Question 1

Nous avons vu en classe un algorithme vorace pour résoudre le problème du retour de la monnaie en un nombre minimum de pièces. Nous avons observé que cet algorithme fonctionne correctement avec de pièces de monnaie canadiennes mais celui-ci ne trouve pas toujours la bonne réponse si, par exemple, des pièces de 12 cents sont ajoutées (e.g. il y a une erreur pour retourner 15 cents). Le problème général du retour de la monnaie peut être résolu exactement en utilisant la programmation dynamique.

Soit n le nombre de pièces distinctes et soit $T[1 \dots n]$ un tableau donnant la valeur de ces pièces (il n'y a aucun intérêt à trier ce tableau). Supposons une quantité illimitée de chaque type de pièces. Soit L, une limite sur le montant à obtenir.

a) Pour $1 \leq i \leq n$ et $1 \leq j \leq L$, soit $c_{i,j}$ le nombre minimum de pièces pour obtenir le montant j si on se limite aux pièces de type $T[1], T[2], \dots, T[i]$. Si ce montant ne peut être obtenu alors $c_{i,j} = +\infty$. Donnez une équation de récurrence pour $c_{i,j}$ incluant les conditions initiales.

$$c_{i,j} \begin{cases} 1 + c_{i,j}(L - T[i]) & \text{si } L \geq 0 \\ +\infty & \text{ailleurs} \end{cases}$$

b) Donnez un algorithme de programmation dynamique pour calculer tous les $c_{n,j}$ où $1 \leq j \leq L$. Votre algorithme ne doit utiliser qu'un seul tableau (à une dimension) de longueur L.

Algorithme 1 : Trouver $c_{n,j}$

```
   Données : T[1...n], entier n, entier L
   Résultat : Entier c[L]
 1 si L = 0 alors
 2 |  retourner (INT_MAX)
 3 fin
 4 pour j ← 1 à L faire
 5 |  c[j] ← INT_MAX
 6 fin
 7 pour j ← 1 à L faire
 8 |  pour i ← 1 à n faire
 9 |  |  si T[i] ≤ j alors
10 |  |  |  si (j − T[i]) = 0 alors
11 |  |  |  |  temp ← 0
12 |  |  |  fin
13 |  |  |  sinon
14 |  |  |  |  temp ← c[j − T[i]]
15 |  |  |  fin
16 |  |  |  si temp ≠ INT_MAX & (temp + 1) < c[j] alors
17 |  |  |  |  c[j] ← temp + 1
18 |  |  |  fin
19 |  |  fin
20 |  fin
21 fin
```

c) Analysez le temps d'exécution de votre algorithme en fonction de n et de L.

Table 1: Analyse du temps d'exécution de l'algorithme

Ligne	Temps/Exécution	Nombre d'exécutions	Ordre
1	c_1	1	$O(1)$
2	c_2	1	$O(1)$
3	c_3	1	$O(1)$
4	c_4	$[L]+1$	$O(L)$
5	c_5	$[L]$	$O(L)$
6	c_6	1	$O(1)$
7	c_7	$[L]+1$	$O(L)$
8	c_8	$[L]\big([n]+1\big)$	$O(Ln)$
9	c_9	$[L][n]$	$O(Ln)$
10	c_10	$[L][n]$	$O(Ln)$
11	c_11	$[L][n]$	$O(Ln)$
12	c_{12}	$[L][n]$	$O(Ln)$
13	c_{13}	$[L][n]$	$O(Ln)$
14	c_{14}	$[L][n]$	$O(Ln)$
15	c_{15}	$[L][n]$	$O(Ln)$
16	c_{16}	$[L][n]$	$O(Ln)$
17	c_{17}	$[L][n]$	$O(Ln)$
18	c_{18}	$[L][n]$	$O(Ln)$
19	c_{19}	$[L][n]$	$O(Ln)$
20	c_{20}	$[L][n]$	$O(Ln)$
21	c_{21}	$[L]$	$O(L)$

À l'aide du tableau ci-dessus, nous pouvons conclure que le temps d'exécution de notre algorithme est de l'ordre de $O(Ln)$.