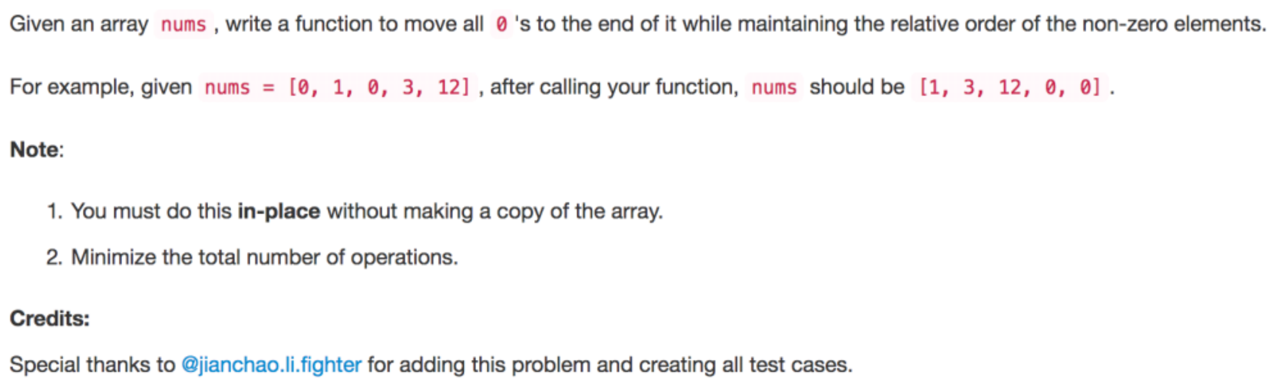
283. Move Zeroes

Saturday, February 18, 2017 3:16 PM



Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

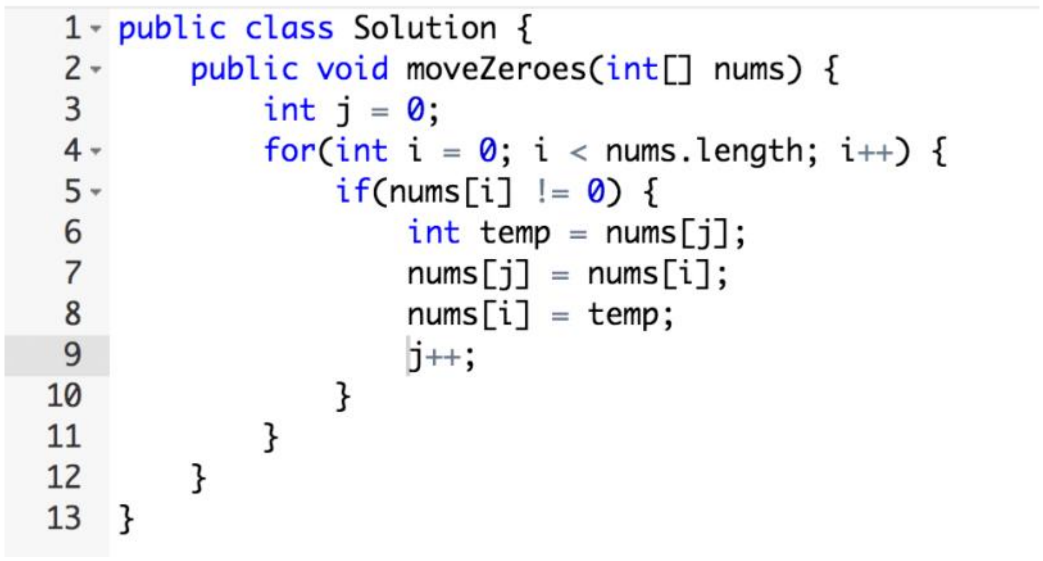
For example, given `nums = [0, 1, 0, 3, 12]`, after calling your function, `nums` should be `[1, 3, 12, 0, 0]`.

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

Credits:

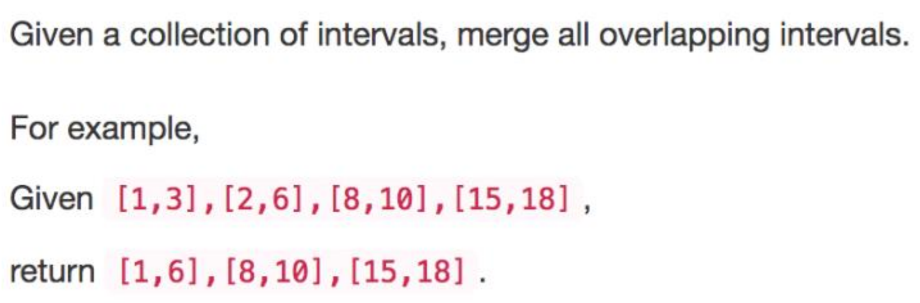
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.



```java
public class Solution {
    public void moveZeroes(int[] nums) {
        int j = 0;
        for(int i = 0; i < nums.length; i++) {
            if(nums[i] != 0) {
                int temp = nums[j];
                nums[j] = nums[i];
                nums[i] = temp;
                j++;
            }
        }
    }
}
```

56. Merge Intervals

Saturday, February 18, 2017 4:18 PM

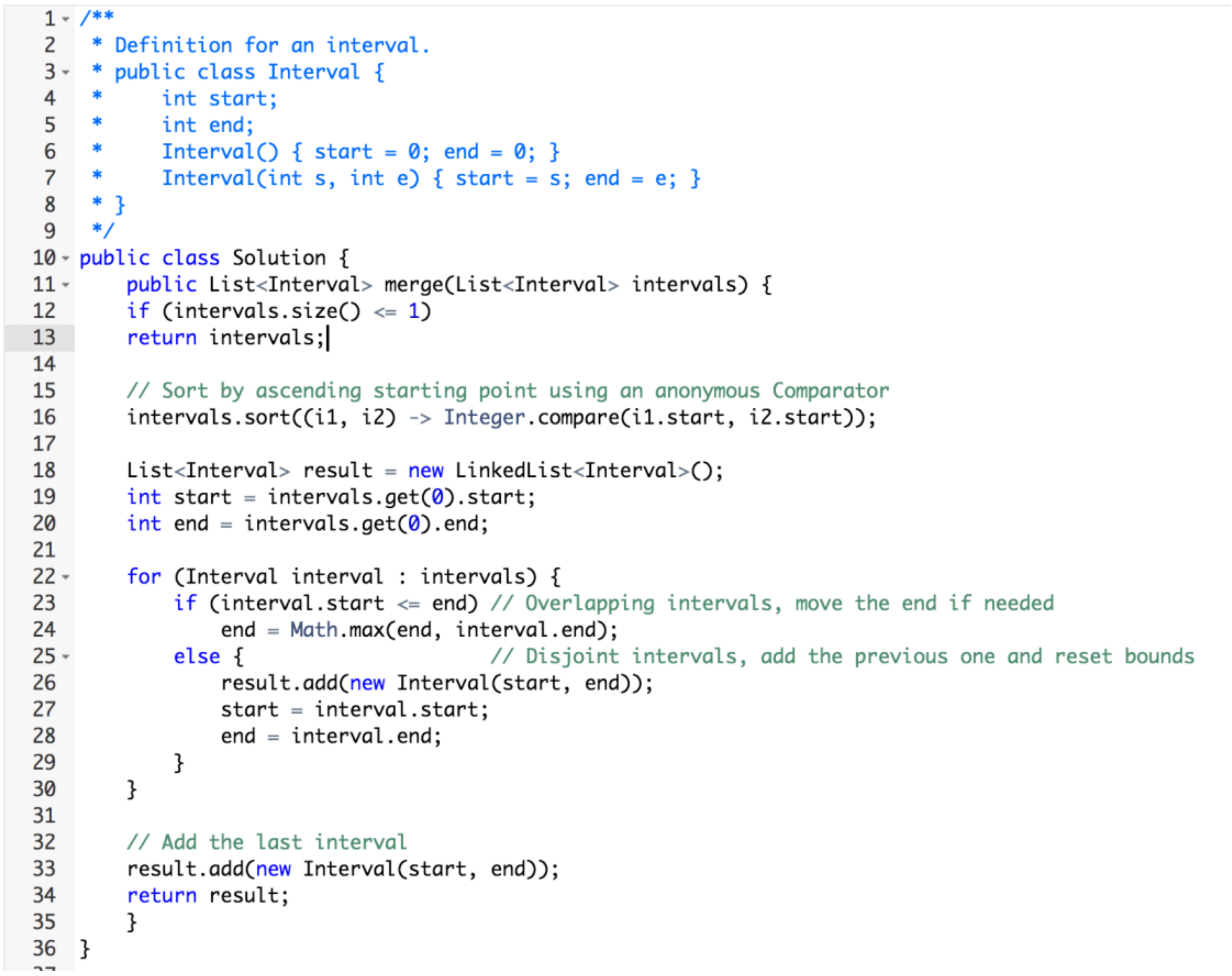


Given a collection of intervals, merge all overlapping intervals.

For example,

Given `[1,3],[2,6],[8,10],[15,18]`,

return `[1,6],[8,10],[15,18]`.



```java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
    if (intervals.size() <= 1)
    return intervals;

    // Sort by ascending starting point using an anonymous Comparator
    intervals.sort((i1, i2) -> Integer.compare(i1.start, i2.start));

    List<Interval> result = new LinkedList<Interval>();
    int start = intervals.get(0).start;
    int end = intervals.get(0).end;

    for (Interval interval : intervals) {
        if (interval.start <= end) // Overlapping intervals, move the end if needed
            end = Math.max(end, interval.end);
        else {                     // Disjoint intervals, add the previous one and reset bounds
            result.add(new Interval(start, end));
            start = interval.start;
            end = interval.end;
        }
    }

    // Add the last interval
    result.add(new Interval(start, end));
    return result;
    }
}
```

The other way to sort:

```java
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        Collections.sort(intervals, new Comparator<Interval>(){
            @Override
            public int compare(Interval obj0, Interval obj1) {
                return obj0.start - obj1.start;
            }
        });

        List<Interval> ret = new ArrayList<>();
        Interval prev = null;
        for (Interval inter : intervals) {
            if (  prev==null || inter.start>prev.end ) {
                ret.add(inter);
                prev = inter;
            } else if (inter.end>prev.end) {
                // Modify the element already in list
                prev.end = inter.end;
            }
        }
        return ret;
    }
}
```

57. Insert Interval

Saturday, February 18, 2017 8:03 PM

Given a set of *non-overlapping* intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

Given intervals `[1,3],[6,9]` , insert and merge `[2,5]` in as `[1,5],[6,9]` .

Example 2:

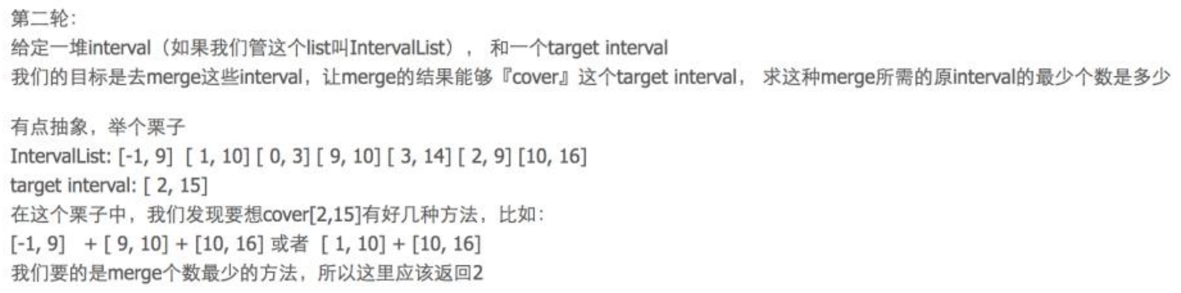
Given `[1,2],[3,5],[6,7],[8,10],[12,16]` , insert and merge `[4,9]` in as `[1,2],[3,10],[12,16]` .

This is because the new interval `[4,9]` overlaps with `[3,5],[6,7],[8,10]` .

```java
public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
    List<Interval> result = new ArrayList<Interval>();
    for (Interval i : intervals) {
        if (newInterval == null || i.end < newInterval.start)
            result.add(i);
        else if (i.start > newInterval.end) {
            result.add(newInterval);
            result.add(i);
            newInterval = null;
        } else {
            newInterval.start = Math.min(newInterval.start, i.start);
            newInterval.end = Math.max(newInterval.end, i.end);
        }
    }
    if (newInterval != null)
        result.add(newInterval);
    return result;
}
```

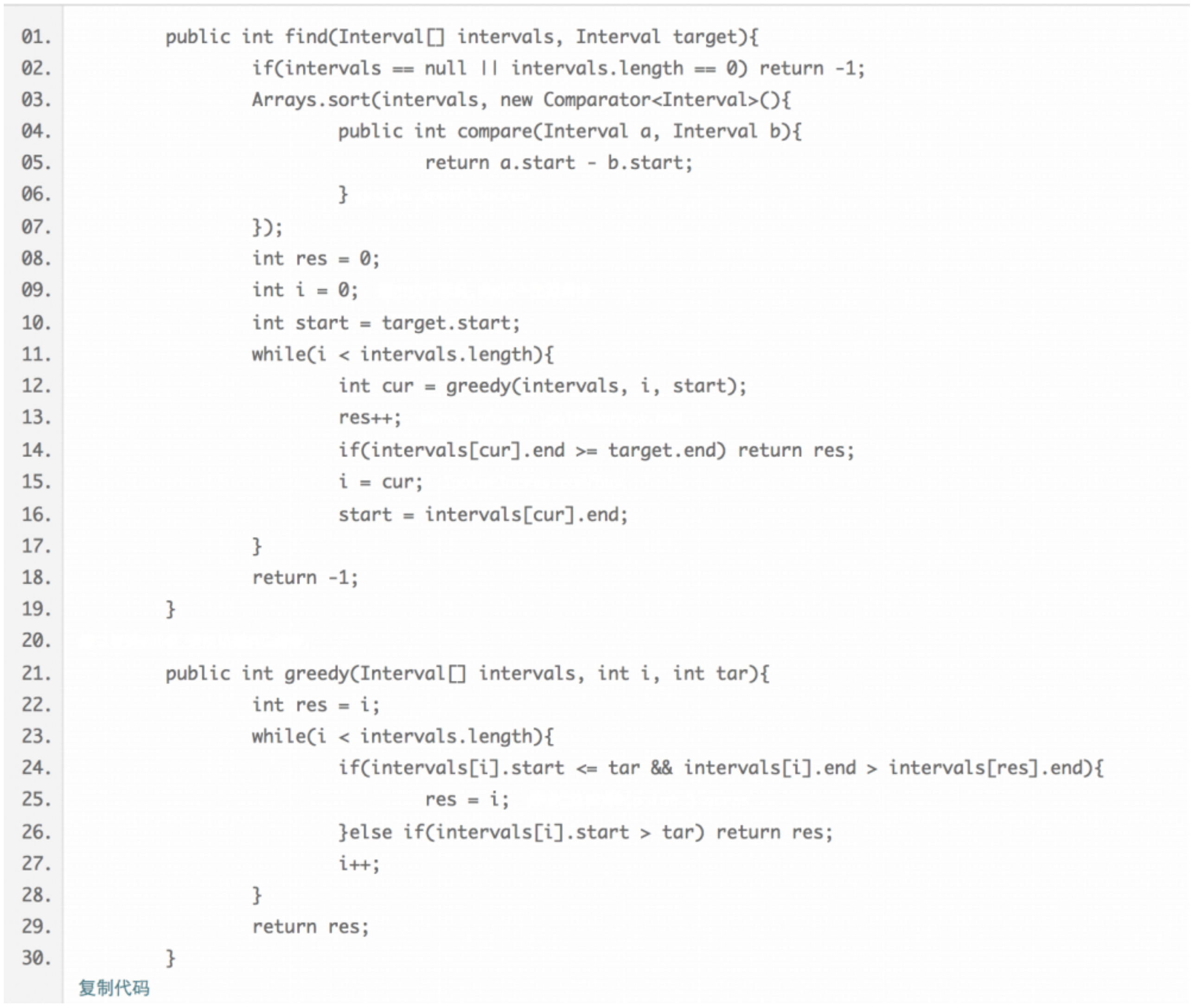
Cover interval minimum number

Tuesday, February 28, 2017 2:12 PM



第二轮：
给定一堆interval（如果我们管这个list叫IntervalList），和一个target interval
我们的目标是去merge这些interval，让merge的结果能够『cover』这个target interval， 求这种merge所需的原interval的最少个数是多少

有点抽象，举个栗子
IntervalList: [-1, 9] [1, 10] [0, 3] [9, 10] [3, 14] [2, 9] [10, 16]
target interval: [2, 15]
在这个栗子中，我们发现要想cover[2,15]有好几种方法，比如：
[-1, 9] + [9, 10] + [10, 16] 或者 [1, 10] + [10, 16]
我们要的是merge个数最少的方法，所以这里应该返回2



```java
public int find(Interval[] intervals, Interval target){
    if(intervals == null || intervals.length == 0) return -1;
    Arrays.sort(intervals, new Comparator<Interval>(){
        public int compare(Interval a, Interval b){
            return a.start - b.start;
        }
    });
    int res = 0;
    int i = 0;
    int start = target.start;
    while(i < intervals.length){
        int cur = greedy(intervals, i, start);
        res++;
        if(intervals[cur].end >= target.end) return res;
        i = cur;
        start = intervals[cur].end;
    }
    return -1;
}

public int greedy(Interval[] intervals, int i, int tar){
    int res = i;
    while(i < intervals.length){
        if(intervals[i].start <= tar && intervals[i].end > intervals[res].end){
            res = i;
        }else if(intervals[i].start > tar) return res;
        i++;
    }
    return res;
}
```

252. Meeting Rooms

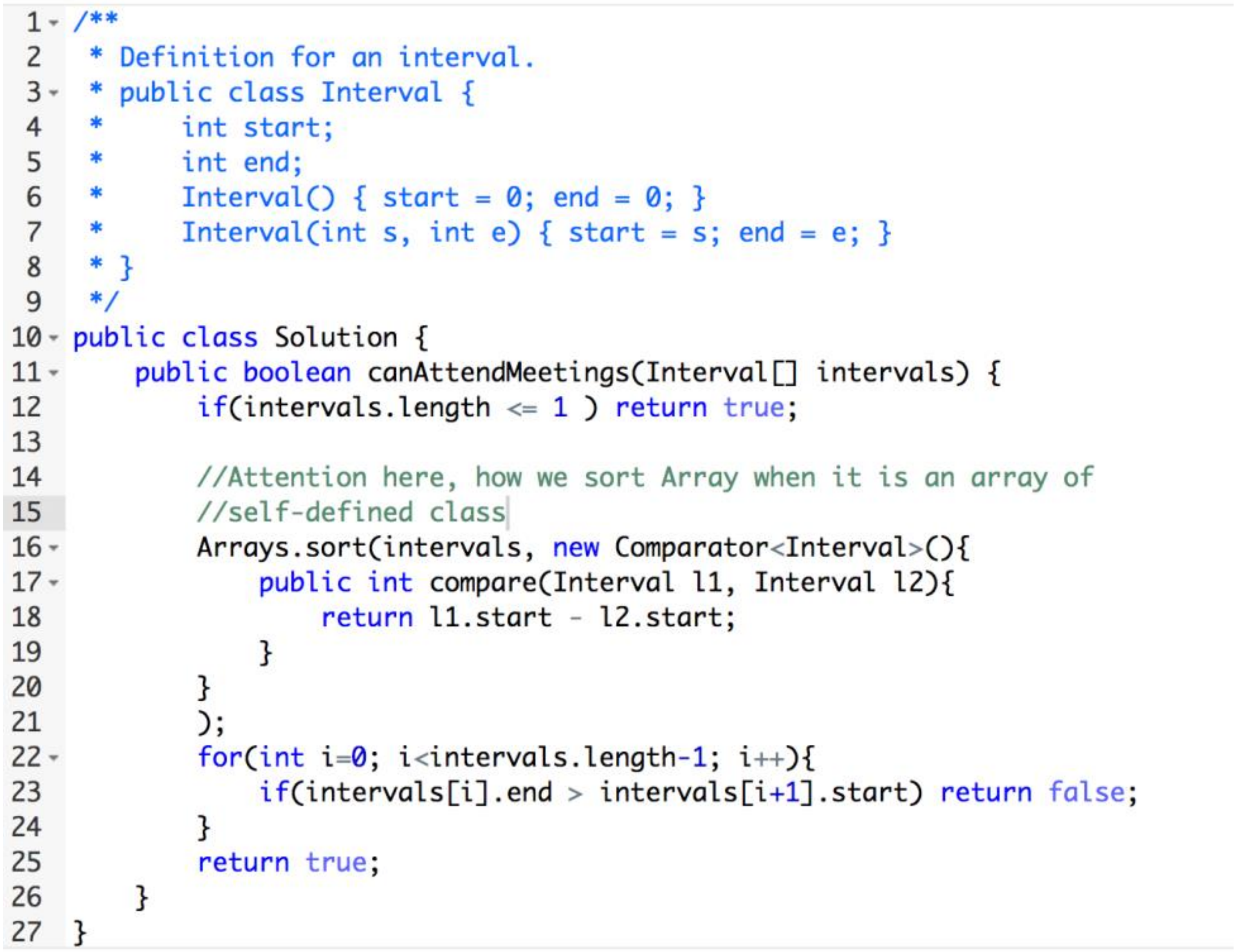
Monday, February 20, 2017 7:38 PM

Given an array of meeting time intervals consisting of start and end times `[[s1,e1],[s2,e2],...]` ($s_i < e_i$), determine if a person could attend all meetings.

For example,

Given `[[0, 30],[5, 10],[15, 20]]`,

return `false`.



```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public boolean canAttendMeetings(Interval[] intervals) {
        if(intervals.length <= 1 ) return true;

        //Attention here, how we sort Array when it is an array of
        //self-defined class
        Arrays.sort(intervals, new Comparator<Interval>(){
            public int compare(Interval l1, Interval l2){
                return l1.start - l2.start;
            }
        }
        );
        for(int i=0; i<intervals.length-1; i++){
            if(intervals[i].end > intervals[i+1].start) return false;
        }
        return true;
    }
}
```

253. Meeting Rooms II

Saturday, February 18, 2017 4:58 PM

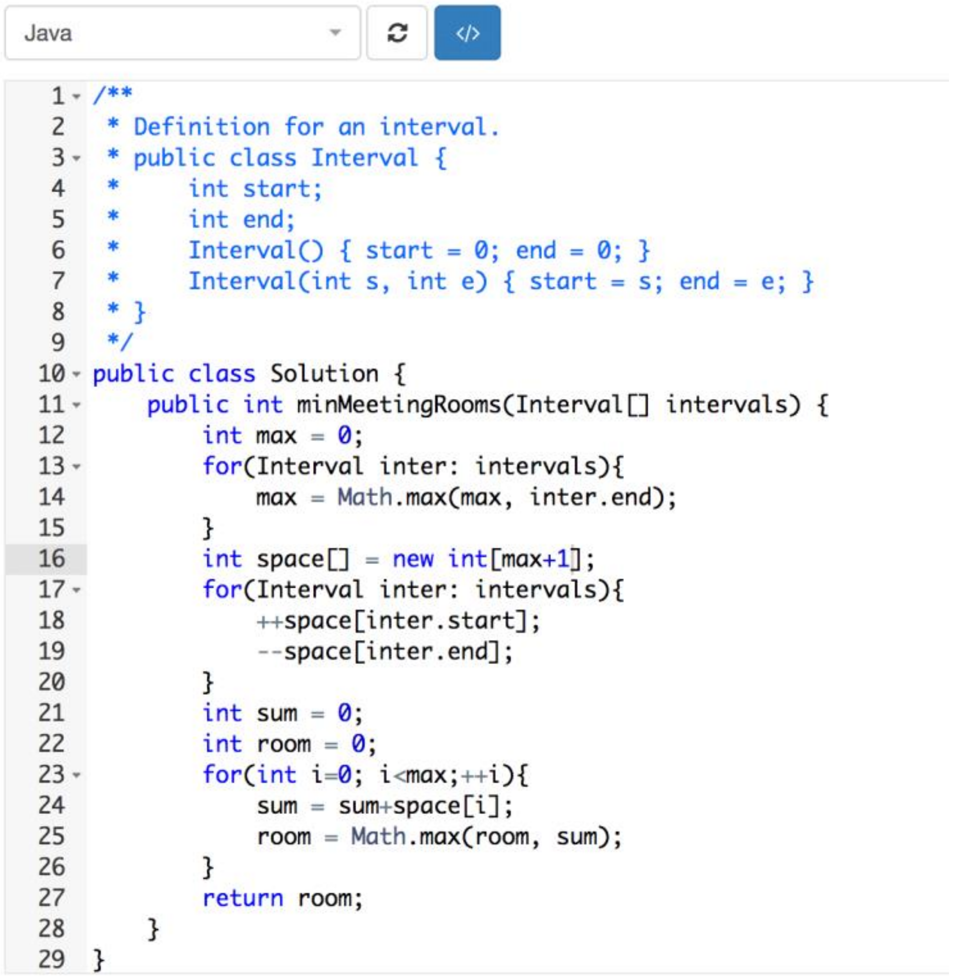
Given an array of meeting time intervals consisting of start and end times `[[s1,e1],[s2,e2],...]` ($s_i < e_i$), find the minimum number of conference rooms required.

For example,

Given `[[0, 30],[5, 10],[15, 20]]`,

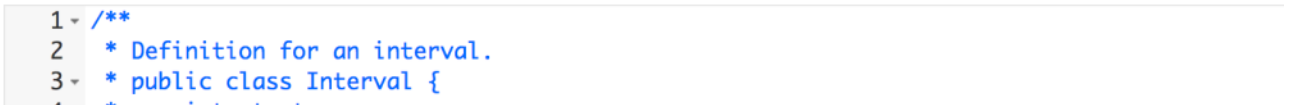
return `2`.

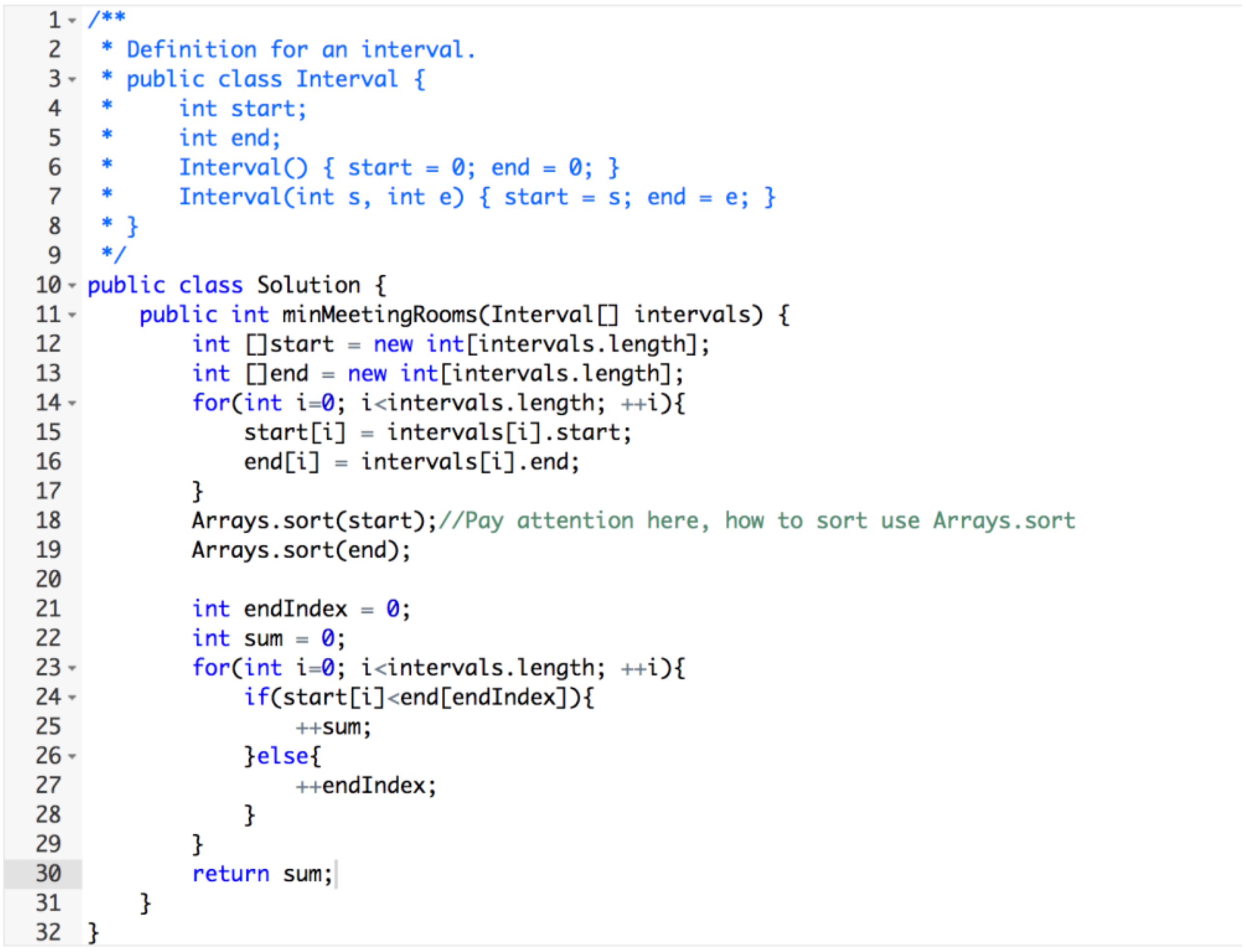
扫描线算法


Java

```java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public int minMeetingRooms(Interval[] intervals) {
        int max = 0;
        for(Interval inter: intervals){
            max = Math.max(max, inter.end);
        }
        int space[] = new int[max+1];
        for(Interval inter: intervals){
            ++space[inter.start];
            --space[inter.end];
        }
        int sum = 0;
        int room = 0;
        for(int i=0; i<max;++i){
            sum = sum+space[i];
            room = Math.max(room, sum);
        }
        return room;
    }
}
```

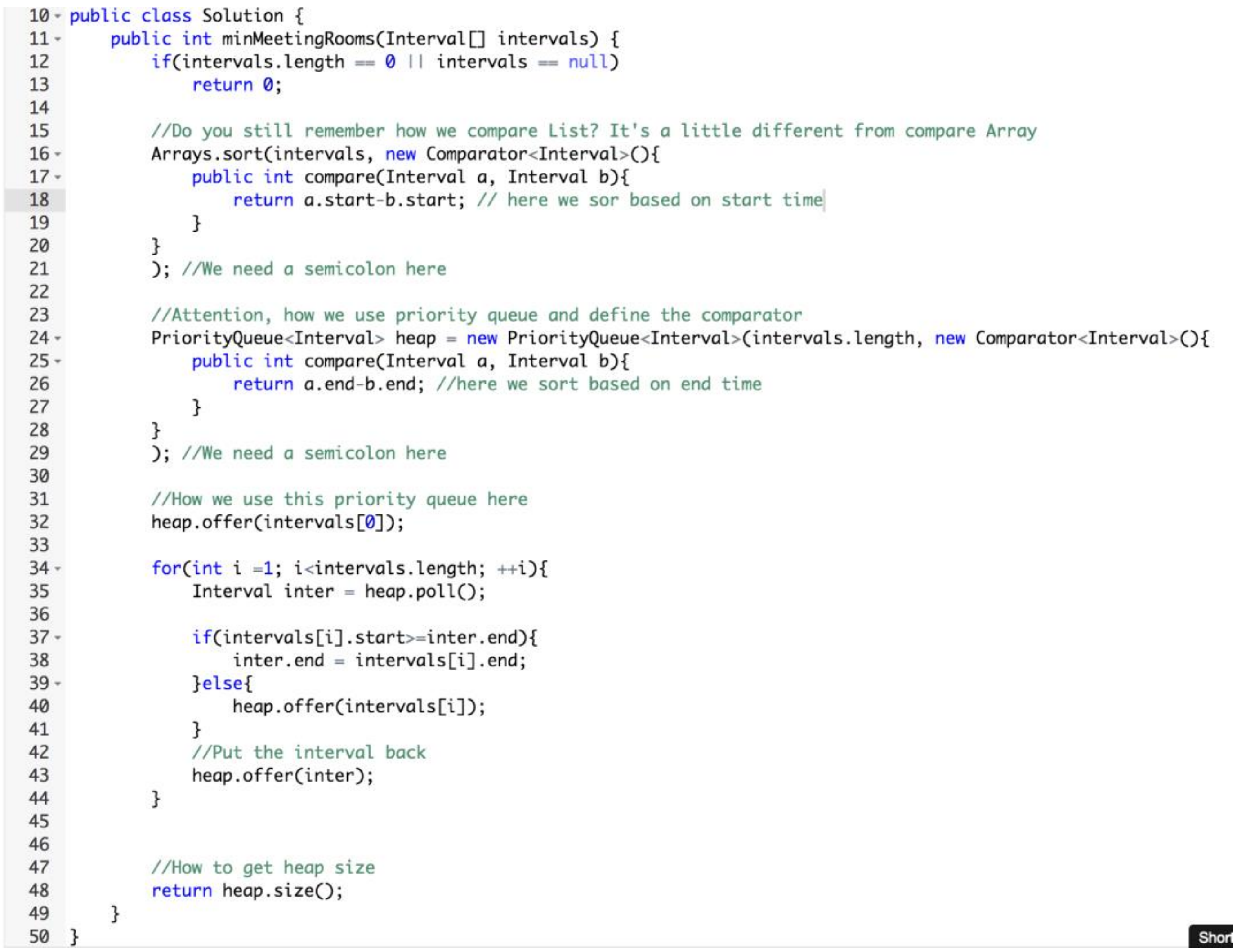
不用Priority queue的算法


```java
/**
 * Definition for an interval.
 * public class Interval {
```



```java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public int minMeetingRooms(Interval[] intervals) {
        int []start = new int[intervals.length];
        int []end = new int[intervals.length];
        for(int i=0; i<intervals.length; ++i){
            start[i] = intervals[i].start;
            end[i] = intervals[i].end;
        }
        Arrays.sort(start);//Pay attention here, how to sort use Arrays.sort
        Arrays.sort(end);

        int endIndex = 0;
        int sum = 0;
        for(int i=0; i<intervals.length; ++i){
            if(start[i]<end[endIndex]){
                ++sum;
            }else{
                ++endIndex;
            }
        }
        return sum;
    }
}
```

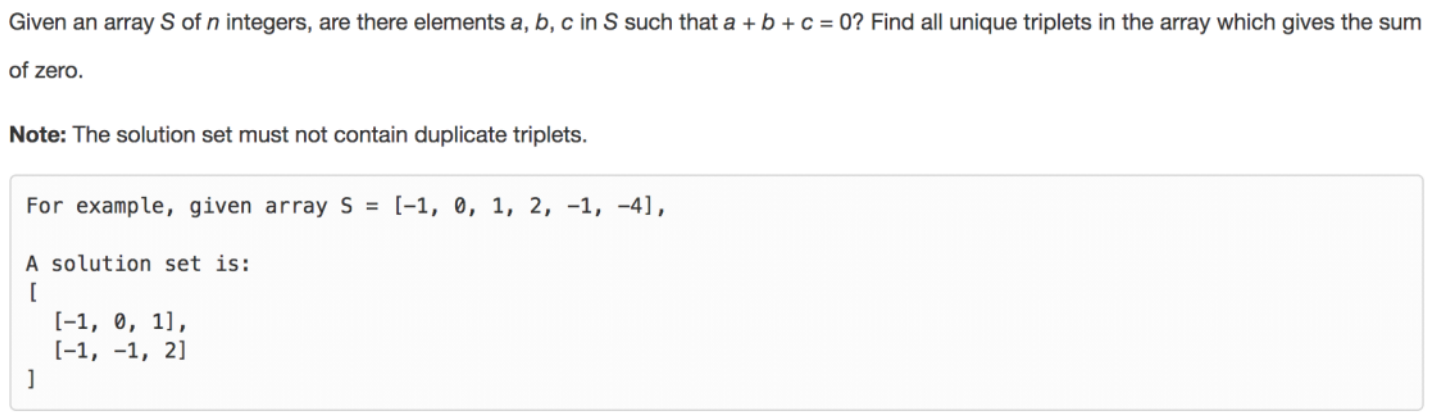
用Priority Queue的算法



```java
public class Solution {
    public int minMeetingRooms(Interval[] intervals) {
        if(intervals.length == 0 || intervals == null)
            return 0;

        //Do you still remember how we compare List? It's a little different from compare Array
        Arrays.sort(intervals, new Comparator<Interval>(){
            public int compare(Interval a, Interval b){
                return a.start-b.start; // here we sor based on start time
            }
        }
        ); //We need a semicolon here

        //Attention, how we use priority queue and define the comparator
        PriorityQueue<Interval> heap = new PriorityQueue<Interval>(intervals.length, new Comparator<Interval>(){
            public int compare(Interval a, Interval b){
                return a.end-b.end; //here we sort based on end time
            }
        }
        ); //We need a semicolon here

        //How we use this priority queue here
        heap.offer(intervals[0]);

        for(int i =1; i<intervals.length; ++i){
            Interval inter = heap.poll();

            if(intervals[i].start>=inter.end){
                inter.end = intervals[i].end;
            }else{
                heap.offer(intervals[i]);
            }
            //Put the interval back
            heap.offer(inter);
        }


        //How to get heap size
        return heap.size();
    }
}
```

15. 3Sum

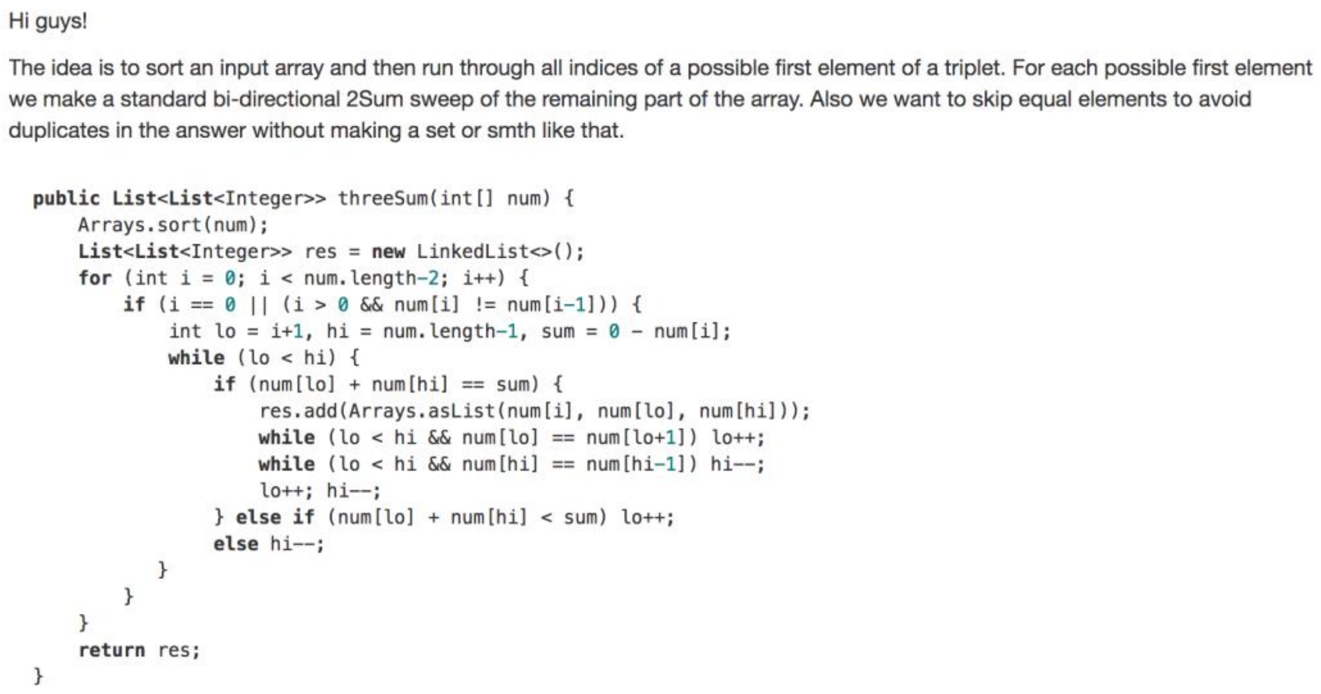
Thursday, February 2, 2017 6:04 PM



Given an array *S* of *n* integers, are there elements *a*, *b*, *c* in *S* such that *a* + *b* + *c* = 0? Find all unique triplets in the array which gives the sum of zero.

Note: The solution set must not contain duplicate triplets.

```
For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```



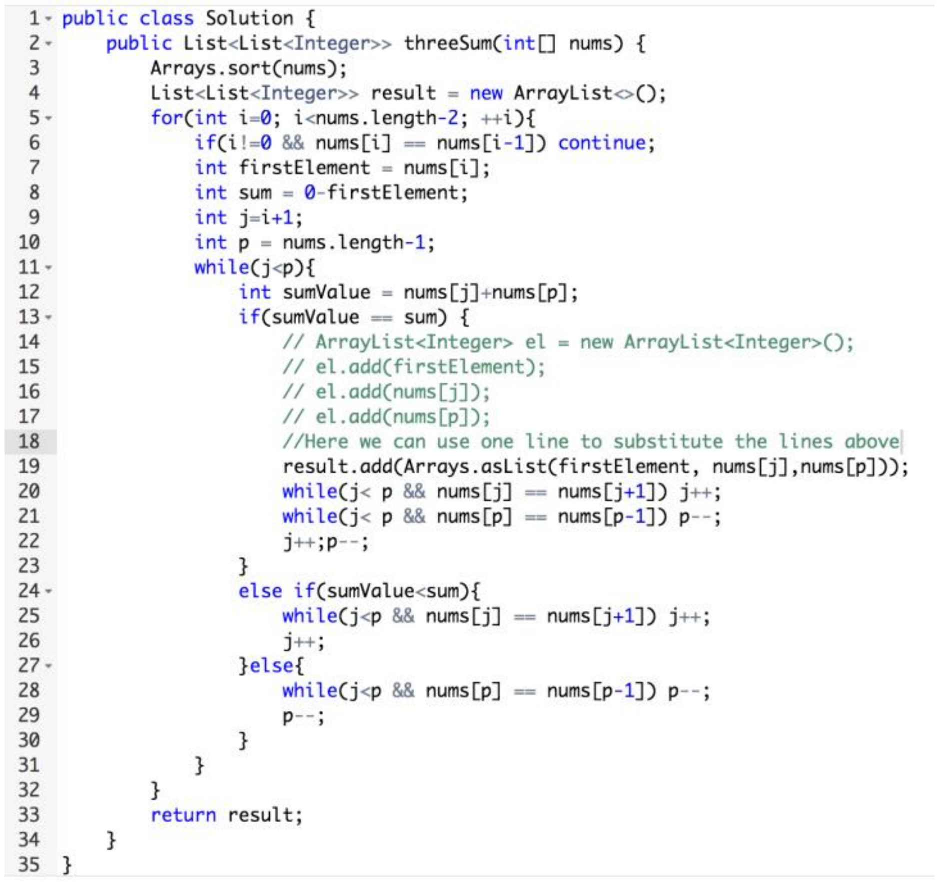
Hi guys!

The idea is to sort an input array and then run through all indices of a possible first element of a triplet. For each possible first element we make a standard bi-directional 2Sum sweep of the remaining part of the array. Also we want to skip equal elements to avoid duplicates in the answer without making a set or smth like that.

```
public List<List<Integer>> threeSum(int[] num) {
    Arrays.sort(num);
    List<List<Integer>> res = new LinkedList<>();
    for (int i = 0; i < num.length-2; i++) {
        if (i == 0 || (i > 0 && num[i] != num[i-1])) {
            int lo = i+1, hi = num.length-1, sum = 0 - num[i];
            while (lo < hi) {
                if (num[lo] + num[hi] == sum) {
                    res.add(Arrays.asList(num[i], num[lo], num[hi]));
                    while (lo < hi && num[lo] == num[lo+1]) lo++;
                    while (lo < hi && num[hi] == num[hi-1]) hi--;
                    lo++; hi--;
                } else if (num[lo] + num[hi] < sum) lo++;
                else hi--;
           }
        }
    }
    return res;
}
```

先排序，排序后两层遍历。第一层从头扫到尾，第二层有两个指针，一个头一个尾，往中间缩。因为是排序后的数组，所以这样bi-direction可以避免triple重复。而且如果原数组中有重复的数字，只要内层的指针相应的加减跳过即可。

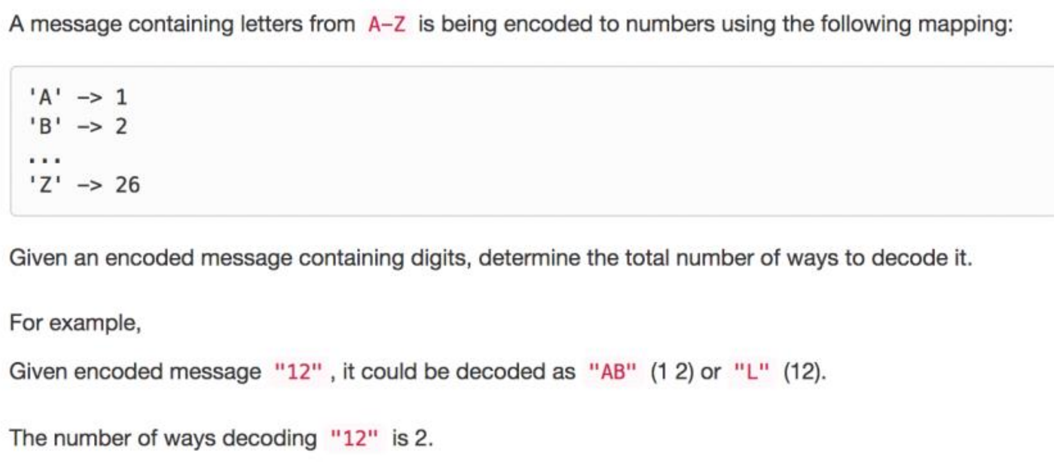
1. 排序数组自带函数是： Arrays.sort(nums);
2. 初始化List时，是这么初始化的：List<List<Integer>> myList = new LinkedList();
3. 要想将一个array转化为list，直接Arrays.asList(num) 即可



```java
public class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> result = new ArrayList<>();
        for(int i=0; i<nums.length-2; ++i){
            if(i!=0 && nums[i] == nums[i-1]) continue;
            int firstElement = nums[i];
            int sum = 0-firstElement;
            int j=i+1;
            int p = nums.length-1;
            while(j<p){
                int sumValue = nums[j]+nums[p];
                if(sumValue == sum) {
                    // ArrayList<Integer> el = new ArrayList<Integer>();
                    // el.add(firstElement);
                    // el.add(nums[j]);
                    // el.add(nums[p]);
                    //Here we can use one line to substitute the lines above
                    result.add(Arrays.asList(firstElement, nums[j],nums[p]));
                    while(j< p && nums[j] == nums[j+1]) j++;
                    while(j< p && nums[p] == nums[p-1]) p--;
                    j++;p--;
                }
                else if(sumValue<sum){
                    while(j<p && nums[j] == nums[j+1]) j++;
                    j++;
                }else{
                    while(j<p && nums[p] == nums[p-1]) p--;
                    p--;
                }
            }
        }
        return result;
    }
}
```

91. Decode Ways

Saturday, February 18, 2017 10:26 PM



A message containing letters from A-Z is being encoded to numbers using the following mapping:

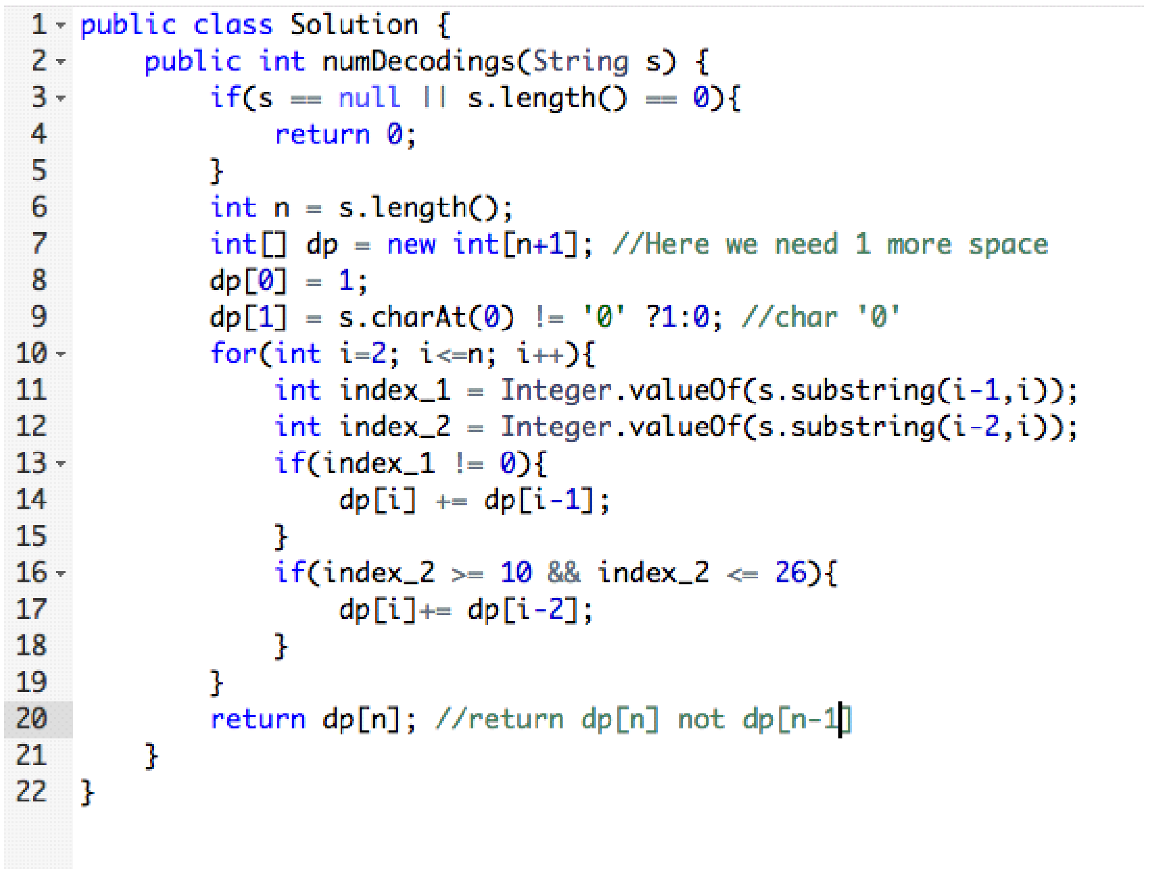
```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,

Given encoded message "12" , it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.



```java
public class Solution {
    public int numDecodings(String s) {
        if(s == null || s.length() == 0){
            return 0;
        }
        int n = s.length();
        int[] dp = new int[n+1]; //Here we need 1 more space
        dp[0] = 1;
        dp[1] = s.charAt(0) != '0' ?1:0; //char '0'
        for(int i=2; i<=n; i++){
            int index_1 = Integer.valueOf(s.substring(i-1,i));
            int index_2 = Integer.valueOf(s.substring(i-2,i));
            if(index_1 != 0){
                dp[i] += dp[i-1];
            }
            if(index_2 >= 10 && index_2 <= 26){
                dp[i]+= dp[i-2];
            }
        }
        return dp[n]; //return dp[n] not dp[n-1]
    }
}
```