Activité : Algorithmes de tri

Tri par sélection

Soit L un tableau de nombres entiers non trié. L'objectif est de trier ce tableau sans utiliser la méthode sort des listes python.

1) Créer un tableau T de valeurs choisies aléatoirement entre 1 et 1000.

2) On présente l'algorithme de tri par sélection ci-dessous. Comme en python, l'accès à une valeur du tableau se fait en donnant l'indice de position entre crochets.

```
# premier indice du tableau
i = 0
n = len(tableau)
tant que i < n:
  j = i
  position_valeur_minimale = i
  tant que j < n:
    si tableau[j] < tableau[position_valeur_minimale]:
      position_valeur_minimale = j
    j = j + 1
  si position_valeur_minimale != i
    on échange les valeurs d'indice i et position_valeur_minimale
  i = i + 1
```

Écrire la fonction tri_selection qui prend en paramètre une liste et renvoie la liste triée.

La fonction applique l'algorithme donné ci-dessus.

3)
- **a)** Combien de comparaisons sont nécessaires pour trier un tableau de 5 valeurs ?
- **b)** Combien de comparaisons sont nécessaires pour trier un tableau de 10 valeurs ?
- **c)** Combien de comparaisons sont nécessaires pour trier un tableau de 20 valeurs ?
- **d)** Combien de comparaisons sont nécessaires pour trier un tableau de n valeurs ?

4) Il est possible de mesurer le temps d'exécution d'une fonction avec le module timeit. Son usage diffère selon l'environnement : notebook ou idle.

On va utiliser le module dans un notebook.

- **a)** Importer dans une nouvelle cellule la fonction timeit du module timeit.
- **b)** Saisir les instructions comme sur la figure ci-dessous :

```
%%timeit

tri_selection(t)
```

- **c)** Réaliser plusieurs mesures en créant des tableaux de dimensions 10, 20, 40, 80 et 160.
- **d)** Comment évolue le temps d'exécution en fonction de la dimension du tableau.

Tri par insertion

Soit L un tableau de nombres entiers non trié. L'objectif est de trier ce tableau sans utiliser la méthode sort des listes python.

1) Créer un tableau T de valeurs choisies aléatoirement entre 1 et 1000.

2) On présente l'algorithme de tri par insertion ci-dessous. Comme en python, l'accès à une valeur du tableau se fait en donnant l'indice de position entre crochets.

```
# indice de la valeur à insérer (la valeur d'indice 0 est déjà insérée)
j = 1
n = len(tableau)
tant que j < n:
  i = j - 1
  valeur_a_inserer = tableau[j]
  tant que i >= 0 et tableau[i] > valeur_a_inserer :
    tableau[i + 1] = tableau[i]
    i = i - 1
  tableau[i + 1] = valeur_a_inserer
  j = j + 1
```

Écrire la fonction tri_insertion qui prend en paramètre une liste et renvoie la liste triée.
La fonction applique l'algorithme donné ci-dessus.

3)
- **a)** Combien de décalages sont nécessaires pour trier un tableau de 5 valeurs ?
- **b)** Combien de décalages sont nécessaires pour trier un tableau de 10 valeurs ?
- **c)** Combien de décalages sont nécessaires pour trier un tableau de 20 valeurs ?
- **d)** Combien de décalages sont nécessaires pour trier un tableau de n valeurs ?

4) Il est possible de mesurer le temps d'exécution d'une fonction avec le module timeit. Son usage diffère selon l'environnement : notebook ou idle.
On va utiliser le module dans un notebook.

- **a)** Importer dans une nouvelle cellule la fonction timeit du module timeit.
- **b)** Saisir les instructions comme sur la figure ci-dessous :

```
%%timeit

tri_selection(t)
```

- **c)** Réaliser plusieurs mesures en créant des tableaux de dimensions 10, 20, 40, 80 et 160.
- **d)** Comment évolue le temps d'exécution en fonction de la dimension du tableau.