CS 252: Algorithms
Exam 3. Due on paper by 5:00PM Monday, 23 November 2015.

This is an open-book, open notes, open-Internet exam. You may not consult with people other than Jeff Ondich. Use LaTeX as much as you can, but you may include neatly hand-drawn diagrams if appropriate.

Hand in your paper either directly to Jeff, or via his 2nd-floor CMC mailbox.

1. (10 points) For each of the following, name an algorithm whose worst-case running time can be modeled by the recurrence relation in question. In addition, in each case give an $f(n)$ (without proof) for which $T(n) \in O(f(n))$. Wherever it appears, consider c to be a positive constant.

   (a) $T(n) = T(\frac{n}{2}) + c$
   (b) $T(n) = 2T(\frac{n}{2}) + cn$
   (c) $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + cn$
   (d) $T(n) = T(n-1) + c$
   (e) $T(n) = T(n-1) + cn$

2. (10 points) Due to slightly over-optimistic sales projections, I find myself with a basement full of boxes of Basque-Aymara/Aymara-Basque translation dictionary software for Amiga computers. Unable to bring myself to simply discard these lovingly crafted floppy disks, I am thrilled to discover that tomorrow, there is going to be an Andes/Pyrenees computer history conference in Ecuador. If I can just get these boxes to Quito by then, my software will support international understanding and good will.

   I have a schedule F of n flights available today. Each flight F_k in this schedule is described by five values $F_k = (s_k, t_k, d_k, a_k, c_k)$:

   s_k = starting city
   t_k = ending city
   d_k = departure time
   a_k = arrival time
   c_k = capacity (number of boxes of my software it can carry)

   Describe an efficient algorithm for figuring out how to get as many boxes as possible from Minneapolis to Quito by tomorrow. You may assume that all the flights run on time, and that it takes no more than one hour to move the boxes from an arriving plane onto a departing plane. Make your algorithm as efficient as possible, and analyze its running time.

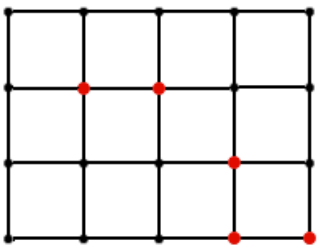
3. (10 points) The streets in the city of Latticeburgville are laid out in a rectangular grid. All the east-west streets are one-way going west to east, and all the north-south streets are one-way going north to south. (Given the power of the Latticeburgville City Planners and their enforcement team, smart citizens don't ask how one is to get home after traveling to the nearest grocery store to the southeast. They just dutifully leave town, follow the state highways back to Northwesterton, and reenter Latticeburgville therefrom.)

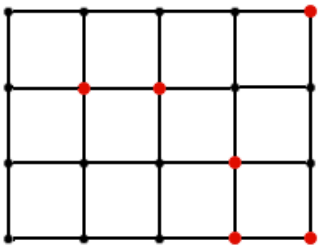
   The Latticeburgville garbage trucks are, of course, constrained by the one-way streets. But when garbage collection day arrives, they have to empty all the garbage bins nonetheless. To make sure to

minimize the time during which town remains bucolically free of garbage truck noise, the City Planners insist that each garbage truck take only one pass from the northwest corner to the southeast corner, emptying the bins they encounter along the way. Note that in the interests of rectilinear tidiness, all garbage bins in Latticeburgville are found at intersections.

Your job, as a new City Planner "recruit," is to devise an algorithm that will take any layout of garbage bins and determine the minimum number of garbage trucks required. You will be given the number W of north-south streets, the number H of east-west streets, and a list of (x, y) coordinates $b_i = (x_i, y_i)$ for the garbage bins, where intersection $(0, 0)$ is the northwest corner of the town and $(H - 1, W - 1)$ is the southeast corner.

For example, if your algorithm were given the map on the left below (with bins indicated in red), it would report that a single garbage truck is sufficient. But for the map on the right, the minimum number of trucks is two.





Pleasing the City Planners is always a high priority for Latticeburgville citizens who wish to wake up refreshed (or at all) tomorrow morning, so you would be well advised to make your algorithm as efficient as possible. Note that the City Planners never hesitate to annex land when necessary, so W and H could grow arbitrarily large.

Propose an algorithm and analyze its running time. Good luck, and don't forget that jaywalking is a capital crime in Latticeburgville.

4. (10 points) A few years ago, my dictionary software company was contacted by a company whose business was domain-name speculation. That is, they bought up domain names they considered likely to be valuable, and tried to sell them for higher prices. One of their techniques was to keep an eye on domain names whose owners were allowing (intentionally or neglectfully) their domain registrations to expire. This company contacted us because they wanted to know which of the 43 million soon-to-be-available domain names actually meant something in any of six European languages.

   My job was to take our dictionary data plus the list of 43 million domains, and try to segment each domain into a sequence of words. For example, if the domain was "miseenscene.com", my software should report (French, "mise en scène") and (English, "mi seen scene"). Obviously, the French interpretation is preferable, since "mise en scène" actually means something, while "mi" is an odd-ball little English word of greatest interest to Scrabble players. But assuming we're just looking for sequences of words that actually appear in a dictionary, both of these interpretations are acceptable.

   An additional constraint I applied to this problem was to seek the "best" segmentation assuming there were several possible segmentations. For example, consider the string "theyouthevent". Though `['the', 'youth', 'event']`, `['the', 'you', 'the', 'vent']`, and `['they', 'out', 'he', 'vent']` are all legal segmentations into English words, the first one is clearly the best on syntactic and semantic grounds. As you work on this problem, you may assume that you have access to a linear-time function `goodness(list of strings)` that returns an integer measuring the quality of the list of strings. We can assume, for example, that

`goodness(['the', 'youth', 'event'])`

is a larger number than

`goodness(['they', 'out', 'he', 'vent'])`.

(a) Write a pseudo-code recursive algorithm to solve this segmentation problem, assuming you have access to a list of English words. Your algorithm should involve a function that takes a string as input, and reports the best segmentation as output (assuming, as discussed above, that you already have a well-designed goodness function). For example, your function would take `'thebigmoose'` and return `['the', 'big', 'moose']`. If the segmentation fails (i.e. the input string has no segmentations into words in your word list), the function should return an empty list.

(b) Discuss the worst-case running time of your recursive algorithm, and provide an upper bound.

(c) Propose a dynamic programming modification to your algorithm. What sorts of data structures will your modification require? Will your modifications improve the performance of the algorithm? If so, explain why, and discuss the extent of the improvement (e.g. if it were to turn an exponential algorithm into a polynomial algorithm, or an n^5 algorithm into an n^3 algorithm, you would want to say so). If not, why not?

5. (2 points, awarded in advance) Have a great break!