Homework #4

1. The greedy algorithm would receive as input a set C of classes with a start time and finish time . The output of the algorithm would result in a non-conflicting schedule with the minimum number of lecture halls as possible. In the algorithm itself, first, the number of lecture halls would be initialized to 0. While the set of classes C is not empty, the class j with the smallest start time will be removed. If there’s a lecture hall for class j, then schedule class j in lecture hall i. Otherwise, increment the number of lecture halls and schedule class j in new lecture hall.

PSEUDOCODE  
sort(classes by start time)  
lecture\_hall = 0  
for i = 1 to n  
 if class i works with a lecture hall j  
 push i in j  
 else  
 lecture\_hall += 1  
 push i to lecture\_hall

The running time of this algorithm is because the set of classes must be sorted by start time.

1. First, the greedy algorithm sorts the jobs in descending order of penalties p and will add them to the timeline in this order. Job is scheduled in the last available time slot that still satisfies the deadline, if available. Otherwise, we can’t do that job.

PSEUDOCODE  
sort(jobs in descending order of penalties)  
T = []  
for i = min(deadlines) to max(deadlines)  
 push [i, []] to T  
 i += 1  
for i=0 to n  
 for j=T.length to 0  
 if deadline[i] <= T[j][0] and T[j][1] == null  
 push job[i] to T[j]

The running time for this algorithm is because it would involve nested loops in order to find the last available time slot for each job.

1. This approach is a greedy algorithm because it is picking the activity with the latest start time that is compatible with all previously selected activities, resulting in one sub-problem in an iterative/recursive fashion. A greedy algorithm always makes the best choice at the moment and assumes it will yield an optimal solution.  
   The following proof was modified/taken from: <https://walkccc.github.io/CLRS/Chap16/16.1/>  
   <http://jt-web-site.tripod.com/MSCS/COMP510/Assignment3/assignment_old.html>  
   Selecting the last activity to start that is compatible with all previously selected activities is the greedy algorithm already shown in 16.1 but reversed.  
   To prove that am is used in some maximum-size subset of mutually compatible activities of Si j. We suppose that Ai j is the maximum-size subset of mutually compatible activities of Si j, also arrange the activities in order of Ai j in monotonically decreasing order of start time. Let ak be the last activity in the Ai j.  
   If ak = am then we do not need to go any further because am is equal with ak that means am is used in the maximum-size subset of mutually compatible activities of Si j.  
   If ak  am then we build the subset A’i j = Ai j – {ak} U {am}. The activities in A’i j are disjoint, since the activities in Ai j are, ak is the last activity in Ai j to start, and sm  sk. Noting that A’i j has the same number of activities as Ai j. That proves A’i j is the maximum-size subset of mutually compatible activities of Si j that includes am.
2. The greedy last-to-start algorithm goes as follows:   
   Sort the activities by start times. Select the activity with the latest start time. Eliminate the activities that could not be scheduled due to conflicts and repeat.

PSEUDOCODE  
  
activitySelector(activityArr)  
 activityNum = 0  
 startTime = 1  
 finishTime = 2  
 mergeSort(activityArr)  
 n = len(activityArr)  
 selectedActivities = activityArr[n][activityNum]  
 i = 1  
 for m = 2 to n  
 if activityArr[m][finishTime] <= activityArr[i][startTime]  
 push activityArr[m][activityNum] to selectedActivities  
 i = m  
 return selectedActivities