CS 341, W 2018  M. Li, B. Ma, S. Salihoglu

ASSIGNMENT 1

DUE: Friday, Jan. 26, before midnight. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.
Please read  http://www.student.cs.uwaterloo.ca/~cs341  for general instructions and policies.
For all algorithm design questions, you should give the algorithm, argue the correctness, and analyze time complexity.

1. [8 marks] Assume you have functions \( f \) and \( g \), such that \( f(n) \geq 1 \), \( g(n) \geq 1 \), and \( f(n) = O(g(n)) \). For each of the following statements, decide whether it is true or false. Briefly justify your choice with a proof or counterexample.
   (a) \( \log_2 f(n) = \Theta(\log_{10}(f(n))) \)
   (b) \( \log f(n) = O(\log(g(n))) \)
   (c) \( 2f(n) = O(2g(n)) \)
   (d) \( (f(n))^2 = O((g(n))^2) \)

2. [6 marks] Solve the following recurrence relation to obtain a closed-form big O upper bound for \( T(n) \). In each question, assume \( T(c) \) is bounded by a constant for any small constant \( c \).
   You need to prove the big O bound. Your bound needs to be tight in order to receive full credit. However, you do not need to prove the tightness of the bound.
   (a) \( T(n) \leq 4T(\frac{n}{2}) + n^2 \)
   (b) \( T(n) \leq T(\frac{n}{3}) + T(\frac{2n}{3}) + n \)
   (c) \( T(n) \leq \sqrt{n} \cdot T(\sqrt{n}) + n \)
   (Hint: Somewhere in between \( O(n) \) and \( O(n \log n) \).)

3. [6 marks] Analyze the time complexity of the following pseudocode, and obtain an big O upper bound with respect to \( n \). You assume that each evaluation of condition_1 and condition_2 takes constant time, and does not change the values of \( i \), \( j \), and \( n \). You need to argue the big O bound. Your bound needs to be tight in order to receive full credit. However, you do not need to prove the tightness of the bound.

\[
i = 0; j = n;
while (2i < j)\{
  if (condition_1)\{
    i++;
  } else if (condition_2)\{
    j--;
  } else \{
    i++;
    j++;
  }\}
\]

1
4. [10 marks] **Four-Sum** Given four integer arrays $A_k$ of length $n$ ($k = 1, 2, 3, 4$), and an integer $m$, design an $O(n^2 \log n)$-time algorithm to find whether there are $i_1, i_2, i_3, i_4$ from $[1, n]$, such that $\sum_{k=1}^4 A_k[i_k] = m$. Note that we require worst-case time complexity here. Thus, for the purpose of this question, the use of hash tables is not recommended.

5. [10 marks] **Power** Given $n > 0, k > 0$, and a prime number $p$, compute

$$n^k \mod p$$

in $O(\log k)$ time. Here $x \mod y$ means the remainder for $x$ being divided by $y$. For example, $2^3 \mod 7 = 1$. You assume that all of $n$, $k$, and $p$ can fit in a single word in your computer. Moreover, if $x$ and $y$ are numbers with sizes bounded by $O(1)$ words, then the computer can carry out each arithmetic operation (+, −, ×, /, mod) on $x$ and $y$ in $O(1)$ time.

6. [20 marks] **Finding Maximum Space**

(a) (10 marks) Imagine an advertising company would like to post a huge poster in Toronto harbour front. There are $n$ consecutive buildings, with height $h_1, \ldots, h_n$ and unit width as shown in Figure 1.

![Figure 1: Three possible solutions are highlighted in the figure. The largest one has area $4 \times 4 = 16$.](image)

Design a divide-and-conquer algorithm to find the maximum rectangular space to post their poster. Stated mathematically, find $1 \leq i \leq j \leq n$ to maximize $(j - i + 1) \times \min_{i \leq k \leq j} h_k$. You will get full marks if the time complexity is $O(n \log n)$ and the proofs are correct.

Remark: The most efficient known algorithm has time complexity $O(n)$ and is not divide-and-conquer based. For the marking, we require a divide-and-conquer solution. However, if you can get the divide-and-conquer solution relatively easily, you can try to develop that linear time algorithm for fun.

(b) (10 marks) Imagine the government would like to build a huge park in the city. The city is an $n \times n$ grid, with some units occupied. The input is given as a two-dimensional boolean array $A$, where $A[i,j]$ indicates whether the unit at grid $(i,j)$ is still available. See Figure 2 for an example. Use (a) or otherwise, design an algorithm to find the maximum rectangular unoccupied space to build the park.
Figure 2: Three possible solutions are highlighted in the figure. The largest one has area $7 \times 3 = 21$.

You will get full marks if the time complexity is $O(n^2)$ and the proofs are correct. You can assume that there is an $O(n)$ time algorithm for part (a).