Solution: Quiz 3

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on the top of every page of this quiz booklet.
- You have 60 minutes to earn a maximum of 60 points. Do not spend too much time on any one problem. Skim them all first, and work on them in an order that allows you to make the most progress.
- **You are allowed three double-sided letter-sized sheet with your own notes**. No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. Pages will be scanned and separated for grading. If you need more space, write "Continued on S1" (or S2) and continue your solution on the referenced scratch page at the end of the exam.
- Do not waste time and paper rederiving facts that we have studied in lecture, recitation, or problem sets. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. Be sure to argue that your **algorithm is correct**, and analyze the **asymptotic running time of your algorithm**. Even if your algorithm does not meet a requested bound, you **may** receive partial credit for inefficient solutions that are correct.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points
0: Information	2	2
1: Pseudo-Circling	3	3
2: Relic Rescue	1	17
3: Road Trip	1	19
4: Cooking Constraints	1	19
Total		60

Name: _____

School Email: _____

Problem 0. [2 points] **Information** (2 parts)

(a) [1 point] Write your name and email address on the cover page.
Solution: OK!

(b) [1 point] Write your name at the top of each page.
Solution: OK!

Please solve problems (2), (3), and (4) using **dynamic programming**. Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient dynamic programs will be awarded significant partial credit.

Problem 1. [3 points] **Pseudo-Circling**

Indicate whether the given running times of each of problems (2), (3), and (4) are polynomial or pseudo-polynomial by circling the appropriate word below. **One can answer this question without actually solving problems (2), (3), and (4).** (1 point each)

(a) Problem 2: Relic Rescue Polynomial Pseudo-polynomial
Solution: Pseudo-polynomial

(b) Problem 3: Road Trip Polynomial Pseudo-polynomial
Solution: Polynomial

(c) Problem 4: Cooking Constraints Polynomial Pseudo-polynomial
Solution: Polynomial

Problem 2. [17 points] **Relic Rescue**

Archaeologist Montana Smith has discovered an ancient tomb housing an immovable treasure chest containing n valuable **relics**. Unfortunately, the chest is known to be (r, g)**-booby-trapped**:

- if strictly more than $r < n$ relics are removed from the chest, or
- if strictly more than $g > n$ grams of total relic weight is removed from the chest,

the entire vault will **collapse**. For each relic i, Montana has measured its **weight** w_i in positive integer grams and appraised its estimated **museum value** v_i in positive integer dollars. Given this relic information and positive integers r and g, describe an $O(rng)$-time algorithm to determine the maximum total value of relics Montana can remove without causing the vault to collapse.

Solution:

1. **Subproblems**
   - Assume the n relics are labeled with the integers $\{1, \ldots, n\}$
   - $x(i, r', g')$: maximum total value removable by removing at most r' relics and at most g' grams from the prefix of relics $\{1, \ldots, i\}$
2. **Relate**
   - Guess whether to remove item i from the chest or not
   - $x(i, r', g') = \max \left\{ \begin{array}{ll} v_i + x(i-1, r'-1, g'-w_i) & \text{if } g' \geq w_i \\ x(i-1, r', g') & \text{always} \end{array} \right\}$
   - Subproblem $x(i, r', g')$ only depends on subproblems with strictly smaller i so acyclic
3. **Base**
   - No more relic value achievable when any of three arguments is zero
   - $x(i, r', g') = 0$ when any of i, r', g' is 0
4. **Solution**
   - $x(n, r, g)$ is the maximum total value removable without collapse by removing at most r relics and at most g grams, choosing among all the relics, as desired
5. **Time**
   - # subproblems: $\leq (n+1)(1+r)(1+g) = \Theta(rng)$
   - work per subproblem: $O(1)$ time
   - $O(rng)$ total running time, which is polynomial in n and r, and pseudo-polynomial in g

Common Mistakes:

- Missing a weight or relics constraint in relation
- Missing one or more base cases
- Mixing up prefix and suffix approaches

Problem 3. [19 points] **Road Trip**

Joy Ryder is planning a road trip that will last k days. She has already determined her driving route: she will visit $n+1 > k$ cities, $(c_0, \ldots, c_n)$ in that order, starting in city c_0 and ending in city c_n. Joy wants to split her driving as evenly as possible among the k days. Given the driving distance d_i between cities c_{i-1} and c_i for every $i \in \{1, \ldots, n\}$, describe an $O(kn^2)$-time algorithm to determine which $k-1$ cities Joy should stay in overnight between days, so as to minimize the maximum total distance she drives on any day of her trip.

Solution:

1. **Subproblems**
   - $x(i, k')$: minimum max daily distance to drive prefix from city c_0 to city c_i in k' days
2. **Relate**
   - Guess the last city $i' \in \{0, \ldots, i\}$ Joy stays overnight
   - $x(i, k') = \min \left\{ \max \left(x(i', k'-1), \sum_{j=i'+1}^{i} d_j \right) \middle| \, i' \in \{0, \ldots, i\} \right\}$
   - Subproblem $x(i, k')$ only depends on subproblems with strictly smaller k' so acyclic
3. **Base**
   - $x(0, k') = 0$ for $k' \geq 0$ (no distance left to drive)
   - $x(i, 0) = \infty$ for $i > 0$ (no more days can be used, so cannot reach destination)
4. **Solution**
   - $x(n, k)$ is the minimum max daily distance to drive entire route in k days as desired
   - Store parent pointers to reconstruct overnight cities in an optimizing solution
5. **Time**
   - # subproblems: $\leq (n+1)(1+k) = \Theta(nk)$
   - work per subproblem: naively $\Theta(i^2) \subset O(n^2)$ time
   - Precompute each $\sum_{j=i'+1}^{i} d_j$ to make work $O(n)$
     (a) **Subproblems:** $s(i', i) = \sum_{j=i'+1}^{i} d_j$ the distance from city i' to city i
     (b) **Relate:** $s(i', i) = d_i + s(i', i-1)$ (depends on smaller i)
     (c) **Base:** $s(i', i') = 0$
     (d) **Solution:** $s(i', i)$ for all $0 \leq i' \leq i \leq n$
     (e) **Time:** $O(n^2)$ subproblems $\times O(1)$ time per, $O(n^2)$ time total
   - $O(kn^2)$ total running time, which is polynomial in n and k (since $k < n+1$)

Common Mistakes:

- Not reconstructing a sequence of cities, e.g., by following parent pointers
- Treating an $\Theta(n)$-length sum as a $O(1)$-time operation
- Over-parameterizing subproblems
- Combining subproblems incorrectly, e.g., sums instead of min of max

Problem 4. [19 points] **Cooking Constraints**

TV Chef Rordon Gamsey is participating in a cooking contest to prepare a known three-course meal. Each course has a provided recipe containing an ordered **ingredient list**: the sequence of ingredients that will be needed to make the course, listed in the order that they will be **used**[1]. To make the contest more interesting, all ingredients will be stored in a personal pantry, and Gamsey may use at most one ingredient from the pantry at a time[2]. Given ingredient lists L_1, L_2, and L_3 for the three courses respectively, Gamsey could cook the three courses, one after the other, by taking $|L_1| + |L_2| + |L_3|$ trips to the pantry. However, if the lists have ingredients in common, it will be possible to take fewer trips to the pantry by cooking the courses in parallel, using ingredients in more than one course at a time. Describe an $O(|L_1||L_2||L_3|)$-time algorithm to determine the minimum number of pantry trips that Gamsey must take to make the three-course meal.

Solution:

1. **Subproblems**
   - Let $L[i]$ denote i^{th} ingredient in ingredient list L (1 indexed, where $L[0]$ is NULL)
   - $x(i_1, i_2, i_3)$: minimum number of trips to pantry to use ordered prefix of ingredients $(L_j[1], \ldots, L_j[i_j])$ from L_j for $j \in \{1, 2, 3\}$
2. **Relate**
   - Guess ingredient retrieved in last trip to the pantry
   - Last trip will return either ingredient $L_1[i_1]$, $L_2[i_2]$, or $L_3[i_3]$
   - Let $f(a, b) = 1$ if $a = b$ and 0 otherwise (computable in $O(1)$ time by assumption)
   - $x(i_1, i_2, i_3) = 1 + \min \left\{ \begin{array}{ll} x(i_1 - 1, i_2 - f(L_1[i_1], L_2[i_2]), i_3 - f(L_1[i_1], L_3[i_3])) & \text{if } i_1 > 0 \\ x(i_1 - f(L_2[i_2], L_1[i_1]), i_2 - 1, i_3 - f(L_2[i_2], L_3[i_3])) & \text{if } i_2 > 0 \\ x(i_1 - f(L_3[i_3], L_1[i_1]), i_2 - f(L_3[i_3], L_2[i_2]), i_3 - 1) & \text{if } i_3 > 0 \end{array} \right\}$
   - Subproblem $x(i_1, i_2, i_3)$ only depends on strictly smaller $i_1 + i_2 + i_3$, so acyclic
3. **Base**
   - $x(0, 0, 0) = 0$ (no more trips needed if all ingredient lists have been finished)
4. **Solution**
   - $x(|L_1|, |L_2|, |L_3|)$ is fewest pantry trips to use all ingredients in all recipes, as desired
5. **Time**
   - # subproblems: $\leq (|L_1| + 1)(|L_2| + 1)(|L_3| + 1) = \Theta(|L_1||L_2||L_3|)$
   - work per subproblem: $O(1)$ time
   - $O(|L_1||L_2||L_3|)$ total running time, which is polynomial in $|L_1|$, $|L_2|$ and $|L_3|$

Common Mistakes:

- Missing some case in the relation
- Not correctly handling cases where a list index corresponds to an empty list

[1] An ingredient may appear multiple times on the list if it is used at different times in the recipe.
[2] After using an ingredient, leftovers must be returned to the pantry before a new ingredient may be used.

SCRATCH PAPER 1. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S1" on the problem statement's page.

SCRATCH PAPER 2. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S2" on the problem statement's page.