Solutions to Homework 3

This homework has 7 questions, for a total of 100 points and 5 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) If you haven't already, carefully read Handout 3 about "giving an algorithm," and follow it in your solutions.

(b) If you need a quick refresher on *matrix multiplication*, check out this link.

(c) If you need a quick refresher on *logic symbols*, check out this link.

(d) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't do last week's homework, choose a problem from it that looks challenging to you, and in a few sentences, explain the key ideas behind its solution in your own words.

Solution:

2. **The power of divide and conquer.**

In cryptography, fast and efficient computation of modular exponentiation is essential for encryption algorithms like RSA (Stay tuned! You will learn this in the cryptography unit). In the first three parts of this problem, we will explore a divide-and-conquer algorithm to compute $a^b \bmod n$ efficiently, for some positive integers a, b, and a modulo $m \geq 2$.

(3 pts) (a) Consider the following "naïve" algorithm for modular exponentiation.

Input: positive integers a, b, and modulo $m \geq 2$
Output: $a^b \bmod m$

```
1: function NAÏVE(a, b, m)
2:     prod ← 1
3:     for i = 1, 2, ..., b do
4:         prod ← (prod · a) mod m
5:     return prod
```

Determine if NAÏVE is efficient with respect to the input size. Briefly justify you answer.

Solution:

(5 pts) (b) Prove that for any positive integers p, q, and modulo $m \geq 2$,

$$pq \bmod m = ((p \bmod m)(q \bmod m)) \bmod m.$$

Fill in the blank in the following statement such that the result is true, then prove it, possibly using the above identity:

$$\text{if } b \text{ is even, then } \quad (\_\_\_\_\_ \bmod m)^2 \bmod m = a^b \bmod m.$$

Solution:

(10 pts) (c) Give an $O(\log b)$ algorithm to compute $a^b \bmod m$. As usual, include correctness proof and runtime analysis in your solution. You may cite the result in Part b in your correctness proof.

Solution:

(6 pts) (d) *Matrix exponentiation*, denoted as A^b, represents multiplying an $n \times n$ square matrix A by itself b times. Just like with scalar exponentiation, efficiently computing $A^b \bmod m$ is essential for many applications in areas such as cryptography and graph theory.

Briefly explain (in 1-2 sentences, do not give pseudocode) how you would modify your algorithm in Part c to compute $A^b \bmod m$, where A is a square matrix. Briefly explain the correctness of your modified algorithm and include a runtime analysis in your solution. The running time should be bounded in terms of two parameters b and n.

You may use the following fact without proof: Let P and Q be two $n \times n$ matrices with positive integers entries, then

$$PQ \bmod m = (P \bmod m)(Q \bmod m) \bmod m.$$

For simplicity, you may also make the following assumptions:

- The runtime of multiplying two $n \times n$ matrices is $O(n^3)$[1].

[1] As of January 2025, the best announced bound on the asymptotic complexity of a matrix multiplication algorithm is $O(n^{2.3713})$ given by Alman, Duan, Williams, Xu, Xu, and Zhou. For this assignment, you may just use the "old-school" method of multiplying square matrices, which takes $O(n^3)$.

- All entries in A are non-negative and are smaller than m. In other words, they are all "preprocessed" by taking modulo m before passing into your algorithm as input.

You algorithm should use $O(\log b)$ numerical operations under these assumptions.

Solution:

(4 pts) (e) The dynamic programming method for computing the kth Fibonacci number F_k requires $O(k)$ numerical operations. In this part, you will show that it is possible to use $O(\log k)$ numerical operations, using a divide-and-conquer approach.

Describe, in clear and precise English, how you would modify and/or use your algorithm in Part d to compute $F_k \bmod m$ using $O(\log k)$ numerical operations. Justify your answer.

Hint: Keep in mind that your solution for Part d computes $A^b \bmod m$ using $O(\log b)$ numerical operations. You may also want to use the following 2×2 matrix:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Solution:

3. **The peak of divide-and-conquer.**

Consider an array $A[1, \ldots, n]$ of integers. The *peak* of the array is the value of the largest element in the array. For example, the peak of the array $A = [281, 280, 376, 370]$ is 376.

(10 pts) (a) We say that an array C is a *circular shift* of an array A if it starts with some position in A, continues to the end, and "wraps around" to include the remaining elements from the beginning of A. In other words, there exists an integer k between 1 and n (inclusive) such that

$$C = [A[k], A[k+1], \ldots, A[n], A[1], A[2], \ldots, A[k-1]]$$

For example, if $A = [280, 281, 370, 376]$, then $C = [370, 376, 280, 281]$ is a circular shift of A with $k = 3$.

Assuming A consists of distinct integers and is sorted in ascending order, i.e., $A[1] < A[2] < \ldots < A[n]$, give an $O(\log n)$ algorithm that takes in a circular shift of A as input, and returns the peak of A. As usual, include correctness proof and runtime analysis in your solution.

Solution:

(10 pts) (b) *For this part of the problem, array A is* not *assumed to be sorted and* may *contain duplicated elements.* A *local peak* of an integer array $A[1, \ldots, n]$ is an element $A[i]$ that is no smaller than its adjacent elements. More precisely, $A[i]$ is a local peak if $A[i] \geq A[i-1]$ *and* $A[i] \geq A[i+1]$. For convenience, we say that $A[0] = A[n+1] = -\infty$ to deal with the endpoints of the array. Note that every array has at least one local peak, and may have multiple local peaks. For example, the array $A = [281, 280, 376, 370]$ has two local peaks, namely, 281 and 376.

Give an $O(\log n)$ algorithm that takes in an integer array A as input, and return a local peak in the array (if there are multiple, you can just return *one* of them). As usual, include correctness proof and runtime analysis in your solution.

Solution:

4. **Warming up with recurrence relations.**

We will soon be *hiking* through the mountains of dynamic programming, so let's warm up with some practice on recurrence relations. For each of the following problems, give, with justification, a recurrence relation (including base cases) that are suitable for a dynamic programming solution to the corresponding problem. Start your solution with a clear English description of what the recurrence relation represents.[2] For example, "let $L(i)$ be the length of the longest increasing subsequence ending at $A[i]$."

(8 pts) (a) Given a staircase you can traverse with two distinct step sizes a and b (i.e. you can take a steps at a time or b steps at a time), determine the number of ways you can reach the kth step of the staircase. For example, when $a = 1$ and $b = 4$, there are 4 possible permutations of steps to reach the $k = 6$th step.

Solution:

(8 pts) (b) Given a rod of length n and an array of prices $P[1, \ldots, n]$, where $P[i]$ is the price for a piece of length i, determine the maximum profit one could make by cutting up the rod and selling the pieces. For example, for $n = 4$ and $P = [1, 6, 7, 4]$, one would cut the rod into two pieces each of length 2, and make a profit of 12.

Solution:

(8 pts) (c) Given a set S of n non-negative integers and an integer target sum T, determine whether there exists a subset of S which sums to T. For example, when $S = \{3, 9, 2, 4\}$ and $T = 7$, the instance[3] (S, T) is a true instance. When $S = \{3, 9, 2, 4\}$ and $T = 8$, the instance (S, T) is a false instance.
Hint: Your recurrence can have multiple inputs (e.g. $f(a, b)$), which you may find helpful for this problem.

Solution:

5. **Hiking in Michigan!**

Uma decides to take a road trip around Michigan! She will start her road trip in Ann Arbor and end in Hancock, MI, the northernmost city in Michigan. Along her journey, there are n hotels she can stop in for the night, located at positive distances $d_1 < d_2 < \cdots < d_n$ from Ann Arbor. As a convention, we treat Ann Arbor itself as the "0th hotel," i.e., $d_0 = 0$.

[2] This is a good practice in general, to avoid confusions for both *you* and those reading your solution.
[3] An *instance* of a decision problem (you will learn this in the next unit) is a specific input or case that needs to be evaluated to determine whether the answer is "yes" or "no" based on the problem's requirements.

She can stop only at these specified hotels, but may choose which one(s) to spend her night(s) at, and which ones to skip. She must end her journey at the last hotel, located in Hancock, at distance d_n from Ann Arbor. She will never drive backwards.

The trip from Ann Arbor to Hancock is 540 miles. Uma doesn't want to drive too much or too little in a day, so she is aiming to go about 200 miles per day. However, due to the distances between the hotels, this might not be possible. If she drives x miles on a given day, she will face a "penalty" of $(200 - x)^2$ for that day.

For example, if $d_1 = 190$, $d_2 = 260$, and $d_3 = 540$, then we have the following possible choices of hotels (by their indices) and the associated penalties:

$$\begin{aligned} 1, 2, 3 &\to (200 - 190)^2 + (200 - 70)^2 + (200 - 280)^2 && = 23400 \\ 2, 3 &\to (200 - 260)^2 + (200 - 280)^2 && = 10000 \\ 1, 3 &\to (200 - 190)^2 + (200 - 350)^2 && = 22600 \\ 3 &\to (200 - 540)^2 && = 115600 \end{aligned}$$

Therefore, the minimum possible total penalty in this example is 10000. To achieve this, Uma should drive to the second hotel (at 260 miles) on the first day, then drive the remaining 280 miles to Hancock on the second day.

The goal in this problem is to devise an efficient algorithm that, given an array $A[1, \ldots, n]$ of the distances $d_1, \ldots, d_n$ (with $d_0 = 0$), determines which hotel(s) to stop at so as to minimize the total penalty incurred on the trip. For this purpose, define a function $p(i)$ that represents the minimum possible total penalty when starting from Ann Arbor and ending at hotel i.

(8 pts) (a) Give, with justification, a recurrence relation for $p(i)$ that is suitable for a dynamic-programming solution to the problem. Be sure to include the base case(s).

Solution:

(10 pts) (b) Give a (bottom-up) dynamic-programming algorithm that computes the minimum total penalty. Also briefly describe, in clear and precise English, how this algorithm could be extended to find a corresponding set of hotels to stop at.

Follow the "DP recipe" from lecture to help guide your approach!

In this and all other DP problems, your correctness argument for the algorithm can rely on the correctness of your previously given recurrence. But the argument should also briefly state why the algorithm fills the DP table in a valid order, and why the final output is correct.

Solution:

(5 EC pts) 6. **Trophy testing (optional extra credit).**

This is a good problem for those who want a challenge. It will be graded with high standards and not much partial credit, and no regrades will be done.

This past weekend, during the National Championship Celebration at Crisler Center, J.J. McCarthy decided to toss the AFCA Coaches' Trophy to Mike Sainristil. Luckily, Mike

caught it (as he does with most footballs that come in his vicinity), but the Michigan athletic department was very concerned, since this trophy is made of Waterford Crystal and valued at more than $34,130. As a result, it has asked researchers at the University's Materials Science and Engineering department to conduct research on Waterford Crystal.

To test its strength, blocks of Waterford Crystal will be dropped from various heights between 1 and n inches (inclusive), for some n of interest. Waterford Crystal is said to have *least breaking height* (LBH) k if a block of it will break when dropped from a height of k inches or more, but will not break if dropped from any fewer number of inches. If it does not break even when dropped from a height of n inches, we say that the LBH is "more than n," denoted "$> n$" for convenience. All blocks have the same LBH; the LBH is a property of the material itself, and doesn't vary from block to block.

Your task is to determine the LBH of Waterford Crystal. Since it's a very expensive material, you are given just two blocks. Fortunately, any two blocks of Waterford Crystal have the same LBH. Unfortunately, once a block breaks, it is unusable for future testing.

It is easy to see that if you had only one block, you would have no choice but to drop the block from 1 inch, then from 2, and so on, until the block breaks (or it doesn't even break from a height of n inches). This is because if you were to skip some height, then you would risk prematurely breaking the block without knowing the exact height from which it would have first broken.

But, with two blocks, you can do better! In this problem you will determine the best way to utilize two blocks to determine the LBH, using the fewest number of drops. Instead of expressing the solution as "if I want to determine the LBH among the $n+1$ possibilities, I can find it using at most $d = d(n)$ drops," it will be cleaner to express the solution *inversely*, as: "With d drops, I can determine the LBH among $n(d)+1$ different possibilities," where $n = n(d)$ is a function of d. (Recall that "$> n$" is the extra case representing "greater than n.")

(a) Suppose you are given two blocks of Waterford Crystal and allowed no more than d drops, from heights of your choice (which can depend on the results of previous drops). What is the largest value of $n = n(d)$ for which you can determine the LBH among the $n+1$ different possibilities? Describe your method clearly and concisely in English, give a recurrence and base case for $n(d)$, and give an exact (not asymptotic) closed-form solution.

Solution:

(b) Give a short explanation why your algorithm is *optimal*; i.e., why *any algorithm* that uses 2 blocks and at most d drops cannot be guaranteed to work for a value larger than $n = n(d)$, for the $n(d)$ you gave in the previous part. A few sentences of intuitive but logically valid explanation suffice here, but if you wish, you're welcome to prove this rigorously.

Solution: