Practice Quiz 2

Problem 1 Halloween Candies

Tanmoy remembers how all of his friends ended up with more candy than him every Halloween. By some miracle, he is able to transfer all his memories to his 9-year-old self on Halloween night. Equipped with the memories of future Tanmoy and the power of designing and analyzing algorithms, 9-year-old Tanmoy decides to come up with an algorithm to maximize the amount of candy obtained.

He labels the houses as 1...n and from the memories of present-day Tanmoy, he labels the amount of candy they should have as c_i. Thanks to future Tanmoy's social psychology classes, kid Tanmoy decides to not visit any houses that are adjacent to each other. For example, if Tanmoy visits house 1 he can not visit house 2, if he visits house 3 he can not visit house 2 or 4, if he visits house n-1 he can not visit house n.

Tanmoy wants to construct a table T such that the last entry of the table T(n) contains the max total number of candies he can get. There may be many solutions to this problem, but only one of the recurrences on the list below achieves a working algorithm.

Select the correct recurrence relation. You do not need to explain your answer.

Example:
Input: A=[5,7,8,10,9,4,2,11,6]
Output: 33
Tanmoy can go to house 1 for 5 candies, 3 for 8 candies 5 for 9 candies, and 8 for 11 candies.

Answer T(i) = $\max(T(i-1), T(i-2)+c_i))$

Problem 2 True and False

For 0-1 Knapsack, the runtime is O(nB) where B is the capacity and n is the number of objects
True

The base case for the change making problem's Knapsack solution involves filling the initial row and column with all 1s
False

The time complexity of running chain matrix multiplication is O(n^3), where n is the number of matrices.
True

The worst case time complexity for Bellman-ford is O(|V||E|).
True

Problem 3 Coin Change

(a) Define the entries of your table in words. E.g., $T(i)$ or $T(i, j)$ is ....

Let $T(i, j)$ be the proposition that it is possible to make change for some change j using the first i limited denominations.

(b) State recurrence for entries of table in terms of smaller subproblems.

Recurrence relation:

$$T(i, j) = \bigvee_{0 \leq k \leq F[i]} \{T(i-1, j - k \cdot D[i]) \mid k \cdot D[i] \leq j\}$$

Base case(s): $T(i, 0) =$ `true` for $0 \leq i \leq n$ and $T(0, j) =$ `false` for $1 \leq j \leq V$

(c) Write pseudocode for your algorithm to solve this problem.

```
1: function CoinChange(D[1, ..., n], F[1, ..., n], V)
2:     Initialize T as a 2-D array of dimensions (V + 1) · (V + 1)
3:     for i = 0 → n do
4:         T(i, 0) = true
5:     for j = 1 → V do
6:         T(0, j) = false
7:     for i = 1 → n do
8:         for j = 1 → V do
9:             for k = 0 → F[i] do
10:                if k · D[i] ≤ j then
11:                    T(i, j) = T(i, j) ∨ T(i − 1, j − k · D[i])
12: return T(n, V)
```

(d) Analyze the running time of your algorithm.

We have a triple nested for loop where i iterates through n integer values in the range $[1, n]$, j iterates through V integer values in the range $[1, S]$, and k iterates through $F[i] + 1$ integer values in the range $[0, F[i]]$. Therefore, the running time of our algorithm is $\mathcal{O}(n \cdot V \cdot max_i F[i])$

Problem 4 Welding Rods

(a) Define the entries of your table in words. E.g., $T(i)$ or $T(i, j)$ is ....

$T(s)$ is boolean valued and denotes whether we can create a rod of length s using a subset of the rods of lengths $l_1, \ldots, l_n$.

(b) State recurrence for entries of table in terms of smaller subproblems.

Recurrence relation:

$$T(s) = \bigvee_{1 \le j \le n} \{T(i - l_j) : l_j \le s\}$$

Base case(s): $T(0) = \texttt{true}$

A rod of length s can be made by welding together a rod of length $s - l_j$ with a rod of length l_j for some j. Thus, we check if it is possible to make a rod of length $s - l_j$. This is just $T(s - l_j)$ in our table.

(c) Write pseudocode for your algorithm to solve this problem.

```
1: function InfinitelyManyRods(S, n, (l_1, ..., l_n))
2:     Initialize T as an array of size S + 1
3:     T[0] = true
4:     for s = 1, ..., S do
5:         T[s] = false
6:         for j = 1, ..., n do
7:             if l_j ≤ s & T[s − l_j] then
8:                 T[s] = true
       return T[S]
```

(d) Analyze the running time of your algorithm.

Since we fill a table of length $O(S)$ and our transitions take $O(n)$ time, the time complexity of our algorithm is $O(nS)$.