In class on March 4, I showed that the approach we took for doing dynamic programming for scheduling activities while minimizing the amount of unscheduled time for a room essentially resulted in a brute force solution --- meaning we would try every possible subset of the activities and then select the one with the minimum amount of unscheduled time.

The problem with that approach is that the number of subsets of a set of n activities is 2^n

e.g.
for 3 activities: a1 a2 a3
all subsets are: {empty, {a1}, {a1, a2}, {a1, a3}, {a1,a2,a3}, {a2}, {a2, a3}, {a3}}
the size of that set of subsets is 8 = 2 cubed

So to avoid this (2nd parameter being the list of activities scheduled), I decided we could order the activities in some fashion which allowed us to have only S_ra (equal to the earliest room availability) as 2nd parameter.

Recall that the list of activities scheduled allowed us to tell whether a_i was compatible with the already scheduled activities.

The way I approached it on March 4 still allows us to tell whether a_i is compatible with the already scheduled activities, but by only having one number which represents the earliest availability of the room.

==========

Compatible means an activity does not overlap with any other already scheduled activity.

Given: a list of activities: a_i each with their start time, s_i
and finish time, f_i
a start time for the room S_r and
a closing time for the room F_r

Objective: Minimize the unused time for the room by scheduling compatible activities.

Notice that maximizing the used time is an equivalent problem and that's the approach I took in class (and below).

So that the approach below will work, we agreed to sort the activity list by finish times (low to high), so that a_i+1 finishes at a time >= the time a_i finishes. And this is true for all i 1 to n.

when were are at i, meaning activity a_i and
have S_ra = the earliest room availability

choice	subproblem(s)	immed. reward
if s_i < S_ra (means we MUST NOT schedule a_i)	i+1, S_ra	0
if s_i >= S_ra either schedule a_i	i+1, f_i	f_i - s_i
OR do not schedule a_i	i+1, S_ra	0

The value function in words:

F(i, S_ra) = optimal solution to the problem of scheduling activities
i through n in a room with earliest availability of Sra
Note: optimal solution = maximizing the time room is used

The value function in Math (based directly on the
choice/subproblem/immed.reward table:

```
            {  F(i+1, S_ra)                # if s_i < S_ra # means a_i is NOT
            {                                              # compatible with the
F(i, S_ra) = {                                             # room availability
            {
            {  MAX(F(i+1, f_i) + (f_i-s_i), #  otherwise (that is if s_i >= S_ra)
            {      F(i+1, S_ra))
```

Base cases:

F(n+1, j) = 0   for all j in range S_r to F_r inclusive

==========

Your job is to finish the dynamic programming solution by writing out the pseudocode for the algorithm. Also analyze the algorithm for runtime.