Tri par insertion

I – Algorithlme du Tri par insertion

1.1 - Principe général

On trie un ensemble d'une carte, puis on en rajoute une qu'on place correctement pour avoir deux cartes triées, puis on rajoute une nouvelle carte...

→ DEBUT : → on prend la **première carte** pour former le sous-tableau trié [0;0]
→ les autres cartes forment le sous-tableau non-trié [1 ; fin]

→ EXECUTION : → on récupère la clé : la premiere carte du sous-tableau non trié.
→ on déplace vers la droite les cartes déjà triées qui sont supérieures à la clé.
→ on insère la clé dans la case disponible

1.2 - Algorithme

Vous devez être capable de → fournir l'algorithme ci-dessous (à connaître par coeur donc)
→ implémenter l'algorithme en Python (l'algorithme étant donné)

ENTREE : le tableau **t** contenant les éléments à trier.

SORTIE : aucune, **t** est modifié sur place .

ALGORITHME :

```
POUR i variant de 1 à (longueur – 1)
    cle ← t[i]
    j ← i – 1
    TANT QUE j ≥ 0 et que t[j] > cle
        t[j+1] ← t[j]
        j ← j – 1
    Fin TANT QUE
    t[j+1] ← cle
Fin POUR
Renvoyer VIDE (⌀)
```

II – Preuve de terminaison

La preuve de terminaison d'un algorithme permet d'affirmer :

- **qu'il s'arrête de façon certaine** (il ne boucle pas infiniment)
- **sur toutes les entrées valides qu'on lui fournit** (les entrées respectant les préconditions)

Pour **faire la preuve de terminaison**, il faut montrer que :

- les boucles s'expriment sous la forme **TANT QUE VARIANT > 0**
- avec un **VARIANT** pouvant s'écrire comme une suite u_n **strictement décroissante d'entiers**.

Ici, il faut montrer la terminaison des deux boucles.

Pour le TANT QUE, on le simplifie en **`TANT QUE j ≥ 0 :`** s'il se termine avec une condition, cela fonctionnera aussi avec deux conditions.

Pour le POUR, on peut soit juste vérifier ses bornes et son sens, soit le transformer en TANT QUE.

Vous devez pouvoir refaire les preuves (voir votre feuille).
CONCLUSION : l'algorithme du tri par insertion se termine.

III – Côuts

Le **coût** permet d'estimer la façon dont le nombre d'instructions varie lorsqu'on augmente le nombre de données **n** à traiter par l'algorithme.

Le coût dans le pire des cas correspond à ce coût dans la configuration la plus défavorable.

Côuts de l'algorithme du tri par insertion :
→ PIRE DES CAS : coût **quadratique** **$\Theta(n^2)$**
→ MEILLEUR DES CAS : coût **linéaire** **$\Theta(n)$**
→ De façon générale, on peut donc noter ***O*(n^2)**

Remarque : pour le démontrer, nous avons besoin d'utiliser ou de démontrer :

$$C_n = \sum_{x=1}^{n-1} x = \frac{(n-1)n}{2} \qquad \text{ou} \qquad \sum_{x=0}^{n} x = \frac{n(n+1)}{2}$$

Voici des exemples de coûts qui permettent de classer les performances des algorithmes :

Complexité	Nom du coût	Evolution des données	Evolution du temps d'exécution
$\Theta(\log_{10} n)$ ou $\Theta(\log n)$	Logarithmique (base 10)	1.10^3 vers 1.10^6 1.10^6 vers 1.10^9	Exécution x 2 uniquement Exécution x 1,5 uniquement
$\Theta(\log_{10} n)$ ou $\Theta(\log n)$	Logarithmique (base 2)	1.10^3 vers 1.10^6 1.10^6 vers 1.10^9	Exécution x 2 uniquement Exécution x 1,5 uniquement
$\Theta(n)$	Linéaire	1.10^3 vers 1.10^6 1.10^6 vers 1.10^9	Exécution x 1000 Exécution x 1000
$\Theta(n \log n)$	Quasi-linéaire	1.10^3 vers 1.10^6 1.10^6 vers 1.10^9	Exécution x 2000 Exécution x 1500
$\Theta(n^2)$	Quadratique	Fois 1000	Exécution x 1000^2, soit 1 million
$\Theta(n^X)$	Polynomial	Fois 1000	Exécution x 1000^X
$\Theta(x^n)$	Exponentiel	1 vers 1000	Exécution x 5.10^{300} si on a 2^X
$\Theta(n\ !)$	Factoriel (10*9*8..*1)	1 vers 1000	Python ne parvient même pas à le calculer...

IV - Preuve de correction

La preuve de correction d'un algorithme permet d'affirmer

- **qu'il fournit toujours la bonne réponse** (réalise sans erreur le travail pour lequel il est conçu)
- **sur toutes les entrées valides qu'on lui fournit** (les entrées respectant les préconditions).

Pour faire la preuve de correction, il faut trouver un INVARIANT.

Défintion d'un INVARIANT : une propriété P vérifiable qui reste vraie tout le long du déroulement de l'algorithme.

Pour l'algorithme de tri par insertion, on peut travailler avec l'INVARIANT $\mathbf{P_k}$ suivant :

P_k : après k tours de boucle, k+1 éléments sont triés dans le sous-tableau [0;k] de gauche.

La preuve de correction se fait en trois temps.

1. L'**initialisation** : on montre que **P_0 est VRAI** avant de rentrer dans la moindre boucle.
2. La **conservation** : on montre que **$P_k \rightarrow P_{k+1}$** en supposant que **P_k** est VRAI après k tours de boucles.
3. La **terminaison** : on montre que l'algorithme a bien traité toutes les donneés une fois sorti des boucles.

Voir la démonstration sur votre cours.