CS141, Winter 2025, Assignment #4 **Due: 11:59 PM, 3/4/2025**
Name: **Net ID:**

Deadline. The homework is due 11:59 PM, 3/4/2025. You must submit your solutions (in pdf format generated by LaTeX) via GradeScope. Canvas will not be accepted.
Assignment Policy.

- Unless mentioned otherwise, late submission allowed for 20% penalty for each calendar day.
- Assignments should be in pdf format generated by LaTeX
- If you are using any external source, you must cite it and clarify what exactly got out of it.
- You are expected to understand any source you use and solve problems in your own.
- All pages relevant to a question must be assigned on Gradescope. Unassigned pages will not be graded.
- Unless stated in the question, your designed algorithm should use the most optimized algorithm you have learned so far. A too-complex algorithm may result in penalties.

1 0/1 Knapsack (20 points)

Part A (10 Points):
You are given n items, each with a weight w_i and a value v_i. You also have a knapsack that can carry at most W units of weight. The goal is to determine the maximum total value of items that can be placed in the knapsack without exceeding its weight limit. You can only include up to 1 of each item at a time.
To solve this problem using dynamic programming, we define a table $DP[i][j]$.
Your Tasks:

1. Write the recurrence for the DP table. Use `\begin{cases}` to write your solution e.g.:

$$DP[i][j] = \begin{cases} xxx & \text{if condition} \\ yyy & \text{if other condition} \end{cases}$$

2. What does the index $DP[i][j]$ represent?
3. What are the dimensions of the DP table?
4. Write the pseudocode for the bottom-up DP algorithm.

Part B (10 Points):
Now, we're going to solve a slightly different problem. Rather than only being able to have 0 or 1 of each item, we can now have 0, 1, or 2. Once again, we want to maximize the total value of items in the knapsack without exceeding its weight limit.

1. Write the recurrence for the DP table. Use `\begin{cases}` to write your solution e.g.:

$$DP[i][j] = \begin{cases} xxx & \text{if condition} \\ yyy & \text{if other condition} \end{cases}$$

2. What does the index $DP[i][j]$ represent?
3. What are the dimensions of the DP table?
4. Write the pseudocode for the bottom-up DP algorithm.

2 Modified Edit Distance (20 points)

The edit distance between two strings A and B of lengths n and m is the minimum number of operations required to transform A into B. The allowed operations are:

- **Insertion:** Insert a character into A (cost: 1 unit).
- **Deletion:** Remove a character from A (cost: 1 unit).
- **Modification:** Change a character in A to a different character (cost: 1 unit).

Let $DP[i][j]$ represent the minimum cost to transform the first i characters of A into the first j characters of B.

Part A (10 Points):

1) Write the recurrence relation for $DP[i][j]$ assuming that **modifications are not allowed**, meaning you can only insert or delete characters. Use `\begin{cases}` to structure your solution. 2) What is the computational complexity of this algorithm? Justify your answer.

Part B (10 Points):

Now assume that each operation has an associated cost:

- Inserting a character into A has cost C_I.
- Deleting a character from A has cost C_D.
- Modifying a character in A has cost C_M.

1) Write the recurrence relation for $DP[i][j]$ in this weighted version of the problem. 2) What is the computational complexity of this algorithm? Justify your answer.

3 Tipping the Scales (20 points)

Assume you gave n balances and n stones. Each balance requires a minimum weight, b_i to tip and each stone weighs at most w_i. You can place only one stone on each balance.

1. Describe a **greedy** algorithm that finds the maximum number of balance that can be tipped.
2. Describe the greedy choice property. Prove that this property holds.
3. Describe the optimal substructure property. Prove that this property holds.

4 High Throughput (20 points)

Xianghu and Michael's server hosting company is taking off, and they finally have enough money to buy some of their own hardware. Then now own a single server that runs a single task at a time. There are n tasks in queue that need processing. The runtime of each task is known in advance: t_i for task i. The goal is to minimize the total weighting time:

$$T = \sum_{i=1}^{n} \text{Time task } i \text{ waits in queue}$$

For example, if the tasks are completed in order of increasing i, the i-th task would sit in queue for $\sum_{j=1}^{i} t_j$ minutes. Therefore, the total waiting time could be expresesd as $\sum_{i=1}^{n} \sum_{j=1}^{i} t_j = \sum_{i=1}^{n} t_i$.

Task 1: Provide a greedy algorithm that runs in $O(n \log n)$ time and minimizes the total queue time.
Task 2: Prove correctness by showing the greedy choice and optimal substructure properties.

5 Task Scheduling (20 points)

Suppose you are given a set $T = (s_1, f_1), (s_2, f_2), ..., (s_n, f_n)$ of n tasks, where s_i and f_i are the start and finish times of task i, respectively. Two tasks are *non-conflicting* if $f_i \leq s_j$ or $f_j \leq s_i$.
The goal is to schedule the larges number of tasks such that no two tasks conflict. The greedy algorithm discussed in class considers tasks one-by-one ordered by increasing finish time. Consider the following alternate approach and prove/disprove it's optimality:
Algorithm:

1. Compute the number of overlaps for each task
2. Sort the tasks in increasing number of overlaps. Ties are broken arbitrarily.
3. Pick task i with the smallest number of overlaps, schedule it, and remove if from further consideration. Also, remove any tasks that conflict with task i.
4. Repeat step 3 until no tasks remain.

Note: The number of overlaps is **NOT** updated after step 1.
If you think this strategy works, prove that the greedy choice property holds. If not, give an example where this strategy fails. You may provide a computer-generated visual example (e.g., create one in powerpoint and export it as a PNG, then include it in the document). Hand-drawn solutions will not be accepted.