

5.3

Tri par sélection

NSI 1ÈRE - JB DUTHOIT

5.3.1 Principe

Principe : on cherche le minimum des éléments non triés et on le place à la suite des éléments triés

VIDÉO http://lwh.free.fr/pages/algo/tri/tri_selection.html

5.3.2 Le tri sélection sur un exemple

Exercice 5.18

Considérons la liste suivante :

3	7	2	6	5	1	4
---	---	---	---	---	---	---

Compléter les lignes suivantes, sachant que les cases grisées correspondent au début de liste, déjà trié, par la méthode du tri par sélection.

3	7	2	6	5	1	4

5.3.3 Quelques algorithmes préliminaires

Exercice 5.19

- Construire un algorithme d'une fonction `echange(t,i,j)` avec `t` un tableau d'entiers et `i` et `j` deux entiers. La fonction doit modifier le tableau `t` dans lequel on a échangé les valeurs situées à l'indice `i` et `j`
- Implémenter cet algorithme en python. l'appel `echange([1,2,3,4,5],2,3)` va modifier le tableau `t` : on aura `t = [1,2,4,3,5]`.

Exercice 5.20

- Construire un algorithme d'une fonction `plus_petit(t,i)` avec `t` un tableau d'entiers et `i` un entier entier.
La fonction doit renvoyer l'indice de la valeur minimale du tableau, **en considérant les valeurs du tableau à parti de l'indice `i` inclus (et donc jusqu'à la fin du tableau)**.
- Implémenter cet algorithme en python. l'appel `plus_petit([1,2,13,4,3],2)` va renvoyer `4` : `4` est l'indice de la valeur `3`, qui est la plus petite valeur du tableau `[1,2,13,4,3]` (On commence à parcourir le tableau à partir de l'indice `2`).

5.3.4 L'algorithme

En utilisant les questions préliminaires précédentes, on arrive à l'algorithme suivant :

```
1 FONCTION tri_selection(T :tableau d'entiers)
2   POUR i DE 0 A longueur(T)-2 FAIRE
3     indice : entier
4     indice = plus_petit(T,i)
5     echange(T,indice,i)
6   RENVOYER T
```



Savoir-Faire 5.18

SAVOIR CONSTRUIRE L'ALGORITHME CORRESPONDANT AU TRI PAR SÉLECTION

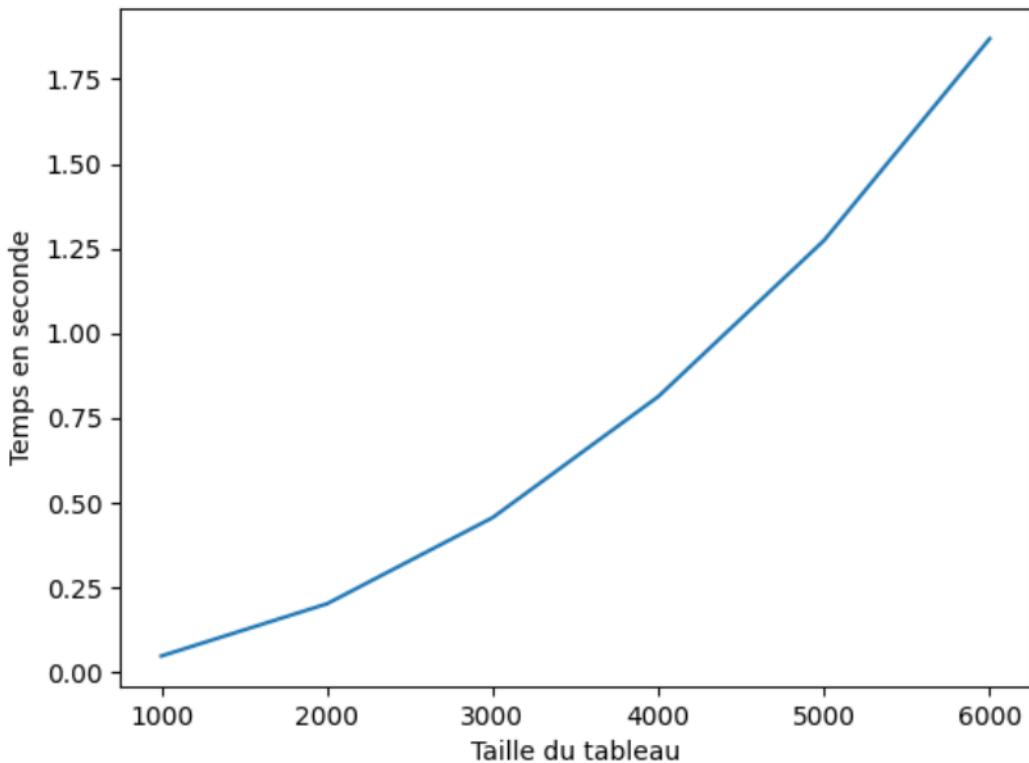
5.3.5 Implémentation en Python

5.3.6 Complexité temporelle

Étude empirique

Exercice 5.21

1. Construire une liste de 1000 éléments, avec des entiers de 0 à 999 rangés dans l'ordre décroissant.
2. Appliquer la fonction `tri_selection` à cette liste et calculer le temps d'exécution. Garder la valeur en mémoire.
3. Recommencer les deux questions précédentes pour une liste de 2 000 , puis 3 000 etc jusqu'à 6 000. Garder les résultats.
4. Placer les points obtenus aux questions précédentes sur un graphique, afin de visualiser la durée d'exécution de la fonction recherche en fonction de la taille de la liste.
Vous devriez obtenir un graphique comme celui-ci :



Durée du tri sélection en fonction de la taille du tableau.

Meilleur des cas - Pire des cas

Pour analyser la complexité de cet algorithme, nous allons analyser le nombre de comparaisons effectué ainsi que le nombre d'échange lors du tri.

Or, le nombre de comparaisons à effectuer ne dépend pas de la liste (du fait qu'elle soit triée ou non).

En conséquence, la complexité en moyenne et dans le pire des cas sera identique.

Notons S_n le nombre de comparaisons pour déterminer le minimum dans un tableau de longueur n et T_n le nombre de comparaisons dans l'algorithme du tri sélection.

Calculer T_n et en déduire la complexité moyenne et dans le pire des cas ?

5.3.7 Pré et post conditions

5.3.8 Terminaison

5.3.9 Validité de l'algorithme

Invariant de boucle

Pour prouver la correction, nous utilisons un invariant de boucle :

Pour chaque i , et à la fin de la boucle, on a :

"la liste $[0 ; i+1[$ est triée par ordre croissant, et les éléments de la liste $[i+1 ; n[$ sont tous supérieurs à tous les éléments de la liste $[0 ; i+1[$ "

Démonstration



Démonstration 1

Étape 1 : La propriété est-elle vraie pour $i=0$? :

Étape 2 : On suppose la propriété vraie pour l'entier k . Est-elle vraie pour l'entier $k+1$?

Etape 3 : Et pour le dernier passage dans la boucle ?

Étape 4 : Conclusion