CS 4820, Spring 2017 Homework 5, Problem 2

Name: Yuxiang Peng

NetID: yp344

Collaborators: jl3455, zl542

(2a) (5 points) Design an algorithm that processes a list of n intervals

$$I_1 = (s_1, f_1),\ I_2 = (s_2, f_2), \dots, I_n = (s_n, f_n)$$

and outputs the number of ordered pairs (I_i, I_j) such that I_i is a subinterval of I_j. You should assume that the 2n numbers $s_1, \dots, s_n, f_1, \dots, f_n$ are all distinct. Do not assume the input is sorted in any particular order. Your algorithm's running time should be O(nlogn).

Solution:
L: the sequence of intervals.
n: number of intervals
Sort the intervals in increasing start time order such that S1 < S2 < ... < Sn. O(n log n)

```
Algorithm(Sort-and-count): If
n=1
.        return 0, and L
else
.        A< − first n/2 elements of L
.        B< − remaining n/2 elements of L
.        (na,A)< − Sort-and-count(A)
.        (nb,B)< − Sort-and-count(B)
.        (nab,L)< − Merge-and-count(A,B)
return na+nb+nab, L

Merge-and-count(A,B)
La=length(A)
Lb=length(B)
ia=La
ib=Lb
Count=0
C=empty set
while(ia >= 1 and ib >= 1)
.        If f[ib] in B<f[ia] in A
.              C=C appended with A[ia] in front.
.              ia=ia-1
.              Count=Count+ib
.        else
.              ib=ib-1
.              C=C appended with B[ib] in front
Append remainder of A or B to C in front.
Return Count and C
```

Runtime:

The initial sorting time is O(nlogn). Works in each level j of a n-level recursion requires time of $2^j * C * n/2^j$ for some constant C. According to the chapter, the total time should be O(nlogn)+O(C*nlogn)=O(nlogn).

Correctness:
The base case is 1 interval whose returning value is 0 which is also right. As for the combination of merge and count, the total number of satisfied pairs equals to the number of pairs from each subset and those across the two intervals. So it outputs the right number of satisfied pairs. And the recursion works to return this correct value to the upper level and continue merge and count.

(2b) (10 points) The input is the same as in part (a), but now your algorithm must output the number of ordered pairs $(\mathbf{I_i}, \mathbf{I_j})$ such that $\mathbf{I_i}$ is a subinterval of $\mathbf{I_j}$ and $\mathbf{i} < \mathbf{j}$. Your algorithm's running time should be $O(n^2)$, meaning that if T (n) is the worst-case running time on inputs of size n, then $\lim(T(n)/n^2) = 0$.

Solution:
L: the sequence of intervals.
n: number of intervals

```
Algorithm(Sort-and-count):
If  n=1
.         return 0, and L
else
.         A< − first n/2 elements of L
.         B< − remaining n/2 elements of L
.         (na,A)< − Sort-and-count(A)
.         (nb,B)< − Sort-and-count(B)
.         (nab,L)< − Merge-and-count(A,B)
return na+nb+nab, L

Merge-and-count(A,B)
La=length(A)
Lb=length(B)
construct a balanced binary search tree BBST for B with one additional field for each node that records
the number of nodes with start time less than its start time
ia=1
ib=1
Count=0
C=empty set
while(ia <= La and ib <= Lb)
.         If f[ia] in A>f[ib] in B
.                  C=C appended with B[ib] at the back.
.                  delete B[ib] from the BBST
.                  ib=ib+1
.         else
.                  find the number of nodes(nc) with start time less than s[ia] in A by inserting A[ia] into
the BBST
.                  Count=Count+nc
.           delete A[ia] from the BBST
.           ia=ia+1
.             C=C appended with A[ia] at the back
Append remainder of A or B to C at the back.
Return Count and C
```

C in merge-and-count outputs the set of intervals in ascending finish time order.
The balanced BST helps us efficiently delete and insert nodes in running time of O(nlogn) and find the number of nodes with a starting time smaller than a particular starting time whose running time is also O(nlogn) in the process of merge and count. Then we use a pointer to traverse the left branch from the smallest finish time and compare with the smallest finish time interval on the right branch with another pointer as described in the algorithm. The worst case is of 2*n iterations in the traversal loop and 2*n deletions and n insertions.

Runtime:
Recursion has log n levels. For Merge and count, it takes O(nlogn) to construct the BST, and O(C*nlogn) for the loop for some constant C. Work in each level j thus is $2^j * C * n/2^j * \log(n/2^j)$ which is bounded by $C * n\log(n)$ for some constant C. Total time is $C * n(\log n)^2 = O(n(\log n)^2)$ which satisfies $o(n^2)$.

Correctness:
The base case is 1 interval whose returning value is 0 which is also right. As for the combination of merge and count, the total number of satisfied pairs equals to the number of pairs from each subset and those across the two intervals. So it outputs the right number of satisfied pairs. And the recursion works to return this correct value to the upper level and continue merge and count.