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: Number Scrabble	1	17
3: Limited-Unlimited	1	19
4: Office Hour Optimization	1	19
Total		60

Name: ______________________________

School Email: ______________________________

Problem 1. [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 (3), (4), and (5) 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 2. [3 points] **Pseudo-Circling**

Indicate whether the given running times of each of problems (3), (4), and (5) are polynomial or pseudopolynomial by circling the appropriate word below. **One can answer this question without actually solving problems (3), (4), and (5).** (1 point each)

(a) Problem 3: Number Scrabble Polynomial Pseudopolynomial

Solution: Polynomial

(b) Problem 4: Limited-Unlimited Polynomial Pseudopolynomial

Solution: Pseudopolynomial

(c) Problem 5: Office Hour Optimization Polynomial Pseudopolynomial

Solution: Polynomial

Problem 3. [17 points] **Number Scrabble**

Number Scrabble is a one-player game played on an array $T = [t_0, \dots, t_{n-1}]$ of n positive integers. There is a list $P = \{(p_0, v(p_0)), \dots, (p_{m-1}, v(p_{m-1}))\}$ of m unique **playable words**, where playable word p_i is a non-empty array of at most 10 positive integers and $v(p_i)$ is the positive integer value of p_i. The objective of the game is to find a **gameplay** S — a list of **non-overlapping** subarrays (i.e., substrings) of T, each a playable word — where S has maximum total value, $\sum_{s \in S} v(s)$. For example, if

$$T = [1, 5, 2, 4, 1] \text{ and } P = \{([2], 3), ([1], 1), ([5, 2], 8), ([1, 2], 12), ([1, 5], 2)\},$$

then $S_1 = ([1, 5], [2], [1])$, $S_2 = ([1], [5, 2], [1])$, and $S_3 = ([1], [2], [1])$ are all valid gameplays, with total values 6, 10, and 5 respectively. Note playable word $[1, 2]$ cannot exist in any gameplay of T, since $[1, 2]$ is not a contiguous subarray of T. Given T and P, describe an $O(n + m)$-time algorithm to return a gameplay of maximum total value.

Solution: To solve this problem, it would be useful to be able to check whether a particular array of at most 10 positive integers is a playable word. Construct an empty hash table D and insert each p_i for $i \in \{0, \dots, m-1\}$ into D, mapping to its value $v(p_i)$. Each hash table insertion takes expected constant time (as each p_i has constant size), so constructing D takes expected $O(m)$ time. Now we solve the problem via dynamic programming.

1. **Subproblems**
   - $x(i)$: the maximum total value of any gameplay on suffix $T[i:]$ for $i \in \{0, \dots, n\}$
2. **Relate**
   - Left-most playable word either starts with t_i or it does not
   - If playable word starts with t_i, word may have any length in $\{1, \dots, 10\}$ (Guess!)
   - $x(i) = \max \{x(i+1)\} \cup \left\{ D[T[i:i+j]] + x(i+j) \;\middle|\; \begin{array}{ll} j \in \{0, \dots, 10\} & \text{and} \\ i + j \le n & \text{and} \\ T[i:i+j] \in D & \end{array} \right\}$
3. **Topo**
   - $x(i)$ only depends on subproblems with strictly larger i, so acyclic
4. **Base**
   - $x(n) = 0$ (empty gameplay admits no value)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - $x(0)$ is the maximum value of any gameplay on T
   - Store parent pointers to reconstruct an optimal gameplay
6. **Time**
   - # subproblems: $n + 1 = O(n)$
   - Work per subproblem: expected $O(1)$
   - Together with hash table construction, yields expected $O(n + m)$ time
   - (See scratch S2 for common mistakes)

Problem 4. [19 points] **Limited-Unlimited**

Given two sets of integers A and B, a **limited-unlimited sequence** of A and B is any sequence S of integers such that each integer $s \in S$ appears in either A or B, and if s appears in A then s appears at most once in S. Given a target sum m and two disjoint sets A and B, each containing exactly n **distinct positive** integers, describe an $O(nm)$-time algorithm to determine whether m is the sum of any limited-unlimited sequence S of A and B, i.e., $m = \sum_{s \in S} s$.

Solution:

1. **Subproblems**
   - Fix an arbitrary order on $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{n-1})$
   - $x_A(i, k)$: Boolean whether k is sum of any subset of suffix of $A[i:]$ (without repeats)
   - $x_B(i, k)$: Boolean whether k is sum of any subset of suffix of $B[i:]$ (allowing repeats)
   - for $i \in \{0, \ldots, n\}, k \in \{0, \ldots m\}$
2. **Relate**
   - Either use a_i once or not (cannot use again)
   - $x_A(i,k) = \text{OR}\left\{\begin{array}{ll} x_A(i+1, k-a_i) & \text{if } a_i \le k \\ x_A(i+1,k) & \text{always} \end{array}\right\}$
   - Either use b_j once or not (but may use again)
   - $x_B(i,k) = \text{OR}\left\{\begin{array}{ll} x_B(i, k-b_j) & \text{if } b_j \le k \\ x_B(i+1,k) & \text{always} \end{array}\right\}$
3. **Topo**
   - Subproblems $x_A(i,k)$ and $x_B(i,k)$ each depend only on subproblems with strictly smaller $k - i$, so acyclic
4. **Base**
   - $x_s(i, 0) = \text{True}$ for $s \in \{A, B\}, i \in \{0, \ldots, n\}$ (can always make zero sum)
   - $x_s(n, k) = \text{False}$ for $s \in \{A, B\}, k \in \{1, \ldots, m\}$ (cannot make positive sum)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - Original is whether a subset of A and a possibly-repeating subset of B sum to m
   - i.e., $\text{OR}\{\text{AND}\{x_A(0,k), x_B(0, m-k)\} \mid k \in \{0, \ldots, m\}\}$
6. **Time**
   - # subproblems: $(n+1)(m+1) = O(nm)$
   - Work per subproblem: $O(1)$
   - Original takes $O(m)$ time
   - $O(nm)$ running time in total
   - (See scratch S2 for common mistakes)

Problem 5. [19 points] **Office Hour Optimization**

Class 0.660 (Algorithms for Introductions) is holding online office hours to help students on three problems — a, b, and c — in three corresponding breakout rooms. The TAs want to develop a 'Sort Bot' to effectively assign each student to a single room at the start of office hours. Assume there are $3n$ students, where each student i has known nonnegative integer **benefit** a_i, b_i, and c_i for being assigned to the room for problem a, b, and c, respectively.

Describe an $O(n^3)$-time algorithm to determine whether it is possible to assign the students **equally** to the three breakout rooms (i.e., n students to each room) while providing **strictly positive** help to every student, and if possible, return the maximum total benefit to students of any such assignment. Note that the assignment must not assign a student to a room for which they would get zero benefit.

Solution:

1. **Subproblems**
   - Let the students be $i \in \{1, \ldots, 3n\}$
   - $x(i, j, k)$: the maximum benefit assigning students $\{1, \ldots, i+j+k\}$ to breakout rooms, with i students to breakout a, j students to breakout b, and k students to breakout c, where each student is assigned to a breakout room with strictly positive benefit (or equals $-\infty$ if no such assignment is possible)
   - for $i, j, k \in \{0, \ldots, n\}$
2. **Relate**
   - Must assign student $i+j+k$ to some room (Guess!)
   - $x(i,j,k) = \max\{-\infty\} \cup \left\{ \begin{array}{ll} a_{i+j+k} + x(i-1,j,k) & \text{if } i > 0 \text{ and } a_{i+j+k} > 0, \\ b_{i+j+k} + x(i,j-1,k) & \text{if } j > 0 \text{ and } b_{i+j+k} > 0, \\ c_{i+j+k} + x(i,j,k-1) & \text{if } k > 0 \text{ and } c_{i+j+k} > 0 \end{array} \right\}$
3. **Topo**
   - Subproblem $x(i, j, k)$ depends only on strictly smaller $i + j + k$, so acyclic
4. **Base**
   - $x(0, 0, 0) = 0$ (no benefit to assigning zero students)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - $x(n, n, n)$ is the maximum benefit to assign all students evenly to rooms
6. **Time**
   - # subproblems: $(n + 1)^3 = O(n^3)$
   - Work per subproblem: $O(1)$
   - $O(n^3)$ running time in total
   - (See scratch S2 for common mistakes)

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.

Common Mistakes: (Problem 3: Number Scrabble)

- Not checking whether substrings of T are playable
- A relation that assumes a gameplay must include every number in T
- Not accounting for time to read P in running time
- Not using parent pointers to return a gameplay

Common Mistakes: (Problem 4: Limited-Unlimited)

- Not allowing integers in B to be used more than once
- Trying to store or maintain an arbitrary subset (can have exponential state)

Common Mistakes: (Problem 5: Office Hour Optimization)

- Poor communication of subproblems (not specifying suffix/prefix, etc.)
- Defining $O(n^4)$ subproblems without arguing that only $O(n^3)$ are needed
- Allowing more than n students in a room, or assigning students to rooms with no benefit
- Pseudopolynomially many subproblems with Boolean output (must be polynomial in n)
- Taking the maximum over an OR

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.