Q2. a)

Before we present our algorithm, we would like to make a few clarification on the terms used in the algorithm. The input A is the unsorted array which stores the closed intervals in their original order. The input p is the starting index of array A. The input r is the last index of array a.

The element stored at A[i] is [ai , bi]. Please be noted that our algorithm sorts the intervals by sorting ai and our algorithm will return an array with n closed intervals [ai , bi] for i = 1, 2, 3, 4,…n where i < j, ai < aj. The permutation <i1, i2, i3, i4, …, in> in our case is <1, 2, 3, 4,…, n>.

Following algorithms are cited from CLRS Quicksort.

Randomized-Partition(A, p, r)

i = Random(p, r)

exchange A[r] with A[i]

return Fuzzy-Partition(A, p, r)

Randomized-Fuzzysort(A, p, r)

If p < r

q = Randomized-Partition(A, p, r)

Randomized-Fuzzysort(A, p, q-1)

Randomized-Fuzzysort(A, q+1, r)

Fuzzy-Partition(A, p, r)

{

X = A[r][0] - ai  of [ai , bi]

Y = A[r][1] - bi  of [ai , bi]

i = p-1

for j = p to r-1

{do if A[j][0] < X and A[j][1] < Y

then {

i = i +1

exchange A[i] <-> A[j]

}

}

exchange A[i+1] <-> A[r]

return i+1

}

b) The algorithm is quite similar to the algorithm used by Randomized-Quicksort in the textbook except there is a change in the if condition in the Partition step. The reason for this change if because now we are sorting intervals and if [a1, b1] and [a2, b2] have overlap interval, there is no need to sort these two intervals. Therefore, the new if condition will highly decrease the time complexity if the n closed intervals we are sorting have a lot intervals overlap with each other. If we are sorting n closed intervals and there is zero pair of intervals overlap with each other, then the expected time is the same as Randomized-QuickSort which is Θ (nlogn). If we are sorting n closed intervals when all intervals overlap with each other, our algorithm would only need to compare the a with X (X is the aof the last interval of unsorted array) for all the intervals except the last interval. The algorithm would not run recursively. Therefore, the expected time for this case is O(n). Now we need to show the time is ω(n). Consider one input array with [[a1, b1], [ a2, b2], [ a3, b3],..., [an, bn]] where a1< a2< a3< ...< an. Since [an, bn] is the pivot interval, our algorithm have to compare the ai of all the intervals before [an, bn] with an. Therefore the running time is ω(n). Combine with the conclusion that expected time for this case is O(n), the expected time is Θ(n).