Homework 7

Due by 4:30pm on October 28, 2020

Viewings and Readings

[Links to the slides, recordings, and quiz are on the course web site.]
Review the week 7 lecture slides.
View the week 7 lecture recordings and the discussion session recording.
Complete the week 7 quiz.
Read sections 4.3-5 of the textbook.

Problems

Please write precise and concise answers. Except where explicitly indicated, your algorithm descriptions should use either clear, concise, and precise plain English or clear, concise, and precise pseudo-code that uses a style similar to the pseudo-code in your textbook. Submit your solutions to problems **1-3(b)** via D2L as a Word or PDF file or as scans/photos of legible handwritten notes. Submit your solution to problem **3(a)** via Kattis.

1. Problem 12 page 130 parts (a) and (b) only. You may assume that you, not Elmo, is making the first move. For part (a), you need to give an example. In part (b) you need to develop a dynamic programming algorithm using the usual approach (1(a), 1(b), 2(a), ...).

You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends. Having never taken an algorithms class, Elmo follows the obvious greedy strategy-when it's his turn, Elmo always takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely hates it when grown-ups let him win.

(a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do not follow the same greedy strategy as Elmo

```
Picking the highest value won't work in every scenario, for example the points could be arranged in

2, 10, 1, 1

If the first player uses the always take the highest points greedy strategy then round one they'll take the left 2, then player 2 gets the left 10 ar
```

(b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

```
Assuming the cards are an array X[1 ... n].  We can recursively loop through the array and calculate the max points we can get based on the decision

elmo(X)
  n = length(X)
  if n == 0
    return 0

  // elmo always chooses highest card value
  if X[1] > X[n]
    X = X[2 ... n]
  else
    X = X[1 ... n-1]

  // try both taking the left and right card and return the max
  left = X[1] + elmo(X[2 ... n])
  right = X[n] + elmo(X[1 ... n - 1])

  return max(left, right)
```

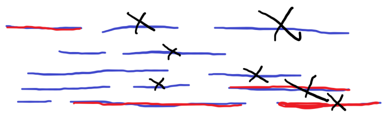
2. Problem 3 page 178. Note that the interval that we hit first, when moving from left to right, must be in the cover. How do you then choose the next interval to add to the optimal cover? Find a greedy approach that works and describe the algorithm. Prove that your algorithm is correct by proving the greedy choice and optimal substructure properties.

Let X be a set of n intervals on the real line. We say that a subset of intervals Y in X covers X if the union of all intervals in Y is equal to the union of all intervals in X. The size of a cover is just the number of intervals. Describe and analyze an efficient algorithm to compute the smallest cover of X. Assume that your input consists of two arrays L[1 .. n] and R[1 .. n], representing the left and right endpoints of the intervals in X. If you use a greedy algorithm, you must prove that it is correct.

```
A greedy algorithm that solves this would be to start at the earliest start time.  Then from there take the interval that starts after the previous i

Assuming L and R are sorted by starting point of the L part of the interval

initial call is line(1)
line(x)
  if R[x] = max(R)
    return 1

  maxR = 0
  for i = 1 to n
    // if interval starts within interval x
    if L[i] > L[x] && R[i] < R[x]
```

```
      // take the max end that overlaps with x
      if i > maxR
        maxR = i

  return line(maxR) + 1
```

```
The optimal solution requires that the earliest start and the latest end are in the solution.
If we reduce the array to only intervals that intersect with the first interval, the optimal solution has to be the intersecting interval that ends l
We can keep extending the array by only intersecting intervals and the next optimal interval will always the the last to end
```



3. Week 7 problem *woodcutting* on Kattis.

(a) Implement a greedy algorithm using your preferred language and submit your implementation via Kattis.

(b) Prove that your algorithm is correct by proving the greedy choice and optimal substructure properties.

```
The idea of the algorithm is to do work in the smallest to largest orders to complete order for customers

In any order the wait time of customer 2 is the sum of wait time for customer 1 + wait time of customer 2.  Customer 3 will be the sum of customer 1

Each time we add a customer the previous customers wait time are added to that new one.  To get the average as low as possible we need to save the lo

For example an optimal solution is:
  1, 5, 10
  Customer 1 = 1
  Customer 2 = 1 + 5
  Customer 3 = 1 + 5 + 10
  So the 10 is only needed in the last wait time when done optimally

Unoptimal solution would be:
  10, 5, 1
  Customer 1 = 10
  Customer 2 = 10 + 5
  Customer 3 = 10 + 5 + 1
  So the 10 is added to everyones wait time
```