Algorithmes fondamentaux

Pour chacun des algorithmes fondamentaux, tu dois savoir :
- compléter une version « à trou »
- donner sa complexité temporelle avec une justification
- prouver sa terminaison

Comme pour tout autre algorithme, tu dois savoir :
- l'appliquer « à la main » sur un exemple
- l'implémenter en Python

I. Recherche d'un élément dans un tableau

PARAMETRES **t : tableau d'entiers** **valeur : nombre entier**

I.1. Parcours séquentiel

```
1  tr ← FAUX
2  i ← 0
3  tant que i<longueur(t) et que tr==FAUX:
4        si t[i]== valeur :
5           tr ← VRAI
6        i ← i+1
7  renvoyer la valeur de tr
```

Cet algorithme renvoie VRAI si l'élément a été trouvé ou FAUX sinon.

La complexité temporelle dans le pire cas est en O(n) car T(n) = T(n-1) + k où k est un entier.

I.2. Recherche par dichotomie dans une liste triée

```
1  i_min ← 0
2  i_max ← longueur(t) - 1
3  tant que i_min <= i_max :
4        i ← (i_min + i_max) // 2
5        si t[i]== valeur :   renvoyer  i
6        sinon si t[i] < valeur : i_min ← i + 1
7        sinon  i_max  ← i - 1
8  renvoyer None
```

Cet algorithme renvoie l'indice de l'élément s'il a été trouvé dans le tableau ou None sinon.

La complexité temporelle est en $O(\log_2 n)$ car $T(n) = T(n/2) + k$ où k est un entier.

II. Tri des éléments d'un tableau

II.1. Tri par insertion

La complexité temporelle est en $O(n^2)$ dans le pire cas mais en O(n) si la liste est déjà trié.

```
1  On dispose d'un tableau de nombres de longueur L
2  Pour i allant de 1 à (L)
3        stocker tab[i]
4        j=i
5        tant que (j>0) et tab[j-1]>valeur stockée
6              tab[j]=tab[j-1]
7              décrémenter j
8        remplacer tab[j] par la valeur stockée
9  retourner le tableau triée
```

II.2. Tri par sélection

La complexité temporelle est en $O(n^2)$ dans tous les cas mais le nombre d'échanges de valeurs est plus petit que pour le tri par insertion.

```
1  On dispose d'une tab de nombres de longueur L
2  Pour i allant de 0 à (L-1)
3        initialiser index_min à i
4        pour j allant de (index_min+1) à L
5              si tab[j]<tab[index_min]
6                    index_min = j
7        échanger tab[i] et tab[index_min]
8  retourner le tableau triée
```