Andrew Derringer
Week 3 Homework
CS325 Algorithms Fall 19

1)

Given the scenario my optimal substructure is to look at each day of the trip and to pick a hotel that allows me to complete as many miles of the trip as possible with the hope that that results in the shortest trip time possible. In that case my greedy choice property is to consider some distance d_i the maximum travel distance from my starting location on any given day and to pick a hotel at some distance d_j that minimizes the unused possible travel distance $d_i - d_j$. This greedy method has the potential to determine suboptimal routes by placing us at hotels one night where the next available hotels for the subproblem of them following day are all inefficiently close, leaving excess time spent not traveling, or so far that they cannot be considered. In this scenario I would first be sorting the data before iterating through it which would be my largest complexity factor thus $O(nlog(n))$.

2)

First, I want my jobs sorted in order of penalty from lowest to highest because given that 2 or more jobs have the same deadline, I would rather take the job that will incur the least penalty if the job is not completed on time. This defines my greedy choice property and my optimal subproblem is to assign the lowest penalty option without considering the rest. Now with an empty array defining available time to work and complete jobs I can loop through my sorted array of jobs and assign the first job to the index corresponding to its deadline. If that index is already full then a job with a better penalty has already claimed that time. In an event that there are minutes open not corresponding to a particular deadline, I assign a task do later with the lowest penalty remaining. The array will take nlog(n) time to sort followed by n time to iterate through. Thus nlog(n) + n = $O(nlog(n))$.

3)

Picking the last activity to finish can provide an optimal solution with very similar functionality as determining the first activity to finish. If we have a sorted array by start time we can pick the array item with the latest start time whose end time is within our scope and reduce the scope of our available time to be scheduled based on when the previously picked assignment ends. In this way we loop through or scheduled time as a subproblem that looks to complete a task using as little time as possible and leaving as much time as possible remaining to consider and complete more tasks. The only difference is that in this case we back-load our activities.

4)

My algorithm uses a series of arrays whose index values correspond to those of the other arrays at the same location (using c). First I merge sort (nlog(n)) by start time, making sure to change the values of end time and event number arrays accordingly. Then I can know that I can always begin by adding the last item in the array to the list as it has the latest start time and no conflicts yet. Then I keep tracking of available time by making my limit the start time of the event just added. As I loop through I need only to consider events whose end times are before or equal to the start time of the last event. The first time I reach an event that does not conflict I add it because I know the events are sorted by latest start time and the first non-conflicting event I reach will always have the latest start time. This continues until I reach the end of my available events.

```
// create arrays for eventNumber, startTime, and endTime
//
// create a loop that populates the arrays for one set each loop
//
// mergeSort by startTime but also pass eventNumber and endTime and whenever
// the index of startTime changes, so do the other two in the same way
//
// create dynamic array to hold events that will be added
//
// Always add the last one and change available time to end at start of that event
//
// for loop backwards through sorted events array from the second to last event and
// and only consider event whose end time is before or equal to current availability
// end time
//
// Add the first event that doesn't conflict and adjust availability to start time of the
// newly added event as well as index number for the next event to add
```