

There are five problems on the exam; you may choose to complete **any four**. If you do all five problems, you will receive credit for the exam as long as you receive credit for at least four of the problems.

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 Internet¹. For the final exam, I will be happy to **offer feedback on potential exam solutions**, but I will not offer assistance.

The exam will take place in MC Reynolds 314 from 8:30–11:30am on Thursday, December 12. 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/>).

¹ Caution: I fed this entire exam to ChatGPT and graded the output. It scored an average of less than 2.0 per problem; its solutions *sounded* good but made no sense.

Question 1. 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.

Question 2. Recall the SUBSET-SUM problem we considered in class: given a set $X = \{x_1, x_2, \dots, x_n\}$ 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 $\Theta(nS)$, 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.

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 $[0, 5, 17, 22]$.

- Describe a brute force algorithm to solve ALL-SUBSET-SUMS and analyze its running time.
- Describe and analyze an algorithm to solve ALL-SUBSET-SUMS in only $O(2^n)$ time.
- 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.
- Prove that $2^{n/2}$ is $o(2^n)$.

Figure 2: \subseteq

Hint for (b): if X is not empty, start by splitting X into an arbitrary element and the rest, $X = \{x_0\} \cup X'$. What if you first solve ALL-SUBSET-SUMS on X' ?

Question 3. 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.

Question 4. 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.

- (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.
- (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.



Figure 3:
AMANAPLANACATACANALPANAMA



Figure 4: Mt. Katahdin

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.

Hint: look up König's Theorem (the one to do with graph theory); you may cite it without proof.

Question 5. 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* (*fibonacci digits*), 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