- Please submit before the submission deadline.
- Assignment submitted one (1) day after the assignment deadline will be accepted with 20% deduction on corresponding assignment grade.
- Assignment submitted more than one (1) day late will not be accepted.

Problem 1: Phone Cable Cutter (25 pts)

You are a **construction worker for a cable company** tasked with cutting a long **phone cable** into smaller segments for various installations. You are informed by your company that The cost of each cut is proportional to the length of the remaining cable, and your goal is to minimize the total cutting cost while ensuring the cable is cut at specific positions.

You are given a **phone cable** of length $L > 0$ and need to make cuts at n specific positions given by p_i. The positions are given as $0 < p_1 < p_2 < \cdots < p_n < L$, where positions 0 and L represent the left and right ends of the cable, respectively. The cost of each cut is proportional to the length of the remaining cable at the time of cutting. Assume that you can only make one cut at a time.

Develop an algorithm with a time complexity of $O(n^3)$ using dynamic programming to find the minimal total cost for making all the cuts at the given positions.

(a) (5 points) Define the subproblem.

Let $C(i, j)$ be the minimal cost of cutting positions $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$ on a stick piece starting at p_i and ending at p_j. Here, we define $p_0 := 0$ and $p_{n+1} := L$ to extend the input domain of $C(i, j)$ to $0 \leq i < j \leq n + 1$.

Grading Criteria:
(+5 points) correct answer and process.
(−2 points) Subproblem doesn't cover the original problem.

(b) (5 points) State the recurrence relation and explain how you derived it.

$$C(i,j) = \begin{cases} 0, & \text{if } i = j-1, \\ (p_j - p_i) + \min_{i<k<j} \left(C(i,k) + C(k,j) \right), & \text{if } i < j-1. \end{cases}$$

The minimal cost of cutting the positions $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$ for a stick with length $p_j - p_i$ is the minimum of all possible cutting cases. This is given by the length of the stick $(p_j - p_i)$ plus the minimal cost of the left part $C(i, k)$ and the minimal cost of the right part $C(k, j)$, where the cutting indices k are between $i + 1$ and $j - 1$, i.e., $i < k < j$.

Grading Criteria:
(+2 points) correct explanation.
(+3 points) correct recurrence relation.

(c) (5 points) Explain the base case.
Base case is when i=j-1. If $i = j - 1$, the stick doesn't need to be cut, so in this case, $C(i, j) = 0$.

Grading Criteria:
(+2 points) correct explanation.
(+3 points) correct base case.

(d) (5 points) Explain the algorithm. You may use pseudocode if needed.

Algorithm 1 CuttingCable(p, L)

$p(0) = 0$
$p(n + 1) = L$
for $i = 0$ **to** n **do**
    $C(i, i + 1) = 0$
end for
for $s = 2$ **to** $n + 1$ **do**
    for $i = 0$ **to** $n + 1 - s$ **do**
        $j = i + s$
        Set $C(i, j)$ as mathematical maximum (Infinity)
        for $k = i + 1$ **to** $j - 1$ **do**
            $C(i, j) = \min(C(i, j), C(i, k) + C(k, j))$
        end for
    end for
end forreturn $C(0, n + 1)$

Grading Criteria:
(+5 points) correct answer that solves the CuttingCable problem.
(−1 points) Returns incorrect answer.
(−2 points) Insufficient Explanation.

(e) (5 points) Analyze the worst-case time complexity using Big-O notation.
The algorithm contains three nested loops:

- The first loop iterates n times.
- The second loop iterates n times (since i ranges from 2 to $n + 1$)
- The third loop iterates at most n times, as $j - 1 - (i + 1) + 1 < n$.

Thus, the total time complexity is $O(n) \cdot O(n) \cdot O(n) = O(n^3)$. **Grading Criteria:**
(+2 points) correct explanation.
(+3 points) correct time complexity.

Problem 2: Understanding NP (20 points)

Given decision problems X and Y and any problem Z, answer the following problems with true or false along with a brief justification. Your explanation may only be 1-2 short sentences.

(a) (5 points) If X is NP-complete, Y is in NP, and $X \leq_p Y$, then Y is NP-complete.
True. Every problem in NP can be reduced to X in polynomial time, and X can be reduced to Y in polynomial time. Thus, every problem in NP can be reduced to Y in polynomial time, i.e. Y is NP-complete.

(b) (5 points) If Y is NP-complete and $X \leq_p Y$, then X is NP-complete.
Depends on the $P = NP$.
If $P = NP$, then True. Since $P = NP = NP$-complete, and X is a decision problem (i.e., in NP).
If $P \neq NP$, then False. If X is in P, then not every problem in NP can be reduced to X.

(c) (5 points) If $P \neq NP$ and Z is in NP, then Z cannot be solved in polynomial time.
False. Even if Z is in NP and P $\neq$ NP, Z can still be solvable in polynomial time since all P problems are NP.

(d) (5 points) If $P \neq NP$ and Z is in P, then Z is NP-complete.
False. If Z is NP-complete under the assumptions, then all problems in NP are reducible to Z in polynomial time, thus will be solvable in polynomial time, i.e. P=NP, which is a contradiction of the assumption.

Grading Criteria:
For all subproblems in problems 2:
(+5 points) correct answer and justification.
(−3 points) no justification.
(no deduction) (b) gets a correct point as long as it's apparent which assumption of $P = NP$ was taken, and the subsequent explanation justifies the answer based on the assumption.

Problem 3: Knapsack Problem(55 points)

Recall the `0-1 Knapsack` problem from the lecture. The `0-1 Knapsack` problem is as follows: We have a set of n items each with a weight w_i and cost c_i for i=1, 2..., n. Given a maximum weight W, the problem is to find a set of variables $x_1, x_2, \ldots, x_n$ that maximizes

$$\sum_{i=1}^{n} x_i c_i$$

while satisfying the following conditions where $x_i \in \{0, 1\}$ for all i:

$$\sum_{i=1}^{n} x_i w_i \leq W$$

Now, consider a `0-1 Knapsack` scenario where W is 6, and we have $n = 4$ items given in the table below:

Item	Weight	Cost
1	2	3
2	3	4
3	4	5
4	5	7

(a) (5 points) Calculate the total cost and the selection items for that cost based on the greedy algorithm from the lecture.

Total cost is 7, when selecting item 2 and 1.
Grading Criteria:
(+3 points) Correct cost.
(+2 points) Correct selection of items for the correct cost.

(b) (20 points) Propose an algorithm using Dynamic Programming for the `0-1 Knapsack` problem. You must (1) define the subproblem, (2) state the recurrence relation, (3) explain the base case, then (4) write the algorithm (You may use pseudocode). You don't have to prove its time complexity.

(1) Let $dp[i][w]$ represent the maximum cost achievable by selecting from the first i items with a knapsack capacity of w.
(2) The recurrence relation is defined as follows:

$$dp[i][w] = \begin{cases} dp[i-1][w] & \text{if } w_i > w \\ \max(dp[i-1][w], dp[i-1][w-w_i] + c_i) & \text{if } w_i \leq w \end{cases}$$

Where:

- $dp[i-1][w]$ is the solution without including item i,
- $dp[i-1][w-w_i] + c_i$ is the solution if item i is included.

(3) For the base case, when there are no items or the knapsack has zero capacity, we have:

$$dp[0][w] = 0 \quad \text{for all } w$$

and

$$dp[i][0] = 0 \quad \text{for all } i.$$

(4) The algorithm to solve the problem is as follows:

Algorithm 2 0-1 Knapsack Problem (Dynamic Programming)

Input: Weights array $w[1 \dots n]$, Costs array $c[1 \dots n]$, Knapsack capacity W
Output: Maximum total cost.
Initialize a 2D array $dp[0 \dots n][0 \dots W]$ with 0
for each $i = 1$ to n **do**
  for each $w = 1$ to W **do**
    if weight of item i, $w_i > w$ **then**
      $dp[i][w] = dp[i-1][w]$ ▷ Item i cannot be included
    else
      $dp[i][w] = \max(dp[i-1][w], dp[i-1][w-w_i] + c_i)$ ▷ Max of excluding or including item i
    end if
  end for
end for
max_cost $= dp[n][W]$
Backtrack to find selected items:
$w = W$
Initialize an empty list selected_items
for each $i = n$ down to 1 **do**
  if $dp[i][w] \neq dp[i-1][w]$ **then** ▷ Item i is included
    Add i to selected_items
    $w = w - w_i$ ▷ Reduce the remaining weight
  end if
end for
Output: max_cost]

Grading Criteria:
(+5 points) Correct subproblem definition.
(+5 points) Correct recurrence relation.
(+5 points) Correct base case explanation.
(+5 points) Correct pseudocode.

(c) (5 points) Calculate the total cost and the selection items for that cost based on the Dyanamic Programming algorithm you presented above.

Total cost is 8, when selecting item 1 and 3.

Grading Criteria:
(+3 points) Correct cost.
(+2 points) Correct selection of items for the correct cost.

Now consider the `0-1 Knapsack-Decision` problem, which is as follows: We have a set of n items $x_1, x_2, \ldots, x_n$, each with a weight w_i and cost c_i. Given a maximum weight W and a budget C, the problem is to decide whether a set of variables $x_1, x_2, \ldots, x_n$ satisfies the following conditions:

$$\sum_{i=1}^{n} x_i c_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} x_i w_i \leq W$$

where $x_i \in \{0, 1\}$ for all i.

(d) (10 points) Show that `0-1 Knapsack-Decision` belongs to NP.
Given n items, it takes linear time to compute

$$\sum_{i=1}^{n} x_i c_i \quad \text{and} \quad \sum_{i=1}^{n} x_i w_i,$$

to check if the total weight is less than W and the total cost is greater than the budget C.

Grading Criteria:
(+5 points) Indicate what the verifier for the problem would be.
(+5 points) Describes how the verifier is computed in polynomial time.

(e) (15 points) Show that `0-1 Knapsack-Decision` is NP-complete by reduction from `SUBSET-SUM`.

Construct an instance of `SUBSET-SUM` by considering the following special case: it consists of n non-negative integers $y_1, y_2, \ldots, y_n$ and a target sum S.

$$y_i = w_i = c_i \quad \text{for all } i = 1, 2, \ldots, n,$$

and let $W = S = C$. The Yes/No answer to this problem corresponds to the same answer to the original problem.

Thus, we can construct a polynomial reduction from SUBSET-SUM. Since SUBSET-SUM is NP-complete, all NP problems are reducible to SUBSET-SUM in polynomial time, and also reducible to the `0-1 Knapsack-Decision` problem, which is NP.
In other words, the `0-1 Knapsack-Decision` problem is NP-Complete.

Grading Criteria:
(+10 points) Show polynomial reduction from SUBSET-SUM
(+5 points) Describes how the problem is both NP and all NP problems can be reduced to it.