CS302 Midterm 1 Review

1. Create a loop invariant for the loop at Line 7 that will allow you to prove that if A_1 and A_2 are sorted arrays which together contains all elements of A, then the output of the algorithm is A, sorted. Write termination, base case, and maintenance conditions.

Algorithm 1: `MergeSort`(A)

Input : Integer array A of length n
Output: Sorted array

```
   // Base Case
 1 if n == 1 then
 2 |  return A;
 3 end
   // Divide and Conquer
 4 A_1 = MergeSort(A[1 : n/2]);
 5 A_2 = MergeSort(A[n/2 + 1 : n]);
   // Combine
 6 p_1 = p_2 = 1;
 7 for i=1 to n do
 8 |  if A_1[p_1] < A_2[p_2] then
 9 |  |  A[i] = A_1[p_1];
10 |  |  p_1++;
11 |  else
12 |  |  A[i] = A_2[p_2];
13 |  |  p_2++;
14 |  end
15 end
16 return A
```

Solution Invariant:

(a) $A[1 : i-1]$ contains all elements of $A_1[1 : p_1 - 1]$ and $A_2[1 : p_2 - 1]$.

(b) All elements in $A[1 : i-1]$ are less than or equal to any elements in $A_1[p_1 : end]$ and $A_2[p_2 : end]$

(c) $A[1 : i-1]$ is sorted.

Termination: At termination, $i = n + 1$, So A is a sorted array containing all n elements in A_1 and A_2.

Base Case: Initially $A[1:0]$, $A_1[1:0]$ and $A_2[1:0]$ contain no elements, so the conditions are trivially true.

Maintenance: At the beginning of the loop, we can assume, $A[1:i-1]$ contains all elements of $A_1[1:p_1-1]$ and $A_2[1:p_2-1]$ (i). This means both p_1 and p_2 are pointing at new elements that are not yet in A (or are off the end of their array), and by condition (ii), these new elements are larger than or equal to anything that is currently in A. Thus if we put the smaller one into A, and shift the corresponding p_1 or p_2 over, all of the conditions will remain true.

2. Suppose you have n events, each with a start time s_i and end time f_i, for $i \in \{1, \ldots, n\}$. Unfortunately, you only have one auditorium, and you can't schedule conflicting events (events where a start time of one is between the start time and end time of another.) We would like to maximize the number of events that are held. Show that a greedy algorithm that always chooses the next event that has the earliest possible starting time is optimal. You may assume that there is exactly one optimal solution (to make the exchange argument the single exchange type.)

Solution Let σ be the greedy schedule, and for contradiction, suppose σ^* is the optimal schedule. Let i be the first event where σ^* differs from σ. This means that σ^* choose the ith event to be one event, while σ chose a different event with a shorter ending time. Then we can create a new schedule $\sigma^{*\prime}$ that is the same as σ^* except with the ith event the event that σ chose, rather than the event σ^* chose. It is possible to make this scheduling change because σ was able to schedule the event, so it doesn't conflict with any earlier events, but because its ending time is earlier than the event that σ^* chose, it doesn't conflict with any later events in σ^*. But now $\sigma^{*\prime}$ has the same number of events scheduled as σ^*, a contradiction, because we said there is exactly one optimal solution.