Algorithms Lab 6

(Divide-and-conquer, dynamic programming)

Laura Toma, csci2200, Bowdoin College

This week's topics

- Divide-and-conquer:
  - Karatsuba integer multiplication
  - Strassen's algorithm for matrix multiplication
  - Maximum partial subarray
- Dynamic programming (board game and rod cutting)

In-class (COLLABORATION LEVEL 0[1])

Playing the board game.

Homework problems (COLLABORATION LEVEL 1[2])

1. You are given a sorted array of numbers where every value except one appears exactly twice; the remaining value appears only once. Design an efficient algorithm for finding which value appears only once.

   Example: Here are some example inputs to the problem:

$$1, 1, 2, 2, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8$$

$$10, 10, 17, 17, 18, 18, 19, 19, 21, 21, 23$$

$$1, 3, 3, 5, 5, 7, 7, 8, 8, 9, 9, 10, 10$$

[1]Collaboration level 0: everything allowed!

[2]Collaboration level 1: verbal collaboration without solution sharing. You are allowed and encouraged to discuss ideas with other class members, but the communication should be verbal and additionally it can include diagrams on board. Noone is allowed to take notes during the discussion (being able to recreate the solution later from memory is proof that you actually understood it). Communication cannot include sharing pseudocode for the problem. Check complete guidelines at: https://turing.bowdoin.edu/dept/collab.php)

2. The skyline problem/the upper envelope problem: In this problem we design a divide-and-conquer algorithm for computing the skyline of a set of n buildings.

   A *building* B_i is represented as a triplet $(\mathbf{L_i}, H_i, \mathbf{R_i})$ where $\mathbf{L_i}$ and $\mathbf{R_i}$ denote the left and right x coordinates of the building, and H_i denotes the height of the building (note that the x coordinates are drawn boldfaced.)

   A *skyline* of a set of n buildings is a list of x coordinates and the heights connecting them arranged in order from left to right (note that the list is of length at most $4n$).

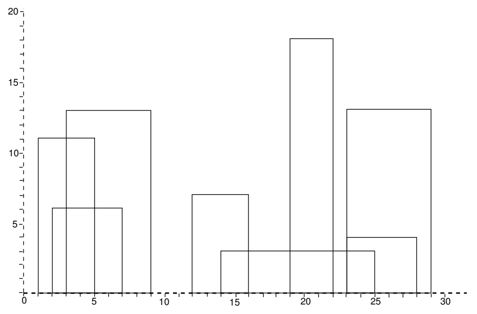
   Example: The skyline of the buildings

$$\{(\mathbf{3}, 13, \mathbf{9}), (\mathbf{1}, 11, \mathbf{5}), (\mathbf{12}, 7, \mathbf{16}), (\mathbf{14}, 3, \mathbf{25}), (\mathbf{19}, 18, \mathbf{22}), (\mathbf{2}, 6, \mathbf{7}), (\mathbf{23}, 13, \mathbf{29}), (\mathbf{23}, 4, \mathbf{28})\}$$

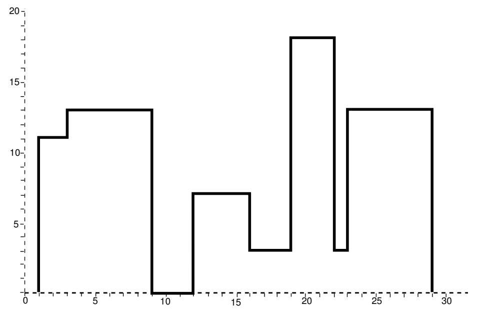
   is

$$\{(\mathbf{1}, 11), (\mathbf{3}, 13), (\mathbf{9}, 0), (\mathbf{12}, 7), (\mathbf{16}, 3), (\mathbf{19}, 18), (\mathbf{22}, 3), (\mathbf{23}, 13), (\mathbf{29}, 0)\}$$

   (note that the x coordinates in a skyline are sorted).



buildings



skyline

   (a) Let the size of a skyline be the number of elements (tuples) in its list.
   Describe an algorithm for combining a skyline A of size n_1 and a skyline B of size n_2 into one skyline S of size $O(n_1 + n_2)$. Your algorithm should run in time $O(n_1 + n_2)$.

   (b) Describe an $O(n \log n)$ algorithm for finding the skyline of n buildings.

3. Rod cutting:

   (a) Describe a *non-recursive, dynamic programming* algorithm for finding the maximum revenue obtainable from a rod of length n, given a price array $p[1..n]$ (where $p[i]$ represents the revenue of selling a rod of length i). Analyze its running time.

   (b) Describe how to augment your solution for the rod cutting problem so that it computes the actual cuts corresponding to the optimal revenue, instead of just the revenue.

4. **Greedy rod cutting:** Max suggested the following strategy as an optimization to the dynamic programming solution: when determining the next cut, instead of trying all possibilities, choose to cut a piece of length i such that the revenue per length p_i/i is maximized. This is called a *greedy* strategy, because it checks all choices "superficially" (w/o recursion) and chooses the option that looks "locally" best, without exploring all choices "deeply". Because the choices are not explored deeply, this implies that: (1) greedy algorithms are usually faster than their DP counterparts; (2) they are not always correct.

   Note that if this particuar greedy strategy for rod cutting were correct, then we could solve the problem in $O(n)$ time plus an initial sort (instead of $O(n^2)$ with dynamic programming).

   Note that a greedy strategy always "works" in the sense that it computes something —- the question to ask is: "Is it correct?" A greedy strategy (for the rod cutting problem) is *correct* if it computes the optimal revenue obtainable for any n and for any price array $p[1..n]$.

   To prove a greedy strategy correct we need to show that for any input, it computes the optimal output (more on this in class).

   To prove a greedy strategy wrong, it's sufficient to find one example (of input) where the greedy strategy will lead to a sub-optimal (and hence wrong) answer. This is called a *counterexample.*

   Prove that this strategy is **not** correct by showing a counterexample.