Kevin Harvell
CS325-400

Homework #4

1.) The greedy algorithm would receive as input a set C of classes with a start time s_j and finish time f_j. 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 $\Theta(nlogn)$ because the set of classes must be sorted by start time.

2.) First, the greedy algorithm sorts the jobs in descending order of penalties p and will add them to the timeline in this order. Job j_i 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 $\Theta(n^2)$ because it would involve nested loops in order to find the last available time slot for each job.

3.) 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 a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}. We suppose that A_{ij} is the maximum-size subset of mutually compatible activities of S_{ij}, also arrange the activities in order of A_{ij} in monotonically decreasing order of start time. Let a_k be the last activity in the A_{ij}.

If $a_k = a_m$ then we do not need to go any further because a_m is equal with a_k that means a_m is used in the maximum-size subset of mutually compatible activities of S_{ij}.

If $a_k \neq a_m$ then we build the subset $A'_{ij} = A_{ij} - \{a_k\} \cup \{a_m\}$. The activities in A'_{ij} are disjoint, since the activities in A_{ij} are, a_k is the last activity in A_{ij} to start, and $s_m \leq s_k$. Noting that A'_{ij} has the same number of activities as A_{ij}. That proves A'_{ij} is the maximum-size subset of mutually compatible activities of S_{ij} that includes a_m.

4.) 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
```