CSCI 3104 Spring 2018 Problem Set 6
First Last (mm/dd)

Problem 1

(15 pts) An exhibition is going on in Hogsmeade today from $t = 0$ (9 am) to $t = 720$ (6 pm), and world-renowned wizards will attend! There are n wizards $W_1, \cdots, W_n$ and each wizard W_j will attend during a time interval $I_j : [s_j, e_j]$ where in $0 \leq s_j < e_j \leq 720$. Note: the ends of the interval are ***inclusive****. The stores in Hogsmeade want to broadcast magical ads in the sky during the exhibition, multiple times during the day. In particular, each wizard must see the ad but the store also wants to minimize the number of times the ad must be shown.*

For example:

Wizard	[s, e]
Minerva McGonagall	[3, 51]
Harry Potter	[6, 60]
Ron Weasley	[6, 99]
Hermione Granger	[105, 155]
Gilderoy Lockhart	[121, 178]
Viktor Krum	[86, 186]

Then, if the ad is shown at times $t_1 = 51$ and $t_2 = 150$, then all 6 of the wizards will see the ad.

(a) *Greedy algorithm $\mathcal{A}$ selects a time instance when the maximum number of wizards are present simultaneously. An ad is scheduled at this time and the wizards who see this ad are then removed from further consideration. The algorithm A is then applied recursively to the remaining wizards.*

Give an example where this algorithm shows more than the minimum number of ads needed.

Wizard	[s, e]
1	[0, 50]
2	[25, 140]
3	[25, 100]
4	[75, 175]
5	[75, 175]
6	[150, 250]

The greedy algorithm would opt to show 3 ads total:

- The first choice where the maximum amount of wixards are at the exhibition at the same time (4) would be sometime in the range [75,100]. This removes wizards 2-5 from consideration, leaving wizards 1 and 6.
- The second and third choices would each be one of the intervals during which the remaining two wizards are there - specifically [0,50] and [150,250].

The minimum amount of ads needed for this example is 2 – one ad at time t=50 would show the ad to wizards 1-3, and another ad at time t=150 shows the ad to the wizards 4-6.

(b) *(10 pts extra credit) Let W_j represent the renowned wizard who leaves first and let $[s_j, e_j]$ be the time interval for W_j. Suppose we have some solution $t_1, t_2, \cdots, t_k$ for the ad times that cover all wizards. Let t_1 be the earliest ad time.*

Prove the following facts for the earliest scheduled ad (at time t_1). For each part, your proof must clearly spell out the argument. Overly long explanations or proofs by examples will receive no credit.

S1. Prove $t_1 \leq e_j$. (Three sentences. Hint: proof by contradiction.)

Suppose $t_1 > e_j$; because e_j = the time at which W_j leaves, W_j would not have seen an ad during his time at the exhibition, as there were no ads shown before t_1. Therefore $t_1, t_2, \cdots, t_k$ is not a solution.

S2. If $t_1 < s_j$, then t_1 can be deleted, and the remaining ads still form a valid solution. (Five sentences. Hint: suppose deleting t_1 leaves results in a wizard not seeing the ad; think about when that wizard must have arrived and left relative to t_1, s_j, e_j. Prove a contradiction.)

Suppose deleting t_1 results in a wizard (W_0) not seeing the ad. Because $t_1 < s_j$, W_0 must have arrived prior to W_j. W_0 must also leave after e_j, as e_j is the time of the first departure. Thus $[s_j, e_j] \subset [s_0, e_0]$, meaning if W_j sees an ad, W_0 will see the same ad. Therefore if W_0 sees only the ad at t_1, then W_j will never be shown an ad during their visit and $t_1, t_2, \cdots, t_k$ is not a solution.

S3. If $t_1 < e_j$, then t_1 can be modified to be equal to e_j , while still remaining a valid solution. (Three sentences. Hint: suppose setting $t_1 := e_j$ leaves a wizard uncovered–that is, without having seen an ad–then when should that wizard have arrived and left? Prove a contradiction.)

Suppose setting $t_1 := e_j$ means a wizard did not see an ad during their visit. That wizard must have left before $e_j = t_1$, meaning e_j is not the time at which the first wizard left.

(c) *Use the results stated in (1b) to design a greedy algorithm that is optimal.*

(i) *Write pseudocode for your algorithm.*
input w = list of tuples of the form (s_i, e_i) where:
s_i = arrival time of wizard i
e_i = departure time of wizard i.

```
1.  greedyAdTimes(w){
2.    remaining = w
3.    ads = []  // list of ad times
4.    while remaining.length > 0{
5.      // t = minimum departure time of all remaining wizards
6.      t = min( [x[1] for x in remaining] )
7.      ads.append(t)
8.      // remove all wizards that saw ad at time t
9.      remaining = [x for x in remaining if (x[0] > t or t > x[1])]
10.    }
11.    return ads
12. }
```

[working python code in appendix 1]

(ii) *Prove that your algorithm is correct (**assuming the results stated in (1b)**) and give its running time complexity.*

My algorithm simply repeats the following steps until every wizard has seen an ad:

i. line 6: find $t_1 = e_j$ (as defined in 1b) for the current list of wizards who have yet to see an ad
ii. line 7: add t_1 to the set of selected ad times
iii. remove all wizards present at time t=t_1 from the list, as they will see an ad at t_1.

Essentially, it just chooses an ad time at t=e_j for a subset of attending wizards until every wizard has seen an ad. Because e_j=the time at which the first wizard leaves is chosen for each subset of wizards, all wizards will see an ad.

e_j is the optimal choice for the first ad because we know from *S1*: $t_1 \leq e_j$ and we know from *S3*: t_1 can be set to e_j without affecting the validity of the solution. Setting t_1 to e_j maximizes the probability that other wizards (besides W_j) will also see the ad. This is the greedy choice.

The algorithms main loop runs $\leq n$ times - each time:

i. computing the minimum departure time e_j from the current subset of wizards, which takes O(n) time (although this could be optimized with a priority queue).
ii. removing the wizards present at time e_j, which also takes O(n)

This means the algorithm runs in $O(n^2)$ in the worst case in which no wizards attend simultaneously.

(iii) *Demonstrate the solution your algorithm yields when applied to the $n = 6$ example above.*

remaining wizards: [(3, 51), (6, 60), (6, 99), (105, 155), (121, 178), (86, 186)]
min departure time = ad 1 = 51
remaining wizards: [(105, 155), (121, 178), (86, 186)]
min departure time = ad 2 = 155
remaining wizards: []
ad times = [51, 155]

Problem 2

(20 pts) Professor Dumbledore needs helpers to watch the gates as much as is possible. In order to minimize disruption to their class schedules, he asks students and professors when they are available, and they each provide a set of time ranges. To simplify scheduling matters, Prof. Dumbledore will simply select a set of these ranges and assign the relevant people–that is, he never assigns someone just a part of one of their ranges.

Given a set S of n time ranges on a given day, he asks you to find a subset T of these ranges which is ***covering****, in the sense that every time that could be covered by someone according to all the ranges S,* ***is*** *covered by one of the ranges in T. Your goal is to minimize the size of T (=the* ***number*** *of ranges it contains, regardless of how long they are).*

For the following, assume that Dumbledore gives you an input consisting of a single array S where the i-th element S[i] describes the i-th range as a pair (s_i, l_i) where $s_i < l_i$.

(a) *In pseudo-code, give a greedy algorithm that computes the minimum-size covering subset T in $\mathcal{O}(n \log n)$ time. Explain your solution in plain English as well, and prove an $\mathcal{O}(n \log n)$ upper bound on its running time. (Hint: Start with the earliest range.)*

TODO

(b) *Prove that your algorithm is correct, in particular, that it correctly computes the* ***minimum-size*** *covering subset.*

TODO

Problem 3

(20 pts) We saw on the previous problem set that the cashiers (greedy) algorithm for making change doesnt handle arbitrary denominations optimally. In this problem youll develop a dynamic programming solution which does, but with a slight twist. Suppose we have at our disposal an arbitrary number of ***cursed*** *coins of each denomination $d_1, d_2, \cdots, d_k$, with $d_1 < d_2 < \cdots < d_k$, and we need to provide n cents in change. We will always have $d_1 = 1$, so that we are assured we can make change for any value of n. The curse on the coins is that in any one exchange between people, with the exception of $i = 2$, if coins of denomination d_i are used, then coins of denomination d_{i-1} cannot be used. Our goal is to make change using the minimal number of these cursed coins (in a single exchange, i.e., the curse applies).*

(a) *For $i \in \{1, \cdots, k\}$, $n \in \mathbb{N}$, and $b \in \{0, 1\}$, let $C(i, n, b)$ denote the number of cursed coins needed to make n cents in change using only the first i denominations $d_1, d_2, \cdots, d_i$, where d_{i-1} is allowed to be used if and only if $i \leq 2$ or $b = 0$. That is, b is a Boolean flag variable indicating whether we are excluding denomination d_{i-1} or not ($b = 1$ means exclude it). Write down a recurrence relation for C and prove it is correct. Be sure to include the base case.*

TODO

(b) *Based on your recurrence relation, describe the order in which a dynamic programming table for $C(i, n, b)$ should be filled in.*

TODO

(c) *Based on your description in part (b), write down pseudocode for a dynamic programming solution to this problem, and give a Θ bound on its running time (remember, this requires proving both an upper and a lower bound).*

TODO

References

1. CLRS

Appendix 1
Python 3 code for (1c):

```
def greedyAdTimes(w):
  remaining = list(w)
  ads = []
  while len(remaining) > 0:
    ej = (min(remaining, key=lambda x:x[1]))[1]
    ads.append(ej)
    remaining = list(filter(lambda x: x[0] > ej or ej > x[1], remaining))
  return ads
```