

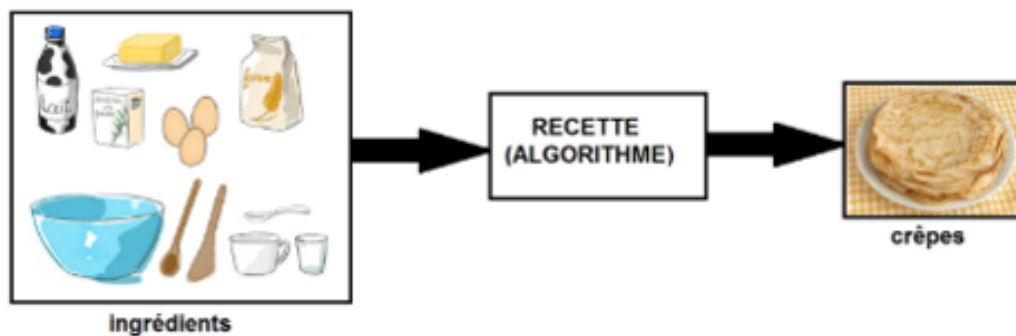
Algorithmie: Recherche d'occurrences

1 Qu'est ce qu'un algorithme ?

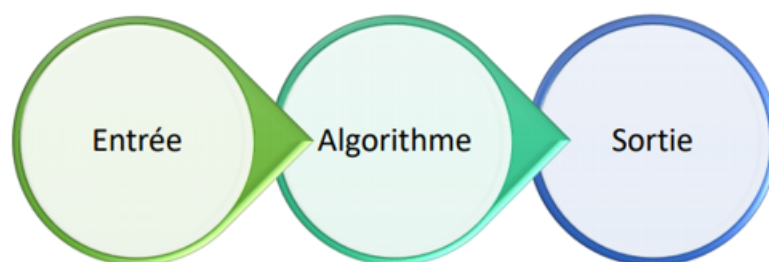
1.1 Définition

Définition 4.1

Un algorithme est une procédure pas-à-pas de résolution d'un problème s'exécutant en un temps fini.



Un délicieux algorithme :-)



La plupart des algorithmes transforment des données d'entrée en des données de sortie.

1.2 Les points essentiels d'un algorithme

La preuve : Il faut être en mesure de prouver que l'algorithme résoud bien le problème pour lequel il a été écrit

L'arrêt : Il faut prouver que l'algorithme s'arrête en un nombre fini d'étapes, et ceci dans tous les cas de figure.

Le choix des structures de données : La façon d'organiser et d'utiliser les données influence l'écriture et la performance de l'algorithme.

Son efficacité : Il est très important d'avoir une idée du comportement de l'algorithme lorsque la taille des données passée en argument augmente.

1.3 Le pseudo code

1.3.1 Définition

En programmation, le pseudo-code, également appelé LDA (pour Langage de Description d'Algorithmes) est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.

L'écriture en pseudo-code permet de comprendre les difficultés de la mise en œuvre de l'algorithme et de développer une démarche structurée dans la construction de celui-ci.

En effet, son aspect descriptif permet de décrire avec plus ou moins de détail l'algorithme, permettant de ce fait de commencer par une vision très large en s'affranchissant de la syntaxe des langages de programmation.

Il n'existe pas de réelle convention standardisée pour le pseudo-code. Nous utiliserons le pseudo-code utilisé à L'Ecole Supérieure d'Informatique.

Les mots-clés seront écrits en majuscules et soulignés Les variables ne contiendront aucun espace, aucun caractère accentué, ne commenceront pas par un chiffre et seront écrites en minuscules Les commentaires seront précédés par `//`.

1.3.2 Le pseudo-code

Afficher un texte : `ECRIRE "Veuillez entrer les nom et prénom"`

Afficher une variable `ECRIRE nom,prenom`

Afficher une variable et du texte `ECRIRE "Les contenus de val1 et val2",val1," et ",val2`

Déclarer les variables et affectation (internes et externes) :

```
val1:entier      // déclaration de variable , ici val1
                  //est du type entier
val2:reel
flag:logique
nom,prenom:chaine
val1 <- 5        // Affectation interne, ici val1 prend //la valeur 5
val2 <-5.5
flag <- VRAI
LIRE nom, prenom // Affectation externe: //l'utilisateur entre les valeurs
```

Les opérateurs arithmétiques :

*	multiplication	>	plus grand que (greater than)
/	division	<	plus petit que (less than)
+	addition ou concaténation	>=	plus grand ou égal à (greater than or loose-equals)
-	soustraction	<=	plus petit ou égal à (less than or loose-equals)
<u>MOD</u>	reste d'une division	=	égalité
<u>DIV</u>	division entière	!=	inégalité
**	exposant		

Les opérateurs logiques :

<u>ET</u>	et
<u>OU</u>	ou inclusif
<u>XOU</u>	ou exclusif
<u>NON</u>	inverse de

Les boucles conditionnelles

```

SI (val1 > 5) ALORS
    ECRIRE "La valeur",val1,"est supérieure à 5"
FINSI
SI (val1 > 5) ALORS
    ECRIRE "La valeur",val1,"est supérieure à 5"
SINON
    ECRIRE "La valeur",val1,"est égale ou inférieure à 5"
FINSI
SI (val1 > 5) ALORS
    ECRIRE "La valeur",val1,"est supérieure à 5"
SINONSI (val1=5) ALORS
    ECRIRE "La valeur",val1,"est égale à 5"
SINON
    ECRIRE "La valeur",val1,"est inférieure à 5"
FINSI

```

Les boucles :

```

TANT QUE (cpt < val1)
    ECRIRE "Tour n°",cpt
    Cpt <- Cpt + 1
FINTANT

POUR(i DE 0 A 5 PAR 1) FAIRE
    ECRIRE "Tour n°",i
FINPOUR

```

Les tableaux :

```

montab1:tableau[0,15]  de entier
// pour déclarer un tableau (une liste // en python)
montab2:tableau[0,5]  de chaine
montab3:tableau[0,9]  de reel
montab4:tableau[0,126] de logique

montab:tableau[0,5] de entier //pour initialiser le tableau avec des 0
i:entier
i <- 0
TANT QUE(i <= 5) FAIRE
  montab[i] <-0
  i <-i + 1
FINTANT

POUR(i DE 0 A 5 PAR 1 FAIRE)      // pour parcourir un tableau
  ECRIRE "Veuillez entrer une valeur"
  LIRE montab[i]
FINPOUR
  //Affichage du tableau
POUR(i DE 0 A 5 PAR 1 FAIRE)
  ECRIRE montab[i]
FINPOUR

```

Les fonctions ou procédures :

```

FONCTION politesse()      //Procédure sans arguments
  ECRIRE "Bonjour"
FIN FONCTION

politesse()      // pour appeler la fonction ou la procédure politesse

FONCTION calcul(val1:entier, val2:entier)
  res:entier
  res = val1 + val2
  RETOURNER res
FIN FUNCTION

val1, val2, res:entier  //Programme principal
LIRE val1, val2
res = calcul(val1,val2)

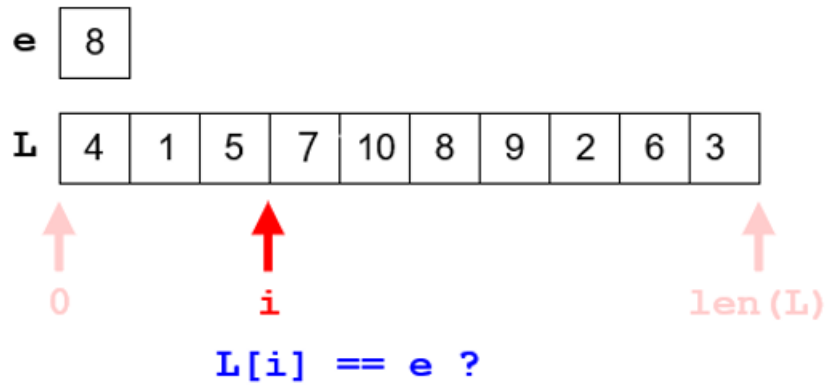
```

2 Recherche d'occurrences

2.1 Algorithme de recherche d'occurrences

La recherche d'une occurrence consiste à déterminer la position (l'indice i) d'un élément e présent dans la liste L . On parle d'occurrence de e dans L .

Exemple : Je recherche l'occurrence de 8 dans la liste L :



Exercice 4.1

Ecrire un algorithme de recherche d'une occurrence utilisant une boucle POUR.

L'algorithme est une fonction qui prend en argument un tableau d'entiers et une valeur entière, et qui renvoie l'occurrence de cette valeur (FAUX si la valeur n'est pas présente dans le tableau).

Exercice 4.2

Ecrire un algorithme de recherche d'une occurrence en utilisant une boucle TANT QUE.

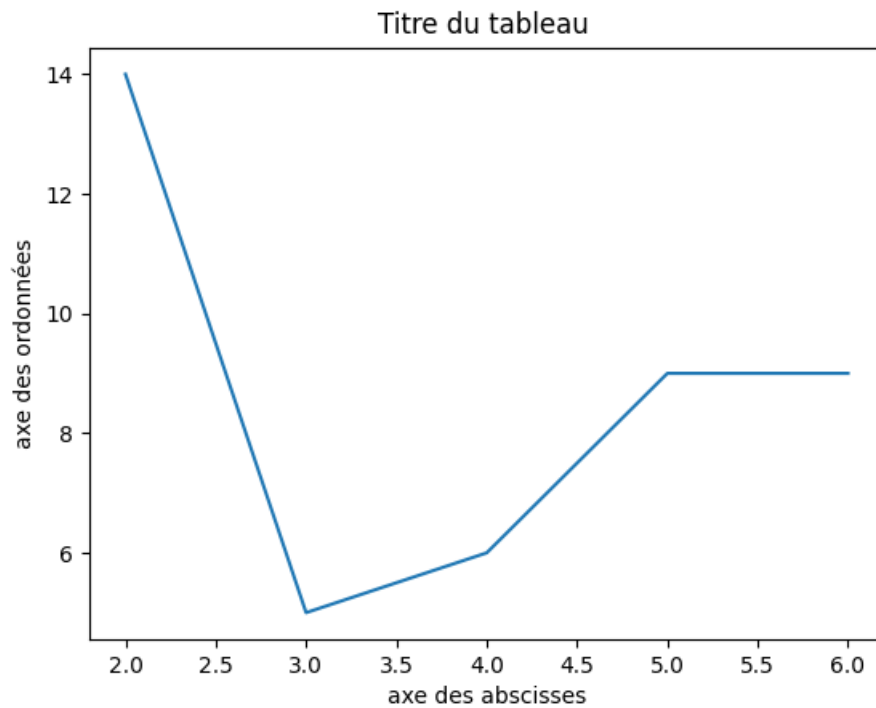
L'algorithme est une fonction qui prend en paramètres un tableau d'entiers et une valeur entière, et qui renvoie l'occurrence de cette valeur (FAUX si la valeur n'est pas présente dans le tableau).

2.2 Durée d'exécution

Exercice 4.3

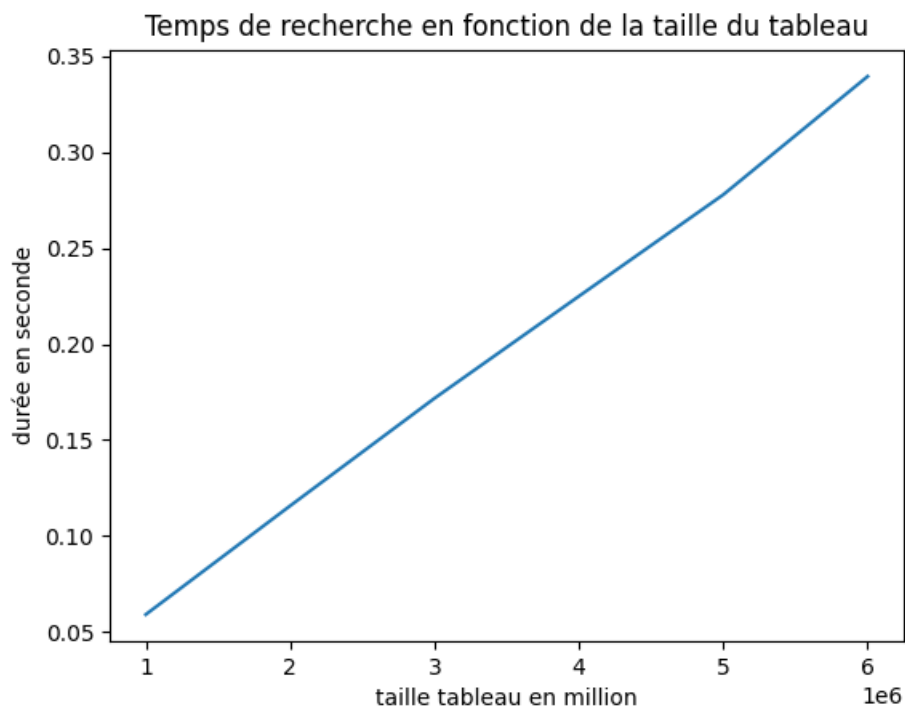
1. Construire une liste de 1 million d'éléments, avec des entiers de 0 à 999999.
2. Appliquer un des algorithmes précédents à cette liste et calculer le temps d'exécution pour la recherche d'un élément qui n'est pas dans le tableau (afin d'être sûr qu'on parcourt le tableau en entier). Garder la valeur en mémoire.
3. Recommencer les deux questions précédentes pour une liste de 2 000 000 , puis 3 000 000 etc jusqu'à 6 000 000. Garder les résultats.
4. Observer ce code qui permet l'affichage du graphique ci-après.

```
import matplotlib.pyplot as plt
plt.title('Titre du tableau ')
plt.plot([2,3,4,5,6],[14,5,6,9,9])
plt.xlabel('axe des abscisses')
plt.ylabel('axe des ordonnées')
plt.show()
```



Exemple d'utilisation de matplotlib

5. 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 :



On en déduit, de façon empirique, que le temps de recherche est linéaire par rapport à la taille du tableau.

2.3 Complexité temporelle

Nous allons essayer de formaliser la notion précédente et d'estimer le coût en temps de cet algorithme.

☞ Nous supposons que chaque opération (affectation, comparaison, opération arithmétique, ...) a un coût de 1 « unité » de temps.

Reprenons un des deux algorithmes, celui avec la boucle pour par exemple :

```
1 FONCTION OCCURRENCE(T :tableau d'entiers, element :entier)
2   POUR i DE 0 A longueur(T)-1 FAIRE
3     SI T[i] = element ALORS
4       RENOYER i
5   RENOYER False
```

On suppose que la taille du tableau est n :

ligne 2 : au pire n affectations pour i plus 1 interrogation de longueur de tableau

ligne 3 : au pire, n comparaisons

Soit $T(n)$ le nombre d'opérations. $T(n) = 2n + 1$. On a donc une complexité en $\mathcal{O}(n)$, soit une complexité linéaire.

● Exercice 4.4

1. Créer l'algorithme de la fonction **maximum(T)** qui reçoit en argument un tableau d'entiers et qui retourne le maximum du tableau.
2. Calculer la complexité temporelle de cette algorithme.
3. Implémenter cet algorithme en Python

● Exercice 4.5

1. Créer l'algorithme de la fonction **minimum(T)** qui reçoit en argument un tableau d'entiers et qui retourne le minimum du tableau.
2. Calculer la complexité temporelle de cette algorithme.
3. Implémenter cet algorithme en Python

3 Tri par sélection

3.1 Principe

3.2 L'algorithme



Savoir-Faire 4.1

SAVOIR CONSTRUIRE L'ALGORITHME CORRESPONDANT AU TRI PAR SÉLECTION

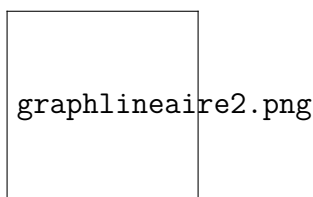
3.3 Implémentation en Python

3.4 Complexité temporelle

3.4.1 Etude empirique

● Exercice 4.6

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 :



3.4.2 Meilleur des cas

Si la liste est déjà triée!

3.4.3 Pire des cas

La liste est triée dans l'ordre inverse!

3.5 Pré et post conditions

3.6 Terminaison

3.7 Correction

3.7.1 Invariant de boucle

Pour prouver la correction, nous utilisons un invariant de boucle H : «pour chaque i , les $i + 1$ premiers éléments sont triés par ordre croissant à l'issue (à la fin) de l'itération i »

3.7.2 Démonstration