CS577
Homework #4, Problem #3
Sreeram Danda

**a)**
```
int find_interval(intervals[] ){
     //First sort intervals by increasing order based on end value
     sorted_intervals <- mergeSort(intervals[]);
     points[];

     //First point is endpoint for first interval
     point <- sorted_intervals[0].end_interval();
     points.add(point);

     //Iterate over array once and add points if start time is less
     //than current end time
     for(x = 1; x < size of interval; x++){
          if(sorted_intervals[x].start_interval() > point){
               point <- sorted_interval[x].end_interval();
               points.add(point);
          }
     }
     return points;
}
```

**b)**
Runtime of the algorithm is O(n log(n)).

**c)**
Program Correctness: The first thing the algorithm does is put all the intervals in sorted order based on their ending value. Then the algorithm iterates through the sorted intervals one time adding a value to the return array only if the starting point of the current interval is greater than the last point added to the return value. This is correct behavior because an interval having a start value greater than the last end value guarantees that there is no overlap between the two intervals and thus requires an additional point. Otherwise, the only cases possible are the start value being less than the last value in which case we can use the current point value as it is contained within the interval. If the start value equals the point then we do the same. Additionally, we know that a sorted list will have intervals such that the next interval always has an end value greater than the previous interval. The only case where a new value will be needed is if there is no overlap between two intervals. The only way to have this case in this sorting is to have a start value greater than the last interval.

Runtime: The first line of the algorithm makes a call to mergesort. The merge sort algorithm has a runtime of O(n log(n)). The next three lines consist of assignment and initialization statements that take

O(1). Then the algorithm makes an iteration over (n - 1) items in the array which takes O(n) time. Adding these together: O(n log(n)) + O(1) + O(n) = O(n log(n).