/\*

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].

way-1 : 先把新的插入，然后合并用merge

way-2:不排序，直接插入。分别处理好 插入左边，插入中间，插入右边的情况就行！

\*/

/\*\*

\* Definition for an interval.

\* struct Interval {

\* int start;

\* int end;

\* Interval() : start(0), end(0) {}

\* Interval(int s, int e) : start(s), end(e) {}

\* };

\*/

bool comp(Interval &a,Interval &b)

{

return a.start<b.start;

}

class Solution {

public:

vector<Interval> insert(vector<Interval>& intervals, Interval newInterval)

{

//way-1

/\*

intervals.push\_back(newInterval);

return merge(intervals);

\*/

//way-2

vector<Interval> ret;

int k=0;

while(k<intervals.size() && intervals[k].end<newInterval.start)

ret.push\_back(intervals[k++]);

ret.push\_back(newInterval);

while(k<intervals.size() && intervals[k].start<=ret[ret.size()-1].end)

{

ret[ret.size()-1].start=min(ret[ret.size()-1].start,intervals[k].start);

ret[ret.size()-1].end=max(ret[ret.size()-1].end,intervals[k].end);

k++;

}

while(k<intervals.size())

ret.push\_back(intervals[k++]);

return ret;

}

vector<Interval> merge(vector<Interval>& intervals)

{

sort(intervals.begin(),intervals.end(),comp);

vector<Interval> ret;

ret.push\_back(intervals[0]);

for(int i=1;i<intervals.size();i++)

{

if(intervals[i].start<=ret[ret.size()-1].end)

ret[ret.size()-1].end=max(ret[ret.size()-1].end,intervals[i].end);

else

ret.push\_back(intervals[i]);

}

return ret;

}

};