EE360C: Algorithms
The University of Texas at Austin — Exam #3
Drs. Christine Julien and Andrea Thomaz — May 15, 2017

Name: **EID:**

Exam #3

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 in the exam booklet, in the space provided. 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 question, write down your assumptions. **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 incorrect information in response to a question, you will not earn full credit. If you have extra markings on a page, be sure to clearly indicate what is to be considered your answer.

You have **180 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 n/b can 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 following problems can be assumed to be known NP-Complete problems. **No other problems are assumed to be in the class of NP-Complete problems for this exam.**

The 3-COLORING Problem. *Given an undirected graph $G = (V, E)$, does there exist a function $c : V \to \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$ (i.e., is there a labeling of the graph using three labels such that any two adjacent vertices do not have the same label)?*

The Independent Set Problem. *Given a graph G and a number k, does G contain an independent set of at least k? In a graph $G = (V, E)$, a set of nodes $S \subseteq V$ is* independent *if no two nodes in S are joined by an edge.*

The Subset Sum Problem. *Given natural numbers $w_1, w_2, \ldots, w_n$ and a target number W, is there a subset of $\{w_1, w_2, \ldots, w_n\}$ that adds up to precisely W?*

Problem 1: Algorithm Styles [20 points]
For each of the following algorithms, determine if the algorithm is using a *greedy*, *divide and conquer*, *memoization* or *dynamic programming* approach. For each algorithm also provide the algorithm's running time in Big-O notation using the most appropriate method (e.g., aggregate analysis, loop analysis, a recurrence relation, etc.).

(a) [5 points] *Factorial.* The problem: Compute the factorial of n, a non-negative integer. Let the array fact$[0\,..\,n]$ be a globally accessible array of length n and let fact$[i] = 0$ initially for all i.

```
FACTORIAL(n)
1  if n = 0 then return 1 [by the convention that 0! = 1]
2  else if fact[n] > 0 then return 0
3  else return FACTORIAL(n − 1)
```

(b) [5 points] *Movie Marathon.* The problem: given the start and finish times of movies playing at a theater in town, find a schedule that maximizes the number of movies you see. The algorithm: sort movies by finish time, set of selected movies S starts empty. For $m = 1...n$ (in order by finish time), if movie m is compatible with the set S, put m in to S then move on to the next movie ordered by finish time.

(c) [5 points] *Subset Sum.* The problem: given an integer bound W and a collection of n items, each with a positive, integer weight w_i, find a subset S of items that maximizes $\sum_{i \in S} w_i$ while keeping $\sum_{i \in S} w_i \leq W$.

```
SUBSETSUM(n, W)
1  let s[0..n][0..W] be a new array
2  s[0, j] = 0 for all 0 ≤ j ≤ W
3  s[i, 0] = 0 for all 0 ≤ i ≤ n
4  for i = 1 to n do
5     for j = 1 to W do
6        if w_i > j then s[i, j] = s[i − 1, j]
7        else s[i, j] = max(s[i − 1, j], s[i − 1, j − w_i] + w_i)
8  return s[n, W]
```

(d) [5 points] *Maximal Subarray.* The problem: given an array of a mix of positive and negative integers, find indices i and j such that the subarray from i to j, inclusive, has the maximal sum. Here, the question is about the MaxSubarray function below; the MaxCrossing function is a helper function; your runtime analysis should consider the cost of MaxCrossing.

```
MaxSubarray(A, low, high)
1  if low = high then return A[low]
2  else mid = ⌊(high + low/2⌋
3       a = MaxSubarray(A, low, mid)
4       b = MaxSubarray(A, mid + 1, high)
5       a = MaxCrossing(A, low, mid, high)
6  return max(a, b, c)
```

```
MaxCrossing(A, low, mid, high)
 1  sum = 0
 2  leftSum = −∞
 3  for i = mid downto low do
 4       sum = sum + A[i]
 5       if sum > leftSum then leftSum = sum
 6  sum = 0
 7  rightSum = −∞
 8  for i = mid + 1 upto high do
 9       sum = sum + A[i]
10       if sum > rightSum then rightSum = sum
11  return leftSum + rightSum
```

Alas, Fruitcake Frank's life of crime finally caught up to him. (Really, he definitely should have chosen a retirement location in a country without an extradition treaty with the United States. Hawai'i? He was bound to be caught.)

Anyway, he finds himself landed in prison. But it turns out that all of his algorithms training has made him very popular in the prison population...

Problem 2: Feeding the Prisoners [20 points]
Fruitcake Frank's first in-prison job assignment is to manage the contracts for the prison's food suppliers. The prison contracts alternately with two different suppliers, *Jailhouse Grub* and *Lockup Eats*. For each of n weeks, the prison needs to purchase some number of meals, though the number may vary across weeks. That is, in week i, the prison needs to purchase m_i meals. Both suppliers have the same per-meal cost (c), but they each have a different way of doing a contract, which results in different fee schedules.

Jailhouse Grub offers contracts for exactly four week blocks and charges a fixed fee per week of the contract, regardless of the number of meals they have to supply (we'll call *Jailhouse Grub's* fee f_A). *Lockup Eats* charges a fixed fee per meal supplied but will contract one week at a time (we'll call *Lockup Eats's* fee f_B). That is, every contract with *Jailhouse Grub* must be done for four consecutive weeks, but *Lockup Eats* can be contracted for individual weeks.

Fruitcake's job is, given the number of meals needed for some n weeks, to figure out a *schedule* of contracts with *Jailhouse Grub* and *Lockup Eats* that is most cost efficient for the prison. For example, assume that the costs of each meal is $c = \$10$, and the fees are: $f_A = \$2000$/week and $f_B = \$2$/meal. Given the following number of meals needed for the next $n = 6$ weeks: [800, 1000, 1500, 1200, 900, 700], the optimal schedule of contracts with *Jailhouse Grub* and *Lockup Eats* is [LE, JG, JG, JG, JG, LE], which results in the following costs:

Week(s)	# Meals	Supplier	Cost of Meals	Fee
1	800	*Lockup Eats*	$\$8000 = (800 \times c)$	$\$1600 = (800 \times f_B)$
2-5	4600	*Jailhouse Grub*	$\$46000 = (4600 \times c)$	$\$8000 = (4 \times f_A)$
6	700	*Lockup Eats*	$\$7000 = (700 \times c)$	$\$1400 = (700 \times f_B)$

You should give a polynomial time algorithm to create the schedule of contracts for n weeks, given c, f_A, f_B and the array $m[1..n]$. Your algorithm should return the schedule (i.e., which supplier to contract with on each week) as well as the total cost for all n weeks.

State and briefly justify the runtime of your algorithm.

Problem 3: Scheduling Guards [30 points]
The prison's guards have become really high maintenance in demanding shift schedules that match their personal lives. To try to get in good with the guards, Fruitcake Frank volunteers to come up with an algorithm that takes the guards' requested work shifts and tries to generate a schedule that respects those requests but keeps the prison staffed.

Here are the inputs to the problem:

- There are n shifts, and for any shift i, there must be *exactly* g_i guards scheduled (no more, no fewer). (Notice that different shifts may have different staffing requirements.)
- There are k guards, and each guard provides a list of the shifts he's willing to work (so guard j provides list L_j).

And here are the goals of the solution:

- Fruitcake's algorithm should produce a schedule for each guard that is a list of the shifts the guard has been assigned to. (For guard j, we'll call this second list S_j.)
- For all j, $S_j \subseteq L_j$. That is, no guard is scheduled to work a shift that he didn't request.
- If we examine all of the S_j for all k guards, we will find that *exactly* g_i guards are scheduled for shift i.
- Since the prison is on a shoestring budget and does not want to pay too much overtime, the guards should all be assiged to very near the same number of shifts. That is, no guard should be assigned more than $\lceil (\sum_{0 \le i \le k} g_i)/k \rceil$ shifts.

(a) [15 points] Take the role of Fruitcake Frank and define a polynomial-time algorithm that implements this scheduling system. Specifically, the algorithm should take the numbers $g_1, g_2, \ldots, g_n$ and the lists $L_1, L_2, \ldots, L_k$ and do one of the following two things:

- Return a set of schedules $S_1, S_2, \ldots, S_k$ that satisfies the above goals or
- Report (correctly) that there is no set of schedules $S_1, S_2, \ldots, S_k$ that satisfies all four of the goals.

State the runtime of your algorithm (in terms of the input sizes to the orignal problem) and briefly justify your statement.

(b) [15 points] After employing this system for just a small amount of time, the prison finds that the guards are providing overly-restrictive sets of shifts they're available to work, in an effort to manipulate the schedule to their liking. So, over time, it becomes the case that the algorithm routinely returns that it's just not possible to generate the schedules.

Fruitcake Frank decides to relax the restrictions of the problem just a little bit. Specifically, he adds a parameter to capture the fact that a guard can be scheduled for a limited number of "extra" shifts (i.e., outside of those he specified as available). We'll call this parameter e and impose the constraint that $e > 0$. We relax the goals of the problem to the following:

- (*same as before*) Fruitcake's algorithm should produce a schedule for each guard that is a list of the shifts the guard has been assigned to. (For guard j, we'll call this second list S_j.)
- (*modified*) For all j, S_j contains at most e shifts that the guard did not request.
- (*same as before*) If we examine all of the S_j for all k guards, we will find that *exactly* g_i guards are scheduled for shift i.
- (*same as before*) Since the prison is on a shoestring budget and does not want to pay too much overtime, the guards should all be assiged to very near the same number of shifts. That is, no guard should be assigned more than $\lceil (\sum_{0 \leq i \leq k} g_i)/k \rceil$ shifts.

Now you (as Fruitcake Frank) should describe a polynomial time algorithm that implements this revised scheduling system. It should take the numbers $g_1, g_2, \ldots, g_n$, the lists $L_1, L_2, \ldots, L_k$, and the parameter e and do one of the following:

- Return a set of schedules $S_1, S_2, \ldots, S_k$ that satisfies the revised goals or
- Report (correctly) that there is no set of schedules $S_1, S_2, \ldots, S_k$ that satisfies all four of the revised goals.

State the runtime of your algorithm (in terms of the input sizes to the orignal problem) and briefly justify your statement.

Problem 4: Swapping Contraband [30 points]
After a few months in prison, life is easier for Fruitcake Frank. He's used his algorithms prowess to endear himself to the guards, so they're willing to look the other way with regards to minor transgressions. Fruitcake is surprised one day when he's assigned a new cell mate. It's Cupcake Carl! Before long, the two old associates have concocted a plan to start a micro-economy in the prison. Fruitcake and Cupcake want to get (what else?) candies by trading other inmates for hot commodities (cigarettes, ramen noodles (seriously, that's apparently a thing, Google it later), postage stamps, etc.).

More specifically, Fruitcake and Cupcake have n items they can trade away, $I_1, I_2, \ldots, I_n$. Each item is indivisible and can only be traded to one other inmate. There are m other inmates, and each of those inmates can place bids for one or more items. That is, the i^{th} bid specifies some subset of items, S_i and the number of candies c_i the bidder is willing to trade for that subset of items. Each bid is therefore a pair (S_i, c_i).

Fruitcake and Cupcake need an algorithm that *accepts* and *rejects* bids. If a bid i is accepted, the bidder gets to take all of the items in the set S_i and must give Fruitcake and Cupcake c_i candies. It therefore naturally follows that the sets S_i and S_j of two accepted bids i and j cannot have any items in common. Fruitcake and Cupcake want an algorithm that maximizes the number of candies they get. That's the *optimization* version of the **Contraband Swap** problem.

(a) [5 points] Fruitcake suggests that there is a somewhat simpler *decision* version of this problem. Succinctly state the decision version of the **Contraband Swap** problem.

(b) [10 points] Because of his work in algorithms Fruitcake quickly realizes that this looks like an NP-Complete algorithm. When he mentions that, Cupcake vaguely remembers hearing something about that concept in a class he dozed through. He eagerly says the following:

> *We can prove that* **Contraband Swap** *is an NP-Complete problem by showing that* **Contraband Swap** *can be reduced to an existing NP-Complete problem in polynomial time!*

Is Cupcake right? Why or why not? (Hint: what are the implications of this reduction?)

(c) [15 points] Prove that your decision version of **Contraband Swap** is an NP-Complete problem. (Don't forget the first step!)