Problem Set 1

All parts are due on February 28, 2017 at 11:59PM. You should submit the theory portion (part **A**) to `gradescope.com` and the coding portion (part **B**) to `alg.csail.mit.edu`. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Last, but not least, take a look at the collaboration policy outlined in the handout for this course.

Part A

Problem 1-1. [8 points] **Asymptotic behavior of functions**

For each group of functions, arrange the functions in the group in increasing order of growth. That is, arrange them as a sequence $f_1, f_2, \ldots$ such that $f_1 = O(f_2), f_2 = O(f_3), f_3 = O(f_4), \ldots$. For each group, briefly explain your ordering. Note that there might be more than one correct ordering.

(a) [4 points] **Group 1:**

$$\begin{aligned} f_1(n) &= 6n \\ f_2(n) &= \log\log n \\ f_3(n) &= n\log n \\ f_4(n) &= \log n \\ f_5(n) &= n\log\sqrt{n} \end{aligned}$$

(b) [4 points] **Group 2:**

$$\begin{aligned} f_1(n) &= n^{6.006}\log n^2 \\ f_2(n) &= n^2\log(n^{6.006}) \\ f_3(n) &= n^3 \\ f_4(n) &= n^2\log n \\ f_5(n) &= n^2\log n^n \end{aligned}$$

Problem 1-2. [12 points] **Recurrences**

For each of the following recurrences, provide a solution with bound specified with asymptotic Θ notation. You may use any relevant method (i.e. master theorem, recursion tree method, substitution method, etc.) to solve the recurrences, but make sure to mention your method of choice, and briefly justify your answer.

(a) [2 points] $T(n) = 2T(n/2) + \Theta(1)$

(b) [2 points] $T(n) = 2T(n/2) + \Theta(n)$

(c) [2 points] $T(n) = 3T(n/2) + \Theta(n)$

(d) [2 points] $T(n) = T(n/2) + \Theta(\log n)$

(e) [2 points] $T(n) = 2T(n/2) + \Theta(n^2)$

(f) [2 points] $T(n) = 7T(n/2) + \Theta(n^2)$

Problem 1-3. [20 points] **Crossing the line**

You are given a strictly increasing function $f : \mathbb{N} \to \mathbb{Z}$, i.e. $f(n) < f(n+1)$ for all natural numbers $n \in \mathbb{N}$. Evaluating f at any natural number $n \in \mathbb{N}$ takes constant time. Give an $O(\log k)$-time algorithm that finds the smallest natural number k such that $f(k) \geq 0$.

Note that the desired runtime complexity bound is in terms of the answer k that we do not know in advance.

(*Hint:* Consider obtaining an $O(\log^2 k)$-time algorithm first, and then try to optimize it.)

Problem 1-4. [30 points] **Sorting it out**

As everything in life, it is time for the `6.006` mailing service to get revamped. This service is responsible for managing all incoming and outgoing MIT packages, where each package x is described by the following two fields:

$priority$: an integer describing the priority of a given package (larger integers correspond to higher priority)

zip: an integer between 0 and $Z - 1$, inclusive, describing the US zip code of the destination of the package

You are asked to design a data structure D that supports the following four operations:

INIT(Z): Return an empty D supporting US zip codes between 0 and $Z - 1$ in $O(Z)$ time.

INSERT(D, x): Insert in D a package x in $O(\log size(D))$ time.

MOST-URGENT(D): Remove from D and return the highest-priority package x in $O(\log size(D))$ time.

REGIONAL-MOST-URGENT(D, r): Exactly as MOST-URGENT but with the constraint $x\,.\,zip = r$; this operation should still take $O(\log size(D))$ time.

Note that $size(D)$ is the number of package records stored in D.

Part B

Problem 1-5. [30 points] **Peak-finding adversary**

Recall the one-dimensional peak-finding problem: given an array of n positive numbers, find an element of the array that is not smaller than its neighbor(s).

Shown in lecture was a divide-and-conquer solution to this problem, guaranteed to make only $O(\log n)$ queries to the array. (It might be a good idea to review this material before proceeding.)

As always, we ask ourselves: is it possible to do better? Here, you will prove that it is not; you will show that $\Omega(\log n)$ queries are required in the worst case.

We will think of the peak-finding problem as a game between a player and an adversary. The game proceeds in rounds until a peak is found. During each round, the player chooses a cell in the array to query; the adversary replies with the value of the chosen cell. Of course, if in different rounds the player queries the same cell, the adversary should give the same answer. We will call such an adversary *consistent*.

In this assignment you will design an adversary to force any player to make $\Omega(\log n)$ queries. The skeleton code for the adversary you will implement is located in `peak_finding.py`. An example of how this code will be called is located in `tests.py`.

(a) [5 points]

Implement an adversary that is consistent.

(b) [25 points]

Implement an adversary that forces the player to make at least $\log_2 n - 1$ queries. More specifically, design and implement a strategy for answering queries such that the player will never find a peak in less than $\log_2 n - 1$ queries. Your adversary should be consistent.

Although we will not be explicitly testing the asymptotic complexity of your code, it should be efficient enough to run in a reasonable amount of time even on large inputs.

Submit your finished `peak_finding.py` (satisfying both parts **(a)** and **(b)**) to `alg.csail.mit.edu`. Your code will be run as `python3`. The autograder will be online for testing this weekend. (You will be permitted to submit to the autograder as many times as you want until the deadline; only the last submission to finish running before the deadline will count.)