Student Name: Yinsheng Dong

Student Number: 11148648

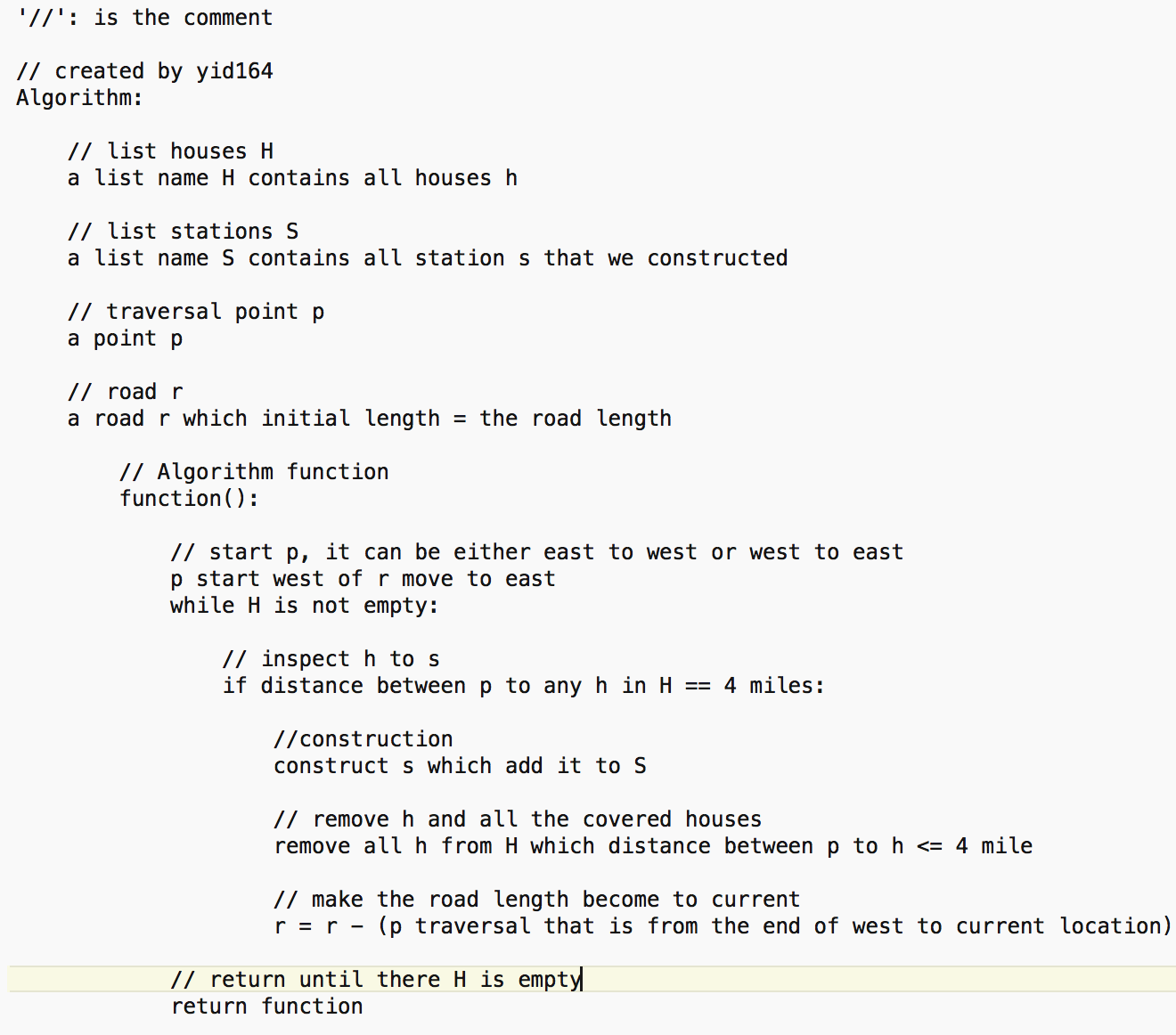
NSID: yid164

CMPT360 Assignment 2

Q1*.* Kleinberg and Tardos p.190 #5

Solution:

**Algorithm:**



**Explanation of why it is correct:**

Proof by induction:

From the case in this question, the worst case in the problem is that for covering n houses, we need to construct k stations which k = n

For the general case in this question, we might need to construct k station to cover n houses, k < n,

Base case:

We assume that in situation: the worst case: in the algorithm I implemented, there are k elements in S, and n elements in H, k = n

Inductive Step:

case 1: we add one more element h to H, and the distance between h an any element of S <=4: the algorithm will remind the k elements in S, so we have k < n

case 2: we add one more element h to H, and the distance between h and any element of S > 4 the algorithm will add one more s to S, so we have k = n

**Conclusion:**

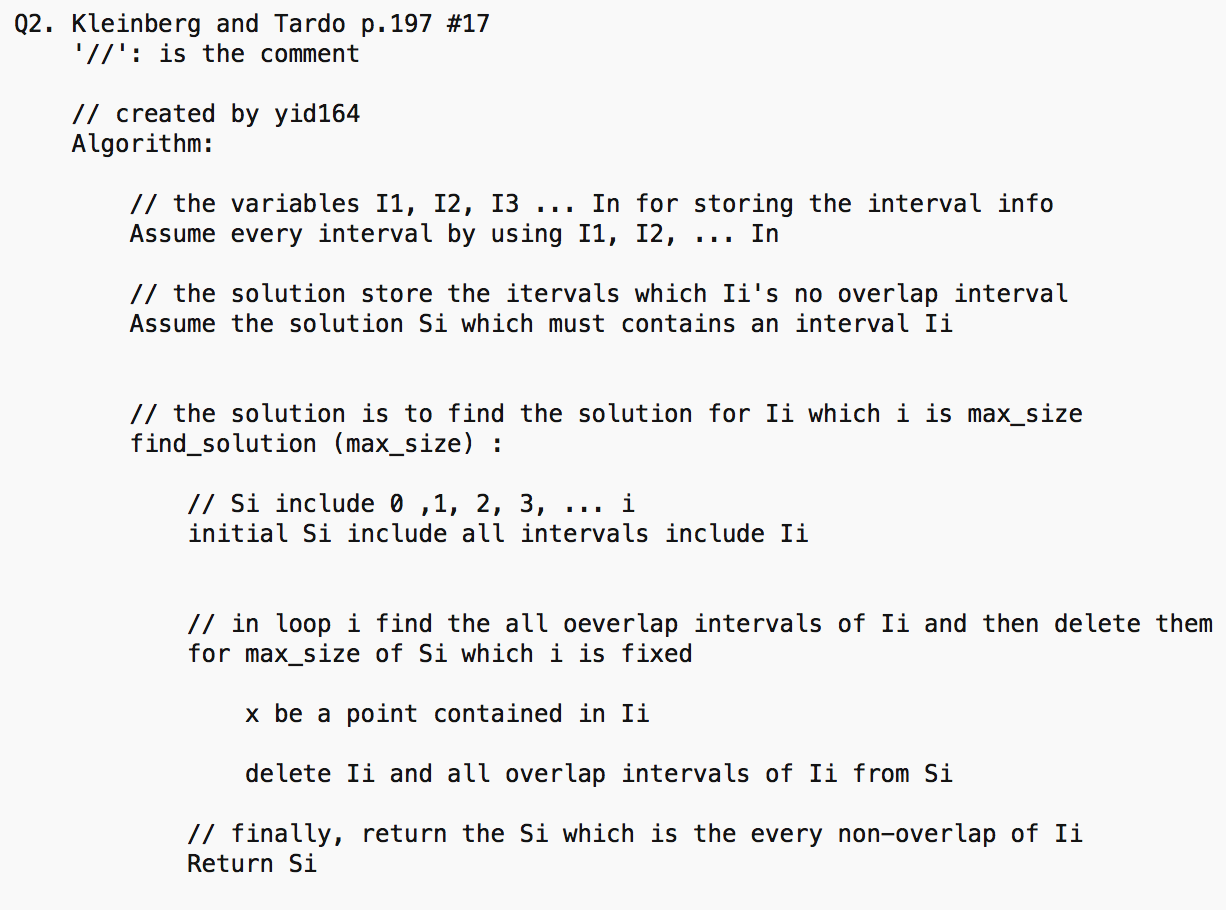
The algorithm is correct since k <= n that I proved

Q2. Kleinberg and Tardo p.197 #17

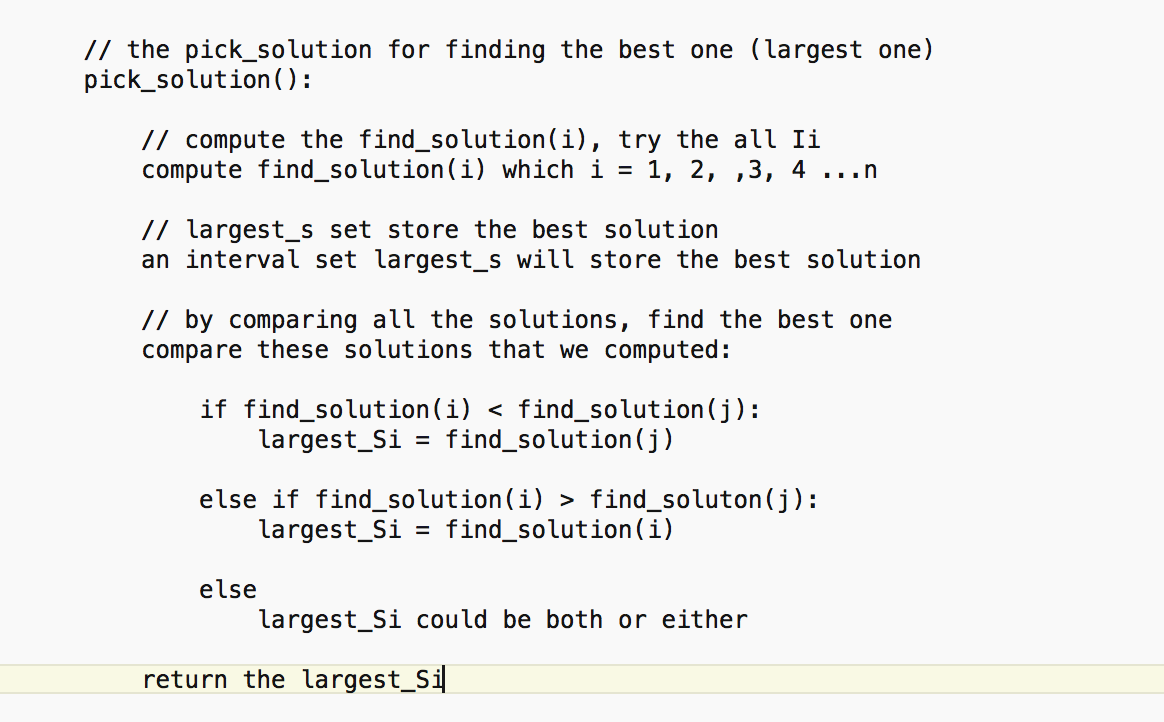
Solution:

**Algorithm:**

**In next page**

****

This function is using to find the potential solution for an interval, and it will return the Si which contains all the non-overlap of Ii

****

This function is looking for the best (largest) solution that intervals contained.

**Explanation of why it is correct:**

By this question, we know that the number of intervals (tasks) will not be 0 or less than 0**,** because if it is 0, CPU takes nothing, or if it is less than 0, it would be an error.

Then we assume that the problem has n intervals(tasks) which n > 0, and a set of S which contains all intervals.

Then pick\_solution() will compare all the potential solutions which find\_solution() produced .

Then largest\_Si will be produced by the pick\_solution()

**Conclusion:**

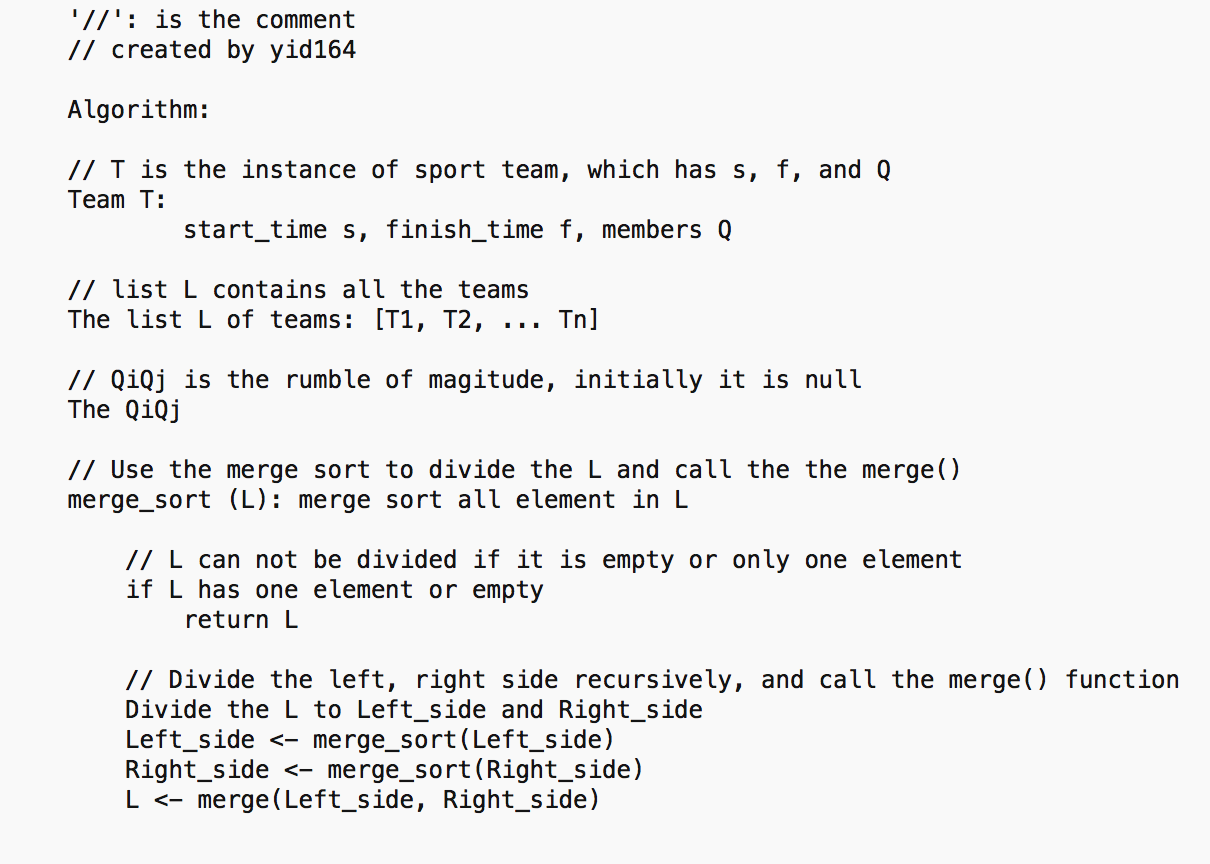
Since find\_solution() produces the potential Si and pick\_solution() gives the largest Si, so it should be correct.

Time-Complexity: find\_solution(): O(n) and pick\_solution(): O(n^2)

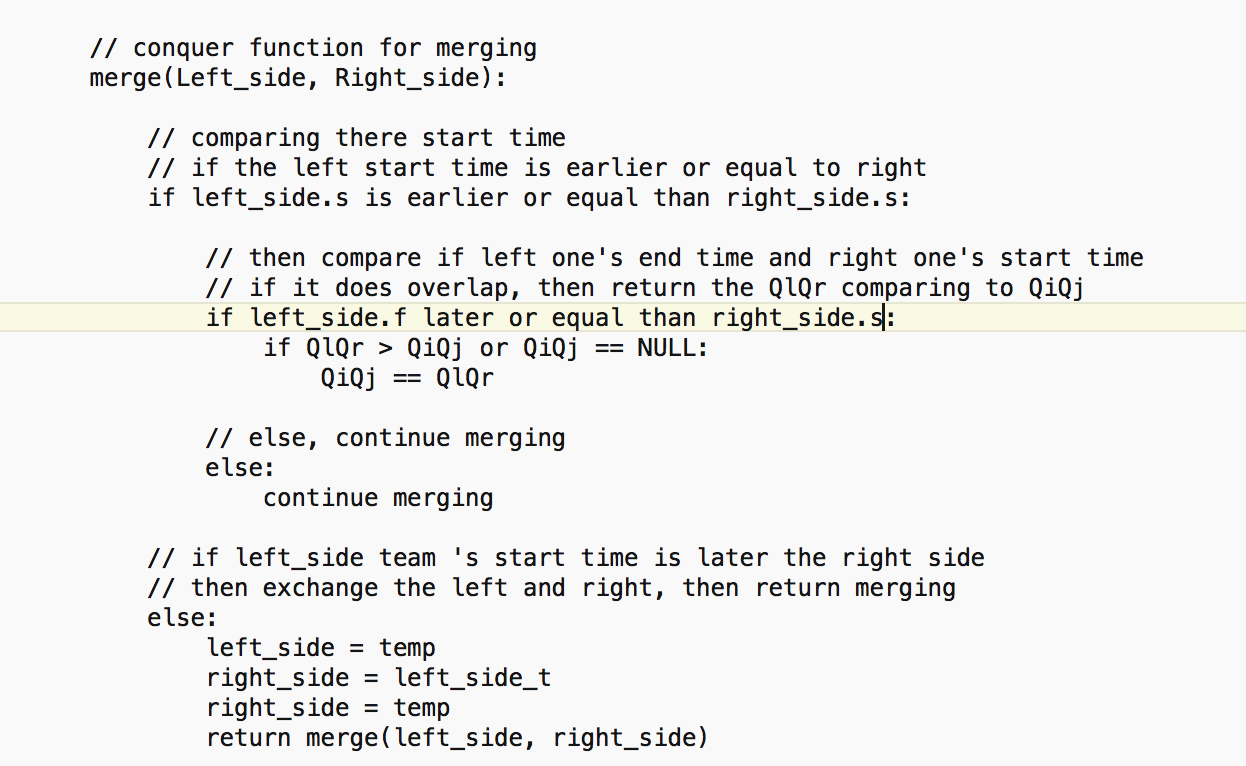
Total: O(n^2)

Q3.

**Algorithm:**

****

This merge\_sort() function is using to divide the list in 2 part (left and right) recursively.



The merge() function is to merge all left and right teams to compare their start time and end time to check if their overlap or not, then to compare all the rumble of moganite, to get the best one.

**The running time of the algorithm:**

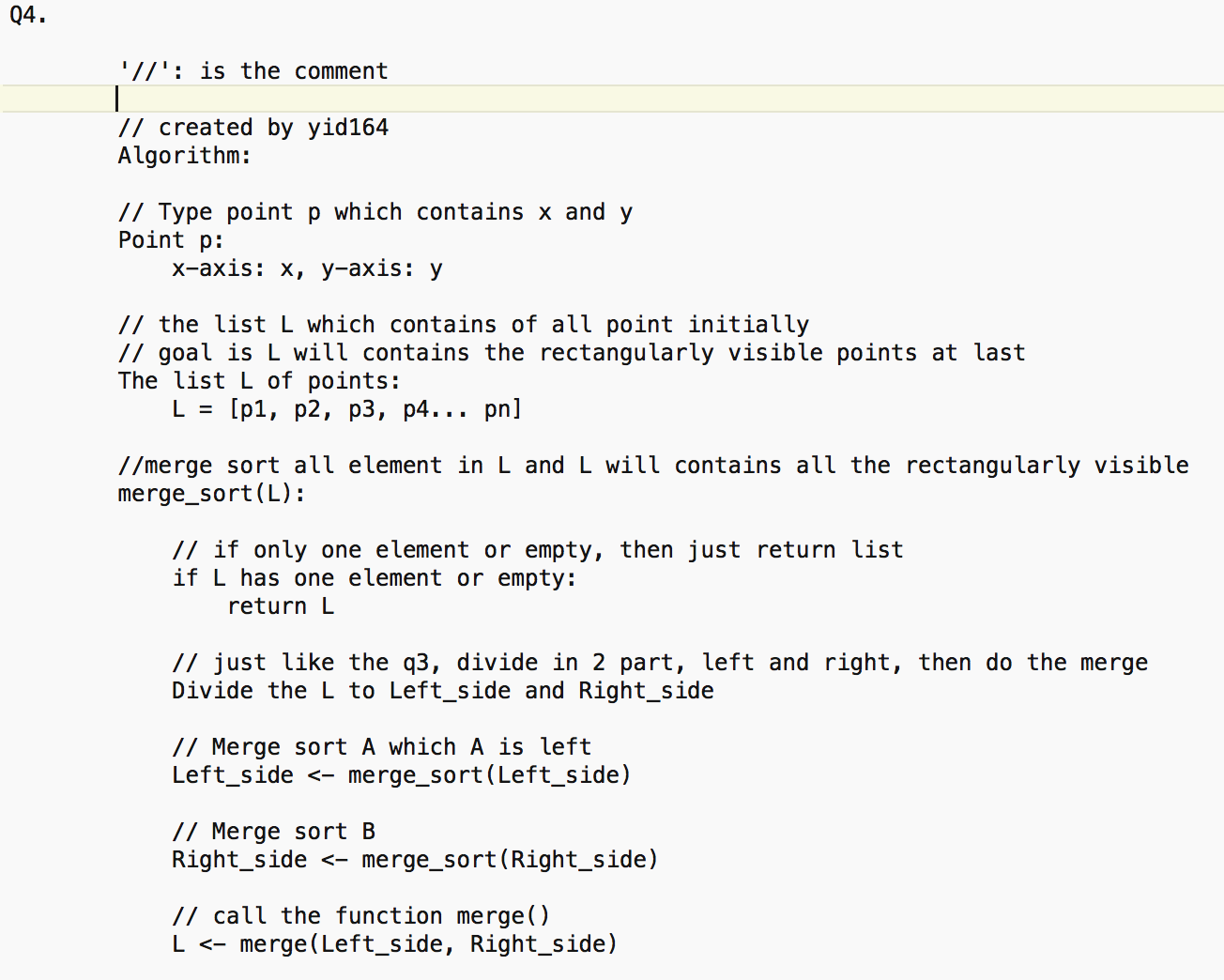
T(n) = 2T(n/2) + cn which c is the constant depends on how many teams in list

Time complexity is O(nlog(n))

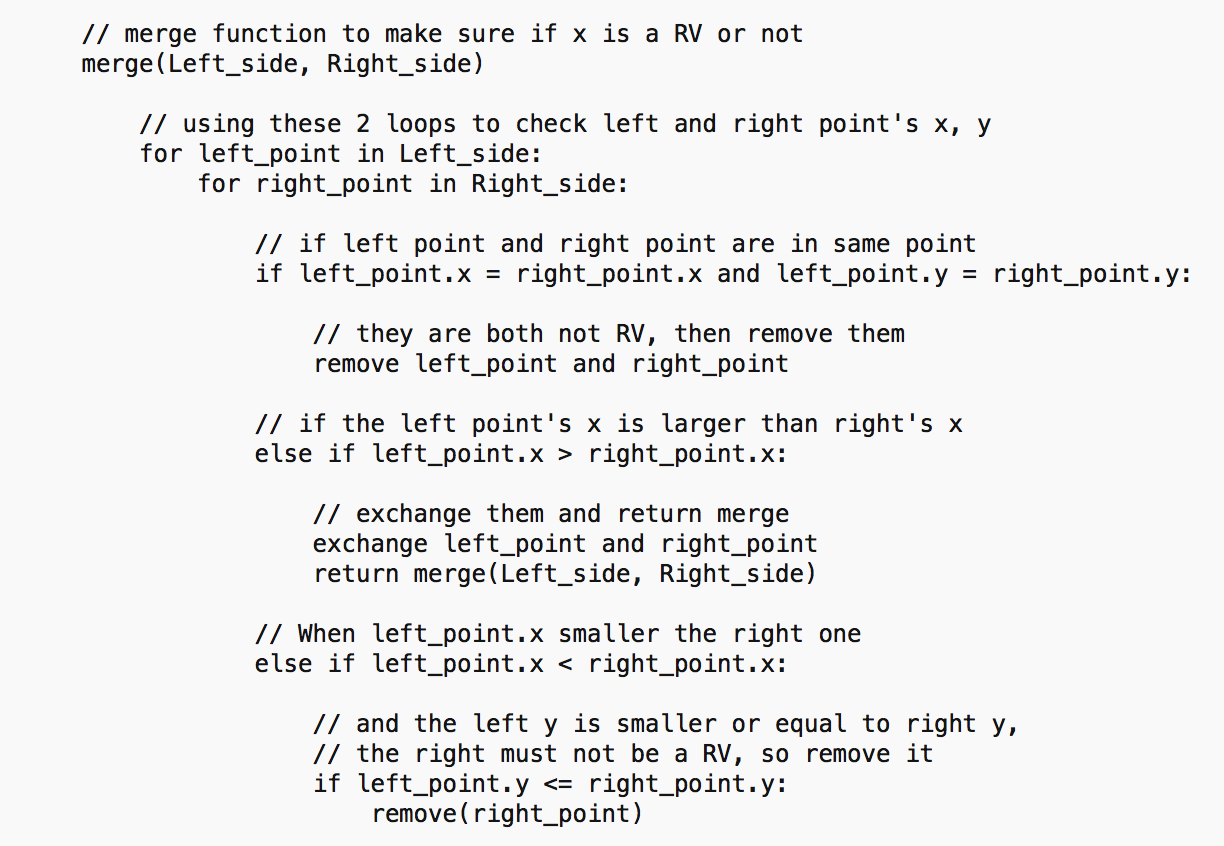
Q4.

**Algorithm:**

**In next page**

****

The merge\_sort() function is using to divide the given list L which L is contains all the points {p1, p2, p3, … pn} in 2 part (left and right) recursively, and calling merge() to calculate

****

The merge() function is using 2 loops to find eligibility by checking left and right’s x, y.

If right’s x is smaller than left’s, exchange them to guarantee the left x is smaller than right’s x.

If we got the left.x is still smaller than right.x, if the right.y is larger than left.y, the right must not be the RV, then we delete it recursively, until L only contains the RV.

**Recurrence for the time:**

merge\_sort() = log(n),

merge() = n^2,

T(n) = log(n) \* n^2

Time complexity: O(n^2)