There are six problems on the exam, each worth 40 points; you can **pick any five** to complete. If you do all six, the problem with the lowest score will be dropped. There are also three optional extra credit problems worth a total of 30 points.

In preparing your solutions to the exam, you are allowed to use any sources, including your textbook, other students and professors, previous homeworks and solutions, or any sources on the Internet. You may ask me for feedback on potential solutions, but I will not give you any hints.[1] Of course, I am also happy to answer general questions, go over homework problems, or answer clarifying questions about exam problems.

[1] Think of me as a polynomial-time certifier for exam problems.

The exam will take place in MC Reynolds 317 from 2–5pm on Friday, December 13. You are not allowed to bring any notes, textbooks, calculators, or any other resources with you to the exam. Bring only something to write with; I will provide a fresh copy of the exam, paper for writing your solutions, and scratch paper.

As usual, to "design and analyze" an algorithm means to (a) **describe** the algorithm, (b) **prove/justify** its correctness, and (c) **analyze** its asymptotic running time. Full credit will only be given for the most efficient possible algorithms. Algorithms must be clearly explained (using pseudocode if appropriate) in sufficient detail that another student could take your description and turn it into working code. You may freely cite (without proof) any theorems proved in class or on homework assignments (whether you completed the relevant homework question or not), or use algorithms from class or homework assignments as subroutines.

Acknowledgment Questions 3 and 5 are adapted from Jeff Erickson (`http://jeffe.cs.illinois.edu/teaching/algorithms/`).

Question 1 (40 points)**.** You are driving from Conway to Millinocket, Maine, to attend your third cousin's wedding. You are on a budget, so you want to spend the least amount of money possible. You have a map which shows all the cities in the whole country as well as all the roads connecting the cities. Each road has a nonnegative integer cost (the amount of money (in cents) you would spend on gas, tolls, bribes, *etc.* to drive on that road). Note, however, that some roads are one-way and some are two-way; two-way roads may have a different cost in each direction (for example, driving *up* a road into the mountains uses more gas than driving *down* the same road). Each city besides Conway also has a (nonnegative integer) cost: for each city you know (from past experience) exactly how much money you would spend there, *e.g.* on food, amusement parks, and other local attractions. Although you are on a budget, you are also impulsive; you always stop in each city you come to, and the only way you can avoid spending money in a city is to avoid visiting it altogether. Design and analyze an algorithm to find the cheapest route from Conway to Millinocket.

Figure 1: Millinocket

Note, for this problem just describe and analyze the most efficient algorithm; don't worry about the practical feasibility of running your algorithm on a graph of the entire country.

(10 points, **Extra credit**). What if you are allowed to skip up to every other city? That is, when coming to a city you can drive right by without stopping (and hence without paying the cost for that city), but if you do this you definitely have to stop in the next city you come to (your bladder is only so big); you can never skip two cities in a row.

Question 2 (40 points)**.** Recall the SUBSET-SUM problem we considered in class: given a set $X = \{x_1, x_2, \ldots, x_n\}$ of nonnegative integers and a target sum S, is there a subset of X which adds up to exactly S? Using dynamic programming, we saw how to solve this in $O(nS)$, but this is not good if S is very large. The best algorithm we have seen so far that only depends on n is the brute force $\Theta(n \cdot 2^n)$ algorithm, but it turns out it is possible to do better than this.

Figure 2: $\subseteq$

Let's begin by considering a related problem, ALL-SUBSET-SUMS: given a set X, generate a *sorted list* of *all* its distinct subset sums. For example, given the set $X = \{1, 2, 3, 4\}$ as input, the correct output would be the sorted list of all integers from 0 up to 10; or given the set $X = \{5, 17\}$ as input, the correct output would be the list $[0, 5, 17, 22]$.

(a) Describe a brute force algorithm to solve ALL-SUBSET-SUMS and analyze its running time.

(b) Describe and analyze an algorithm to solve ALL-SUBSET-SUMS in only $O(2^n)$ time.

Hint for (b): if X is not empty, start by splitting X into an arbitrary element and the rest, $X = \{x_0\} \cup X'$.

(c) Given the existence of an $O(2^n)$-time algorithm to solve ALL-SUBSET-SUMS, describe and analyze an algorithm to solve SUBSET-SUM in $O(2^{n/2})$ time.

(d) Prove that $2^{n/2}$ is $o(2^n)$.

Question 3 (40 points)**.** A palindrome is any string that is equal to its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`.

Design and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome. For example, the longest palindrome subsequence of

`MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM`

is `MHYMRORMYHM`, so given that string as input, your algorithm should output the number 11.

Figure 3: `AMANAPLANACATACANALPANAMA`

Question 4 (40 points)**.** Your third cousin has asked you to organize a group hike up Mount Katahdin for other family members who will be in town for the wedding. This would be simple, except for two problems. First, your third cousin didn't ask you until the last minute, so you have only 24 hours to organize the hike. The second problem is that some family members hate each other; you can't include any two people who hate each other in the hiking group (because they would have a terrible time and ruin your third cousin's wedding celebrations). However, the more people who come on the hike, the more fun it will be. You therefore want to find the largest possible group you can invite such that no two people in the group hate each other.

Figure 4: Mt. Katahdin

(a) Your cousin gives you a list of the 25 family members who might want to go on the hike, along with a list of pairs of family members who hate each other. What should you do? Describe and analyze an appropriate algorithm you could use to find an optimal hiking group given the amount of time you have.

Hint: your answer to part (a) should not need to be very long. As a rule of thumb you should assume that your computer can do about 10^8 operations per second.

(b) A few hours later, after you have come up with a list of people to invite, your third cousin sheepishly tells you that they forgot to mention the 3975 other family members invited by their mother, for a total of 4000 family members who might want to come on the hike. Fortunately, they remembered something else as well: there are two branches of the family, and people only hate people from the *other* branch. In other words, no two people from the same family branch ever hate each other. Your cousin also feels a little bad for asking you to do all of this at the last minute, so they tell you that this time it's fine if you just tell them the *size* of an optimal hiking group, instead of coming up with an actual list of people to invite.

Your third cousin gives you the new list of 4000 family members and the list of pairs of people who hate each other; however, they unfortunately don't remember which branch of the family each person is from (and there's no way to tell from their names). Now what should you do? Describe and analyze an appropriate algorithm you could use to find the *size* of an optimal hiking group from the new data.

Hint: look up Kőnig's Theorem; you may cite it without proof.

(c) (10 points, **Extra credit**) Extend your answer to part (b) to find an actual list of people in an optimal hiking group instead of just the size.

Question 5 (40 points)**.** Suppose that instead of powers of two, we represent integers as a sum of distinct Fibonacci numbers. In other words, instead of an array of bits, we keep an array of *fits* (*fi*bonacci dig*its*), where each fit is either 0 or 1, and the i^{th} least significant fit indicates whether the sum includes the i^{th} Fibonacci number F_i. For example, the fit-string 101110_F represents the number

$$F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14.$$

Design and analyze an algorithm to increment a fit-string in $\Theta(1)$ amortized time, assuming that a fit-string counter starts at 0 and is repeatedly incremented. As with the example above, assume that $F_1 = F_2 = 1$, that the array of fits is indexed starting from 1, and that the i^{th} index of the array stores the i^{th} fit. Also observe that unlike binary, a given number n does not necessarily have a unique representation as a fit-string; it does not matter which representations are generated by your algorithm.

Figure 5: F_3

Question 6 (40 points)**.** A *dominating set* for an undirected graph $G = (V, E)$ is a subset of the vertices $D \subseteq V$ such that for every vertex $v \in V$, either $v \in D$, or v is adjacent to at least one member of D.

For example, the marked black vertices form a dominating set in the graph at the top of Figure 6; every other vertex is adjacent to at least one of the marked vertices. On the other hand, the marked vertices in the bottom graph do not form a dominating set, since the lower-right vertex is not adjacent to any marked vertex.

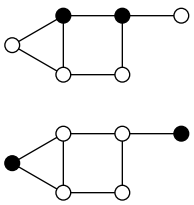

Figure 6: Examples of dominating and non-dominating sets

The DOMINATING SET problem asks, given a graph G and a positive integer k, does G have a dominating set of size $\leq k$? Prove that DOMINATING SET is NP-complete.

Question 7 (10 points, **Extra credit**)**.** An undirected graph $G = (V, E)$ is 3-*colorable* if every vertex can be assigned one of three colors such that no two adjacent vertices have the same color. For example, the graph on the left in Figure 7 is 3-colorable: you can check that only three different colors are used for the vertices, and no two adjacent vertices have the same color. On the other hand, the graph on the right is not 3-colorable; there is no way to ensure that every pair of adjacent vertices has different colors if we are only allowed to use 3 colors (in fact, for this graph 5 colors would be necessary).

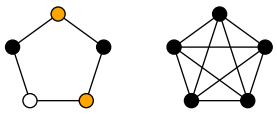

Figure 7: Examples of 3-colorable and non-3-colorable graphs

For your birthday, your fairy godmother has given you a strange gift: a mysterious black box with a slot on one end and two lights on top, one green and one red. When you put a drawing of an undirected graph into the slot, one of the two lights comes on: green if the graph is 3-colorable, and red if it isn't.[2] Curiously, a light always comes on right away, no matter how many vertices and edges the input graph has.

Describe a polynomial-time algorithm, making use of your magic box, which takes a graph G as input and either constructs a valid 3-coloring for G, or correctly reports that G is not 3-colorable.

[2] Your fairy godmother didn't actually tell you this; how you figured it out is an exciting tale of adventure, involving a circumnavigation of the globe, a talking unicorn named Charles, and a list of all the real numbers, but unfortunately the tale is too long to fit on this exam.