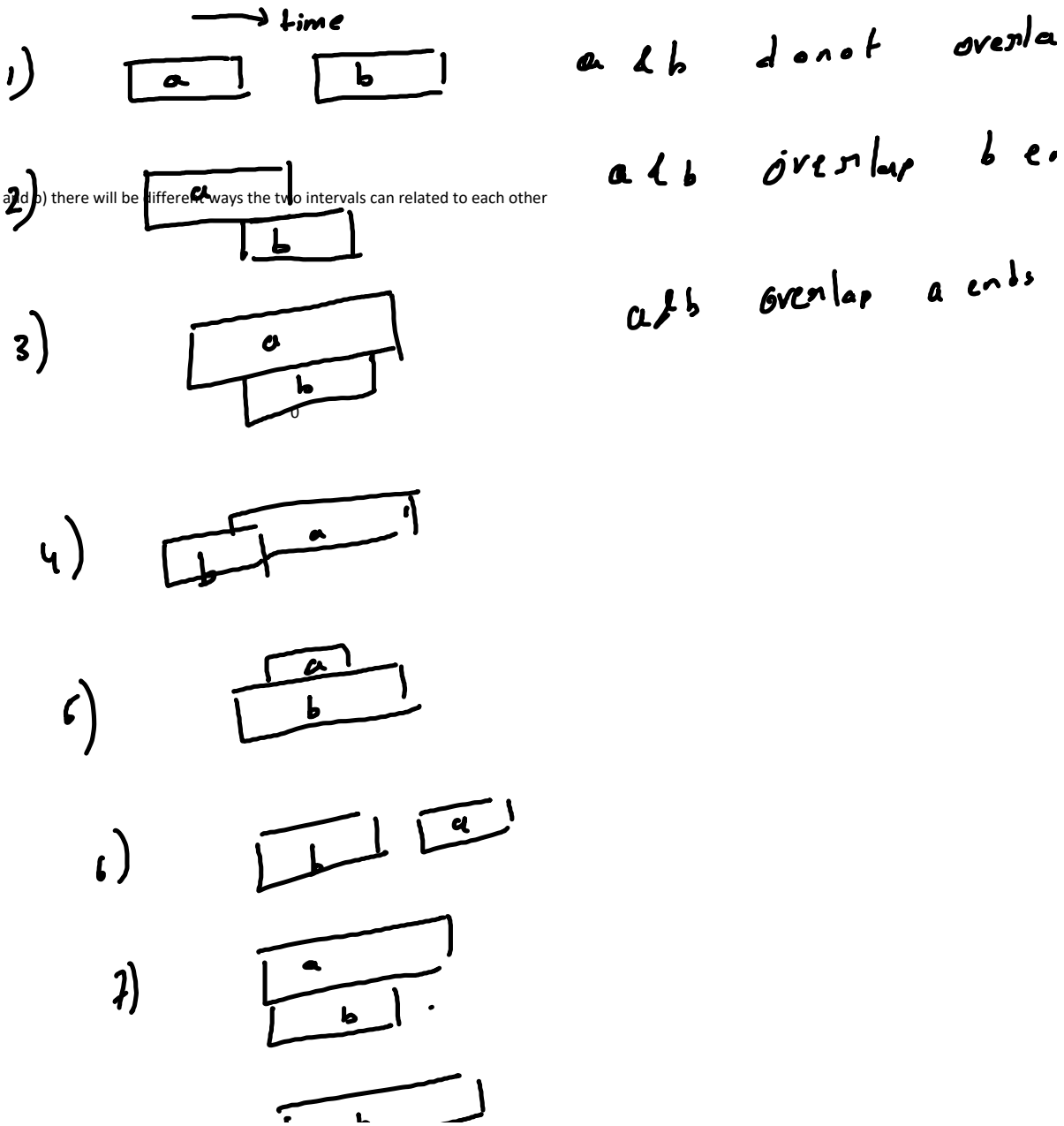
8. https://leetcode.com/problems/interval-list-intersections/

Understand all the sorting algorithms :

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort
6. Bucket Sort

Merge Intervals :

Given two intervals (a and b) there will be different ways the two intervals can related to each other



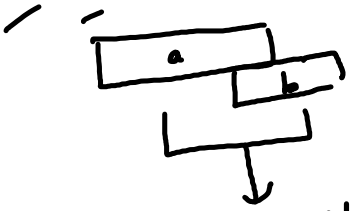
8) 

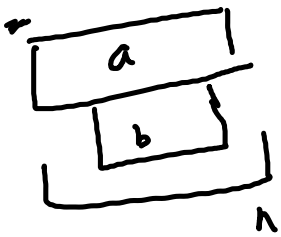
Intervals :- [[1,4], [2,5], [7,9]]

M overlap intervals

[[1, 5] , [7, 9]]

[[1,4],



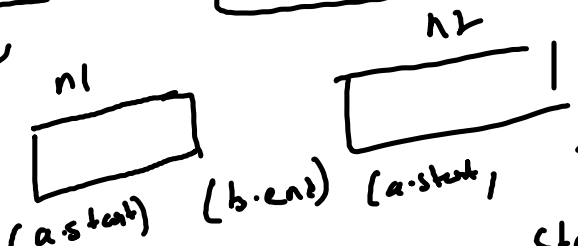


interval

{ [start
end

}

Merge Intervals =>



List<int>

1) sort the intervals on start ti

a.start <= b.start.

Collection.sort(intervals, (a,b) ->

=>

https://dzone.com/articles/java-8-lambda-functions-usage-examples

list<interval> mergeIntervals

.

```
class Interval
{
   int start ;
   int end;

   public Interval(int start , int end )
   {
      this.start = start;
      this.end = end;
   }
```

```
}


Class MergeIntervals
{
   public static List<Interval> merge(List<Interval> intervals )
   {
      if(intervals.size() < 2)
      {
         return intervals;
      }

      //Sort it
      Collections.sort(intervals , (a, b) -> Integer.Compare(a.start, b,start));

      List<Interval> mergeIntervals = new LinkedList();
      Iterator<Interval> intervalItr = interval.iterator();;
      Interval interval = intervalIt.next();
      int start = interval.start;
      int end  = interval .end ;

      //[1, 4] , [2, 5] , [7 , 9]

      // Start has 1
      // End has 4
      // interval has 2 & 5
      while(intervalItr.hasNext())
      {
         interval = interval.next();

         if(interval.start <= end)
         {
            end = Math.max(interval.end , end);
         }
         else{
            mergedIntervals.add(new Interval(start , end ));
            start = interval.start;
            end = interval.end;
         }
      }

      mergedIntervals.add(new Interval(start, end);

      return intervals;
   }

}
```

Merge Intervals

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and re
overlapping intervals that cover all the intervals in the input.

Example 1:
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

Given a set of *non-overlapping* intervals, insert a new interval into the intervals (merge if necessa
You may assume that the intervals were initially sorted according to their start times.

Given a set of *non-overlapping* intervals, sorted by there start time , insert a given interval at the
And merge all the nessary intervals to produce a list

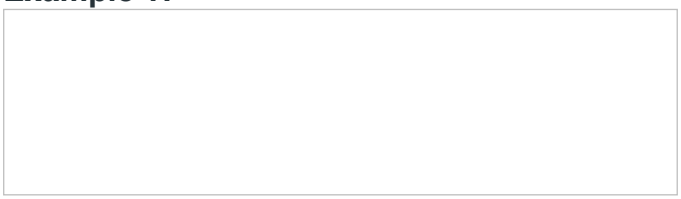
Example 1:
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]

Output: [[1,5],[6,9]]

Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
1, 2] , [3,10] ,[12, 16

Interval List Intersections
Given two lists of **closed** intervals, each list of intervals is pairwise disjoint and in sorted order.
Return the intersection of these two interval lists.
(Formally, a closed interval [a, b] (with a <= b) denotes the set of real numbers x with a <= x <= b.
closed intervals is a set of real numbers that is either empty, or can be represented as a closed i
the intersection of [1, 3] and [2, 4] is [2, 3].)

Example 1:



Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],[15,24],[25,26]]

Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]

Arr 1[0,2] Arr 2[1, 5]

Arr1[i].start >= Arr2[j]. Start && Arr1[i].start <= arr2[j] .end
ll arr2[j].start >= arr1[i]. Start && arr2[j].start <= arr1[i].end

Result.add(new Interval(Math.max(arr[i].start , arr2[j].start), Math.min(I end , j end);

If(arr[i].end < arr2[j].end)
{
 i++
}
Else
{
J++;
}

Implement a MyCalendar class to store your events. A new event can be added if adding the event
Your class will have the method, book(int start, int end). Formally, this represents a booking on the
end.
A *double booking* happens when two events have some non-empty intersection (ie., there is some t

For each call to the method MyCalendar.book, return true if the event can be added to the calendar not add the event to the calendar.
Your class will be called like this: MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)

Example 1:

MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true

Explanation:

The first event can be booked. The second can't because time 15 is already booked by another ev
The third event can be booked, as the first event takes every time less than 20, but not including 2(

Algorithm

We will maintain a list of interval *events* (not necessarily sorted).
Evidently, two events [s1, e1) and [s2, e2) do *not* conflict if and only if one of them starts after the oth
By De Morgan's laws, this means the events conflict when s1 < e2 AND s2 < e1.

```
public class MyCalendar {
   List<int[]> calendar;

   MyCalendar() {
      calendar = new ArrayList();
   }

   public boolean book(int start, int end) {
      for (int[] iv: calendar) {
         if (iv[0] < end && start < iv[1]) return false;
      }
      calendar.add(new int[]{start, end});
      return true;
   }
}
```

Given a collection of intervals, find the minimum number of intervals you need to remove to make tl

Example 1:
Input: [[1,2],[2,3],[3,4],[1,3]]
Output: 1
Explanation: [1,3] can be removed and the rest of intervals are non-overlapping.

[4, 8] ,[5 , 7]

```
if (iv[0] < end && start < iv[1]) return false;
```

iV[0] --> 4
End --> 7

---

4 5 6 7 8

[4 , 8] [5 , 9]