

Amérique du Nord – 2025 – sujet1 - Correction

Exercice 2 (6 points)

1. Méthode passer_transit

```
class Colis:
    def passer_transit(self):
        self.etat = 'transit'
```

👉 *Commentaire : la méthode modifie simplement l'attribut etat de l'objet.*

2. Fonction ajouter_colis avec contrainte de poids

```
def ajouter_colis(liste, colis):
    if colis.poids <= 25:
        liste.append(colis)
    else:
        print("Dépassement du poids maximal autorisé")
```

👉 *Commentaire : on teste le poids avant l'ajout à la liste.*

3. Fonction nb_colis

```
def nb_colis(liste):
    return len(liste)
```

👉 *Commentaire : la longueur de la liste correspond au nombre de colis.*

4. Fonction poids_total

```
def poids_total(liste):
    total = 0                                # ligne 2
    for c in liste :
        total = total + c.poids             # ligne 4
    return total
```

👉 *Commentaire : on additionne les poids de tous les colis de la liste.*

5. Fonction liste_colis_etat

```
def liste_colis_etat(liste, statut):
    resultat = []
    for colis in liste:
        if colis.etat == statut:
            resultat.append(colis)
    return resultat
```

👉 *Commentaire : on construit une nouvelle liste contenant uniquement les colis ayant l'état demandé.*

6. Tri décroissant et complexité

Le tri utilisé est un **tri par sélection** : à chaque étape, on cherche le colis le plus lourd parmi ceux qui restent et on le place à la position i .

Son coût dans le pire des cas est **quadratique** : $O(n^2)$.

7. Autre algorithme

Un autre algorithme possible est le **tri fusion**, de coût $O(n \log n)$ dans le pire des cas.

8. Fonction récursive chargement_glouton

```
def chargement_glouton(liste, rang, capacite):
    if rang == len(liste):
        return []
    elif liste[rang].poids <= capacite:
        return [liste[rang]] + chargement_glouton(liste, rang + 1,
                                                    capacite - liste[rang].poids)
    else:
        return chargement_glouton(liste, rang + 1, capacite)
```

👉 *Commentaire : On examine les colis du plus lourd au plus léger : un colis qui tient dans la place restante est chargé, et la capacité restante diminue de son poids ; sinon on passe au suivant.*

9. Origine possible de l'erreur RecursionError

Chaque appel de `chargement_glouton` fait un nouvel appel récursif avec `rang + 1` : pour une liste de n colis, il y a $n + 1$ appels imbriqués, tous en attente sur la pile d'appels.

Si la liste contient beaucoup de colis (plus de 1000 environ, la limite par défaut de Python), **la pile d'appels déborde** et Python lève `RecursionError`.

👉 *Commentaire : Ce n'est pas une erreur de programmation : le cas de base est bien atteint, mais trop tard. D'où l'intérêt de la version itérative de la question 10.*

10. Version itérative chargement_glouton2

```
def chargement_glouton2(liste, capacite):
    colis_a_charger = []
    charge = capacite  # place restante dans le camion
    for colis in liste:
        if colis.poids <= charge:
            colis_a_charger.append(colis)
            charge = charge - colis.poids
    return colis_a_charger
```

👉 *Commentaire : cette version évite la récursivité et applique directement l'algorithme glouton à l'aide d'une boucle for. Remarque : contrairement à ce que laisse entendre l'énoncé, un algorithme glouton ne garantit pas de maximiser le poids chargé (exemple : capacité 10 kg et colis de 6, 5 et 5 kg : le glouton charge 6 kg, alors que $5 + 5 = 10$ kg était possible).*
