I collaborated with: Nevin Bernet

Notes to Instructor or TA: If needed, include here any special notes for TAs or instructor; delete if no notes

Problem

Free-style: Suppose you are given a set S of n intervals—that is, n pairs $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ of numbers with each $s_i < t_i$.

(a) Design an algorithm to find the maximum sized subset of S such that every pair of intervals in the subset overlap and prove that your algorithm works correctly in all cases.

Suggestion: Draw some pictures. See if you can identify any helpful properties that hold for any subset of intervals such that all of the intervals in the subset intersect pairwise (if you use such a property, though, you must prove that it holds!). Did I say "Draw some pictures"? Yes. Yes, I did. Imagine processing the intervals in, say, left-to-right order.

(b) Describe an implementation of your algorithm, including any appropriate data structures, and determine its time and space requirements. Pseudo-code can be helpful, but is insufficient on its own (and is also not required). A clear, concise prose description of an algorithm should be considered a necessity for your response to be complete.

Solution

Algorithm 1 Biggest Interval

Require: Set S

```
S* ← empty set
Result ← S*
A ← array
Put s_i and t_i in A ∀ i  1 ≤ i ≤ n
Sort A
for i ← 0 to size of A do
    e ← A[i]
    if e is a starting time then
        add e to S*
    else
        Result ← S* only if S* > Result
        remove s_i that corresponds with e from S*
    end if
end for
return Result
```

a) If S is empty then the algorithm returns an empty set. Consider first instance of t_i in A. $S*$ holds all of the intervals that overlap with the first s added to $S*$. At this point, $S*$ can either hold the largest subset of overlapping intervals or not. If it does, then $Result$ will take the value of $S*$. If it does not, then $Result$ will keep its old value until a larger $S*$ appears. s_i will be removed from $S*$ and the algorithm repeats, adding overlapping intervaks to $S*$ and comparing the set to $Result$.
b) The pairs s_i and t_i will each be objects that are able to hold their index i as an instance variable, and point to their value pair. The Set $S*$ will be a linked list where adding to the front is constant time while looking up the corresponding s_i will be linear time. The sorting of A will occur in $O(n \log n)$ time. As we walk through A, we add the starting numbers to $S*$ and if we see an end number we look for the corresponding starting time and remove it. Before removing, we have to check if that state of $S*$ is the biggest we have seen thus far.
Time Complexity: Sorting $O(n \log n)$ + Loop $O(n) * O(n) = O(n \log n + n^2) = O(n^2)$
Space Requirement: Array $O(2n)$ + $S*$ $O(n)$ + Result $O(n)$ = $O(4n) = O(n)$