College of Computer & Information Science
Northeastern University
Spring 2019

CS5800
Algorithms
1 February 2019

Problem Set 2 (Due Monday, February 11, 11:59PM)

Instructions:

- The assignment is due at the time and date specified. Late assignments will be accepted, up until Tuesday, February 12, 11:59 PM. *Note, however, that you can use at most 3 late days for your problem set and programming assignment submissions throughout the course of the term.*
- We encourage you to attempt and work out all of the problems on your own. You are permitted to study with friends and discuss the problems; however, *you must write up you own solutions, in your own words.*
- Please refrain from searching online or asking your peers or other students for solutions. The best way to learn the material is to attempt the problem yourself, and if you are stuck, identify where and why you are stuck and seek help to overcome the associated hurdles.
- If you do collaborate with any of the other students on any problem, *please list all your collaborators in your submission for each problem.*
- We require that all homework submissions be neat, organized, and *typeset*. You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions. If you need to draw any diagrams, however, you may draw them with your hand.

1. **(4 + 4 = 8 points) Induction and Recursion in Binary Codes**

   Digital transmission protocols transmit signals using binary codes. In order to minimize the effect of errors, it is often useful to select a code such that "similar" signals use "similar" codewords.

   One such code is a list of 2^n n-bit strings in which each string (except the first) differs from the previous one in exactly one bit. Let us call such a list a *twiddling list* since we go from one string to the next by just flipping one bit.

   Consider the following recursive algorithm for listing the n-bit strings of a twiddling list. If $n = 1$, the list is 0,1. If $n > 1$, first take a twiddling list of $(n-1)$-bit strings, and place a 0 in front of each string. Then, take a second copy of the twiddling list of $(n-1)$-bit strings, place a 1 in front of each string, reverse the order of the strings and place it after the first list. So, for example, for $n = 2$, the list is 00,01,11,10, and for $n = 3$, we get 000,001,011,010,110,111,101,100.

   (a) Prove by induction on n that every n-bit string appears exactly once in the list generated by the algorithm.

   (b) Express the worst-case time complexity of the algorithm above as a recurrence relation, and solve the recurrence to obtain a tight-bound on its worst-case time complexity.

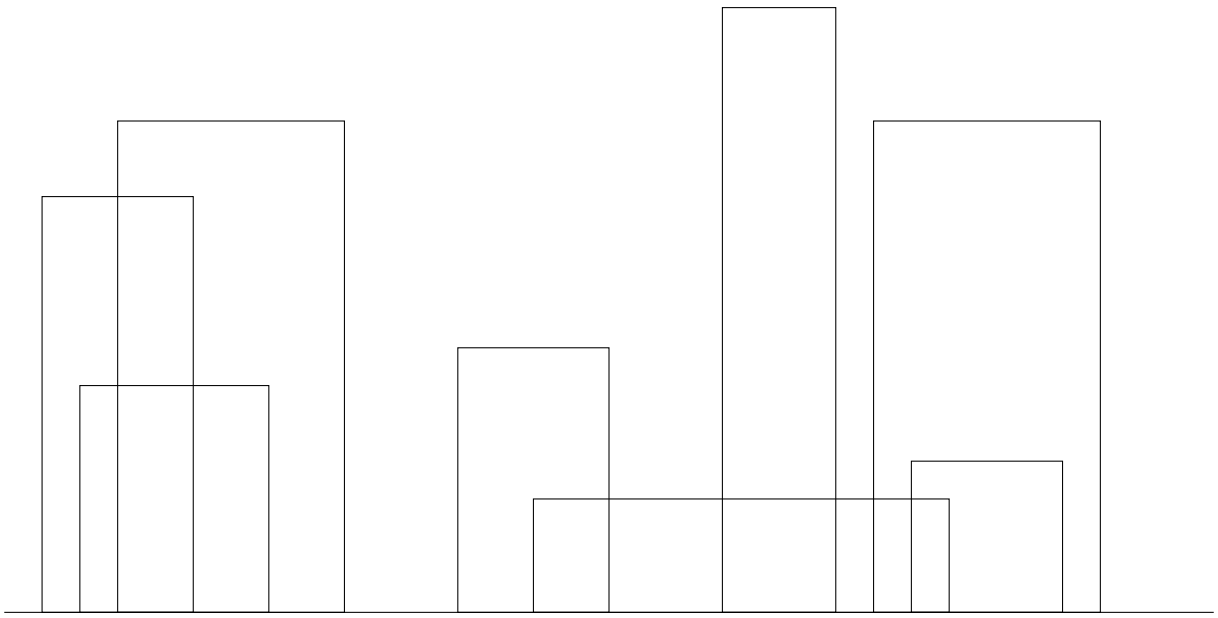

Figure 1: Input Buildings

2. **(3 + 3 + 3 + 3 = 12 points) Solving Recurrence Relations**

   Give asymptotically tight bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for suffuciently small n. You may ignore floors and ceilings in your calculations.

   (a) $T(n) = 4T(n/2) + n^{5/2}$

   (b) $T(n) = 9T(n/3) + n^2$

   (c) $T(n) = 2T(n/4) + n$

   (d) $T(n) = 2T(n/3) + T(n/4) + n$.

3. **(8 + 6 = 14 points) Drawing the Skyline**

   You are given the exact locations and shapes of several rectangular buildings in the city, in two dimensions. We assume that the bottoms of these buildings lie on a straight line. Building B_i is represented by the triple (L_i, H_i, R_i), where L_i and R_i denote the left and right coordinates of the building, and H_i denotes its height. For example, the figure above corresponds to the following input:
   (1,11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14,3,25), (19,16,22), (23, 13, 29), and (24,4,28)

   A *skyline* is a list of x coordinates and the heights connecting them arranged in order from left to right. The skyline for the example above is represented as:
   (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19,16, 22, 3, 23, 13, 29, 0).

   The figure showing the skyline is on the next page.

   (a) Design an $O(n \log_2 n)$ algorithm to determine the two-dimensional skyline of a set of n input buildings, eliminating hidden lines to produce the skyline.

   (b) Prove the correctness and time complexity of your algorithm.

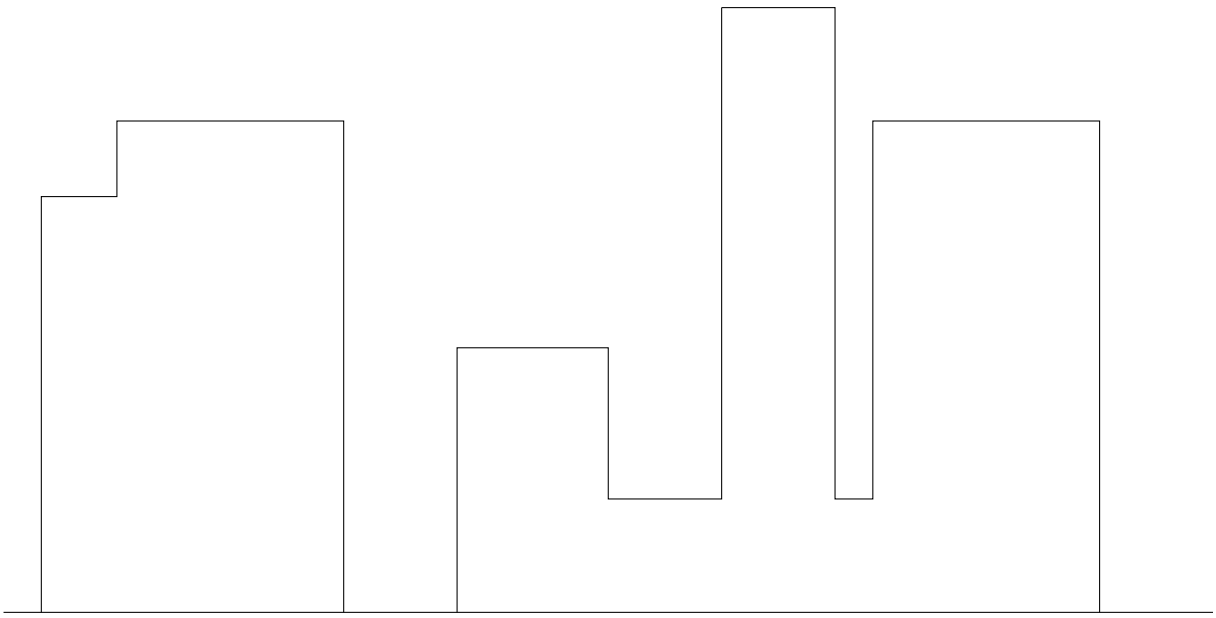

Figure 2: The Skyline

4. **(6 + 4 + 6 = 16 points) Finding the Majority Element**

   An array $A[1..n]$ is said to have a *majority element* if more than half of its entries are the same. We would like to determine whether a given array A has a majority element, and if so, find the element. We assume that the elements of the array are not necessarily from some ordered domain like the integers, so there can be no comparisons of the form "is $A[i] > A[j]$?"; only questions of the form "is $A[i] = A[j]$?" can be answered.

   (a) Design an algorithm to find the majority element of a set of n elements, if it exists, in $\Theta(n \log n)$ time. Prove the correctness of your algorithm.

   (b) Prove the worst-case time complexity of your algorithm using induction or other means.

   (c) Can you find the majority element in linear time? If so, describe your algorithm and establish its time complexity.