NSI 1ère - Algorithmique - Tris 2

QK

Plan

td / tp : tri sélection,

1. algo avec indices : à la main,
2. trad python
3. prog
4. si le temps : complexité tri sélection, terminaison, inviariant, variant

Seconde partie : TD/TP sur le tri par sélection

Exercice papier : du pseudo code au programme.

Reprendre l'algorithme du tri par sélection présenté durant la séance précédente.

1. Donner les étapes du tri par sélection pour le tri du tableau `[5, 3, 2, 1, 4]`
2. Écrire un algorithme en langage naturel avec les indices nécessaires.
   On pourra utiliser une fonction `echanger(tableau, i, j)` qui échange les éléments d'indices `i` et `j` d'un tableau et ne renvoie rien.
3. Écrire la fonction `echanger` en python.
4. Traduire les algorithmes et fonctions précédentes en Python (toujours sur feuille).
5. Écrire un algorithme qui vérifie qu'un tableau est trié par ordre croissant. La signature de la fonction est : `est_trie(tableau: list) -> bool`.

Exercice sur poste : programmer le tri par sélection.

1. Écrire la fonction `echanger` en Python.
2. Programmer la fonction `tri_selection(tableau: list) -> None` qui prend un tableau d'entiers en paramètres et le trie avec le tri par sélection. Elle ne renvoie rien.
3. Écrire deux versions Python de la fonction `est_triee` :
   - avec `sorted()` (lire la documentation...)
   - avec l'algorithme proposé à la question 5 de l'exercice précédent.
4. Proposer un jeu de tests afin de vérifier notre tri par sélection.
5. Un seul échange est réalisé lors d'une étape de la boucle. On peut encore réduire ce nombre d'échange à l'aide d'une condition. Dans quel cas est-il inutile d'échanger ? Améliorer le tri par sélection (dans un nouvelle fonction).

Exercice papier : complexité du tri par sélection.

Nous allons chercher à *majorer* le nombre d'étapes nécessaires au tri par sélection.

1. Parmi toutes les manières d'ordonner la liste des entiers `1, 2, ..., n` quelle est celle qui engendre le plus d'échanges lors du tri par sélection ?
2. On se place dans le cas précédent. La première étape de la boucle consiste à déterminer la position de l'élément minimal dans la partie "non triée" du tableau.
   Combien de comparaisons sont nécessaires pour déterminer le minimum d'un tableau comportant k éléments ?
3. Proposer une formule permettant de compter le nombre de comparaisons à effectuer.
4. Simplifier cette formule à l'aide de votre cours de mathématiques sur la somme des termes d'une suite arithmétique.

Conclusion : dans le pire des cas, il faut réaliser de l'ordre de n^2 comparaisons lors d'un tri par sélections. En admettant que ces comparaisons aient toutes une durée égales (on dit qu'elles sont en temps constant), on dit que le tri par sélection est un algorithme en $O(n^2)$.

Exercice papier : preuve et terminaison du tri par sélection

1. Vérifier que, dans l'lgorithme du tri par sélection, la première partie du tableau est toujours constituée d'éléments triés.
   C'est un *invariant* de boucle.
2. Vérifier que, dans l'algorithme du tri par sélection, la seconde partie du tableau a une taille qui diminue à chaque tour de la boucle.

Ces deux éléments prouvent que :

2. L'algorithme se *termine*
3. Le tableau final *est trié.*

Tri par sélection

```
tri_selection(tableau t, entier n)
    pour i de 1 à n - 1
        min = i
        pour j de i + 1 à n
            si t[j] < t[min], alors min = j
        fin pour
        échanger t[i] et t[min]
    fin pour
```