EE360C: Algorithms
The University of Texas at Austin — Exam #2
Drs. Christine Julien and Andrea Thomaz — March 30, 2017

Name: **EID:**

Exam #2

Instructions. WRITE LEGIBLY.
No calculators, laptops, or other devices are allowed. This exam is **closed book**, but you are allowed to use a **one-page, handwritten** reference sheet. Write your answers on the test pages, in the space provided. If you need scratch paper, use the back of the test pages, but note that **we will not look at the backs of pages** during our grading process. Write down your process for solving questions and intermediate answers that **may** earn you partial credit.

If you are unsure of the meaning of a specific test question, write down your assumptions and proceed to answer the question on that basis. **Questions about the meaning of a question will not be answered during the exam.**

When asked to describe an algorithm, you may describe it in English or in pseudocode. If you choose the latter, make sure the pseudocode is understandable.

If you write information in response to a question, and that information is incorrect, you will not earn full credit. In the same vein, if a question asks for a finite number of things, and you provide "extra" things, we will ignore anything extra, and grade only the first answers. If many you have extra markings on a page, be sure to clearly indicate what is to be considered your answer.
You have **90 minutes** to complete the exam. The maximum possible score is 100.

The Master Theorem
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence:

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b to be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

The Master Theorem Revisited
When a reccurence is written as:

$$T(n) = aT(n/b) + \Theta(n^l \, (\lg n)^k)$$

where $a \geq 1$, $b > 1$, $l \geq 0$, and $k \geq 0$.

Case 1: $l < \log_b a$. Then $T(n) = \Theta(n^{\log_b a})$.
Case 2: $l = \log_b a$. Then $T(n) = \Theta(f(n) \lg n)$. Which is equivalent to $T(n) = \Theta(n^l (\lg n)^{k+1})$.
Case 3: $l > \log_b a$. Then $T(n) = \Theta(f(n)) = \Theta(n^l \, (\lg n)^k)$.

Name: **EID:**

Problem 1: Greedy Trash Can Placement [20 points]

The City of Austin is planning ahead for next year's SXSW festival. They noticed that the placement of trash and recycle bins along 6th street was not optimal this year. Next year they want to strategically place bins to keep trash off the street by making sure trash bins are placed along the sidewalk in such a way that they are on the same side of the street and within 20ft of the entrance to any establishment that serves take-away food. But they haven't figured out the algorithm.

(a) [**10 points**] Give an efficient greedy algorithm that finds a set of locations along one side of 6th street for placing trash bins that achieves the city's goal while minimizing the number of bins they have to place. State and briefly justify the runtime of your algorithm? (Hint: this is a one-dimensional problem, consider the sidewalk as a line and determine the locations along that line to place trash bins.)

Name: **EID:**

(b) [**10 points**] Prove that the greedy choice that your algorithm makes is the optimal choice.

Name: **EID:**

Problem 2: Efficient Optimal Band Booking [20 points]

You are the agent for a band for which you would like to book a gig at a particular venue. You've found publicly available information about the popular times of the venue. Specifically, for every hour of a particular day of the festival, you know the estimated number of people expected at the venue. Assume that the popularity of the venue is *unimodal* on the given day, that is, the number of expected people increases up to a certain time, after which it only decreases. You would obviously like to schedule your band for the peak popular time. Assume that the popularity values for a given day are provided to you in an array, ordered by time.

(a) [**10 points**] Give an efficient algorithm to find the peak popular time for the given venue on the given day.

Name: **EID:**

(b) [**10 points**] If your algorithm is a greedy one, state its running time and prove that the greedy choice is optimal. If your algorithm is a divide and conquer one, state the relevent recurrence, briefly argue why it represents your algorithm's running time, then solve it.

Name: **EID:**

Problem 3: Bouncer Schedules [20 points points]
Each SXSW venue hires bouncers to check IDs at the door. A particular venue that wants to ensure that it has *two bouncers* scheduled to work at all times, from open (t_o) to close (t_c).

- All of the bouncers in Austin make their schedules available in some central repository.
- Bouncers make themselves available in shifts, and each shift has a start time and end time.
- A venue can hire a bouncer for a complete shift or for any fraction of an available shift.
- Assume that there are always at least two bouncers available at any given time.

From a venue's perspective, an optimal bouncer schedule minimizes the number of distinct shifts that have to be scheduled (e.g., hiring bouncer A to cover door 1 from open to close is preferred to scheduling two people to cover door 1).

Given the open and close time of our specific venue, describe an efficient algorithm for generating that venue's optimal bouncer schedule, given the set of available bouncer shifts. Recall that our venue needs to have two bouncers scheduled whenever the venue is open. Your algorithm should return the schedule of shifts for the venue, where each shift in the schedule includes the bouncer's name, the start time of the shift, and the end time of the shift.

State and briefly justify the runtime of your algorithm.

Name: **EID:**

Problem 4: Festival Traffic Planning [20 points]
During SXSW, road closures and delays are caused by festival traffic and events. This makes it very difficult to figure out the best route to drive from one point to another.

To aid drivers, the city has implemented a system that will give the (correct) travel time on a given road segment, given a specific starting time within the festival. That is, given a road segment (e.g., the 600 block of N. Congress Avenue) and a time (e.g., 7pm on the Tuesday of the festival), the system returns a travel time for that road segment (e.g., 5 minutes). Travel times in the city obey two rules: (1) travel times are always positive and (2) for $t_1 < t_2$, $t_1 + f_e(t_1) \leq t_2 + f_e(t_2)$ (that is, you cannot arrive earlier by waiting to start later).

Given a graph that represents the city's street network, this system can be represented as a function $f_e(t)$ that, for a given edge e, returns the travel time for road segment e, assuming the driver enters the segment at time t.

Give an efficient algorithm to determine the earliest possible arrival time at a given destination d, given a starting location s and a departure time t_d. State and briefly justify the running time of your algorithm.

Name: **EID:**

Problem 5: Optimal Festival Scheduling [20 points]
To maximize festival profits, the organizers of SXSW have decided that bands with higher popularity should play later than bands with lower popularity. They have tasked you with identifying the swaps necessary to transform a schedule based on last minute popularity information.

As an example, consider the following schedule:

7pm Jain
8pm The Big Moon
9pm Vagabon
10pm Las Kellies
11pm Marika Hackman

Suppose that we discover that the following is true about their popularity: Jain is the least popular, followed by Las Kellies, then Vagabon, then The Big Moon, then Marika Hackmann. That is, the schedule, sorted by popularity, should be:

7pm Jain
8pm Las Kellies
9pm Vagabon
10pm The Big Moon
11pm Marika Hackman

As a first step in your task, you would like to identify how many "swaps" it would take to get the original schedule in order. For logistical reasons, swaps can only be done incrementally (that is, a single swap must exchange two bands that are scheduled to play back to back). In the example, three such swaps are required to get the schedule in order (we would initially move The Big Moon to 9pm and Vagabon to 8pm, then swap The Big Moon to 10pm and bring Las Kellies to 9pm, and finally swap Vagabon and Las Kellies to get the final schedule). Give a divide and conquer algorithm to solve the generic version of the problem. Your algorithm should take as input the two schedules (the current schedule and the target schedule) and return the number of swaps that need to be made in order to transform the original schedule into the target one. Write the recurrence for your algorithm and solve it using a recursion tree.

(Write your solution on the next page.)

[Solution to Question 5 goes here.]