# Tris simples

|  |  |  |
| --- | --- | --- |
| **Programme** | **Contenus** | **Capacités attendues** |
| Algorithmique | Tris par insertion, par sélection | Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection. |
| Algorithmique | Parcours séquentiel d’un tableau | Écrire un algorithme de recherche d’une occurrence sur des valeurs de type quelconque. Écrire un algorithme de recherche d’un extremum, de calcul d’une moyenne. |
| Données 2 - Types construits | Tableau indexé, tableau donné en compréhension | Lire et modifier les éléments d’un tableau grâce à leurs index.  Construire un tableau par compréhension.  Utiliser des tableaux de tableaux pour représenter des matrices : notation a [i] [j].  Itérer sur les éléments d’un tableau. |

## Introduction aux tris

### Cartes

Vous possédez une dizaine de cartes portant chacune un numéro (deux cartes peuvent avoir le même numéro).

1. Trier vos cartes et essayer de voir votre mode de raisonnement.
2. Expliquer votre raisonnement.

### https://lh3.googleusercontent.com/rZjk900DqB0TjFreKQSsaASA0eyVnM8esLzvhOwy77DeTaNxclHKtBXt3q2L3ZebWnF7ag9dLaeGkzQeqQeP3dvJ7QbsAfzM5Nnhc3kpxJV5a0R4HRC8q-eZXkS0TLlUj5YOFuf3Pourquoi trier ?

Un exemple pour commencer :

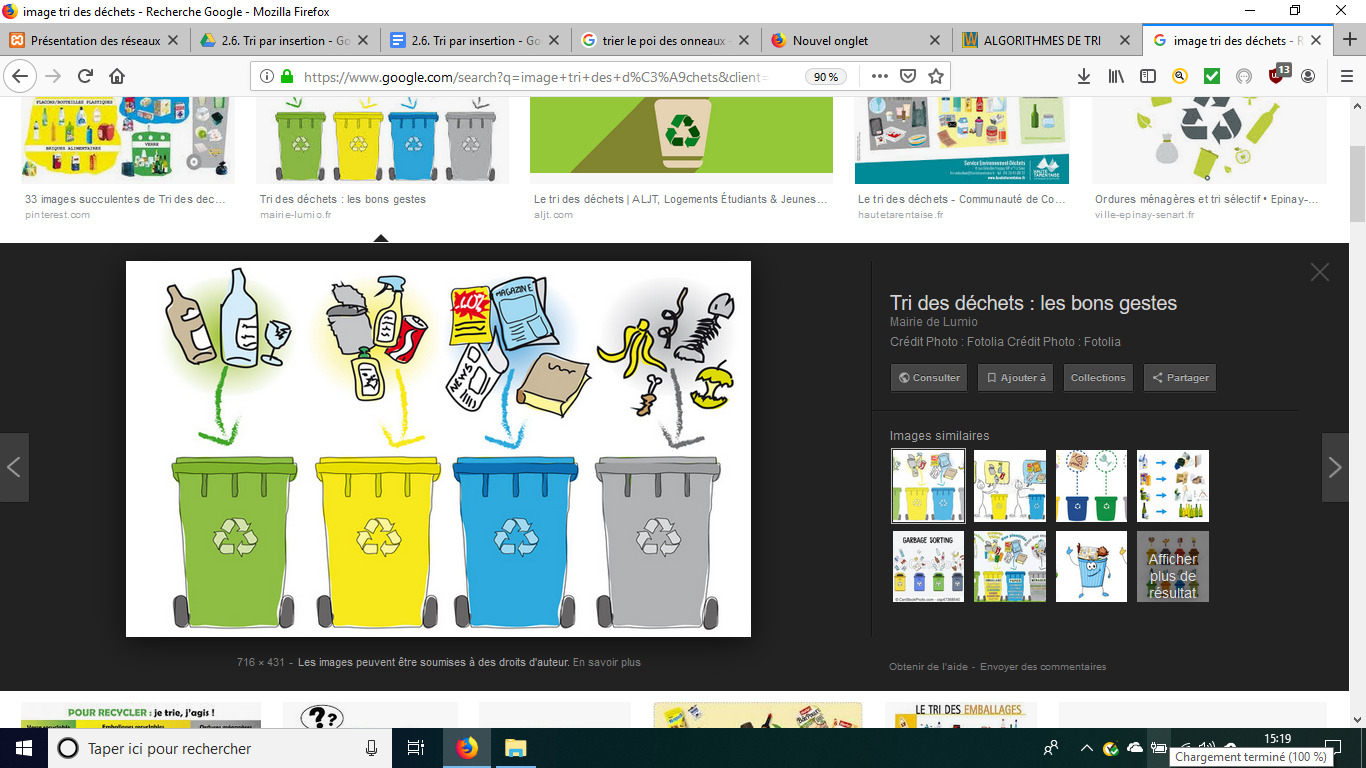
Si je souhaite acheter un nouvel ordinateur, je me rends donc sur un site de vente en ligne pour comparer les prix et bien évidemment les caractéristiques de chaque ordinateur pouvant l'intéresser (le prix n’est pas toujours gage de qualité).

Cet exemple illustre le besoin et la nécessité de trier, d’organiser les données. On peut organiser les données suivant le prix par ordre croissant, ou même par nom du processeur.

Cet exemple illustre le fait qu’il n’y a pas qu’une seule façon de trier. On a ici :

* Ordre lexicographique (nom de processeur, classer ses fichier dans un dossier,…)
* Ordre des prix (cf exemple precedent)

On retrouve ces deux ordres assez régulièrement

* ordre lexicographique : consitition des listes electorale, élaboration d’un disctonnaire…
* Ordre « classque » en statistique, le fait de claser les données permet de retrouver assez immédiateùent des éléments important
  + Minimum, maximum
  + Médiane, quartiles …

Mais on peut aussi penser au tri des déchets !

Le problème de trier a depuis longtemps était étudié par les informaticiens. C’est une action qui est nécessaire pour parcourir des listes, des arbres, des tables …

### Comment trier ?

Il est indispensable de pouvoir trier mais reste à savoir comment trier. Il existe en effet plusieurs façons de trier.Le choix d’un tri nécessite donc de se poser des questions :

* *Qu’est que trier :*

C’est « organiser » une collection d’objet suivant certain ordre

* *Que peut-on trier ?*

Des objet ayant une caractéristique commune

* *Trier par rapport à quel critère ?*

*C’est surement la question la plus importante. En effet si on se donne une liste de mots on peut par exemple trier ces mot dans l’ordre lexicographique, nombre de lettre, croissant, décroissant…*

### Définition

Il existe de nombreux algorithmes de tri qui ont été développés par les informaticiens. Les algorithmes n’ont évidemment pas la même efficacité, ni même la même complexité et ne sont pas basés sur les mêmes structures de données (des nombres, des tableaux, des listes, des arbres, des mots …)

### Générer des listes aléatoires à trier

fromrandom import randrange

deftri(liste\_a\_trier) :

# Inscrire ici le code de la fonction de tri

t = [randrange(0,100) for i in range (50)]

print(t)

tri(t)

print(t)

* Introduction aux algorithmes de tri :  
  <https://www.lumni.fr/video/les-algorithmes-de-tri>
* Visualiser les algorithmes de tri : <http://lwh.free.fr/pages/algo/tri/tri.htm>
* Comparer les vitesses d’exécution des algorithmes de tri : <https://www.toptal.com/developers/sorting-algorithms>
* Comparer les algorithmes de tri en musique : <https://www.youtube.com/watch?v=kPRA0W1kECg>

## Tri du singe

1. Écrire une fonction est\_trie\_dans\_ordre\_croissanttel que:

def**est\_trie\_dans\_ordre\_croissant(liste: list) 🡪bool:  
 """**

renvoie**True** si la liste est triée et **False** sinon

"""

Un singe trie les cartes de la façon suivante : il prend les cartes, les jette en l’air, les ramassepuis regarde si elles sont triées. Si oui, il s’arrête, sinon il relance les cartes.

1. Ce programme est-il correct ? (i.e. : se termine-t-il toujours ?)
2. Implémenter ce tri en Python
3. Le tester sur le tableaut=[3,1,9,4,4,5]
4. Exécuter plusieurs fois la fonction en affichantle nombre d’exécutions nécessairespourtrier « correctement » le tableau.

## Tri postal

* <http://www.france-ioi.org/algo/task.php?idChapter=538&iOrder=3>

Le tri postal est vraiment très simple. On crée plein de boîtes, exactement une par valeur possible. Pour chaque valeur à trier, on la met dans la boite correspondante. Ensuite, on parcourt toutes les boîtes dans l'ordre, et on affiche leur contenu.

Créer le programme Python qui permet de trier un tableaud’entiers entre 0 et 100 inclus.

## Tris quadratiques

### [www.france-ioi.org](http://www.france-ioi.org)

[Niveau 3.6 – Tris simples](http://www.france-ioi.org/algo/chapter.php?idChapter=525)

## Tri par insertion

### Principe

Le tri par insertion est un tri intuitif, on dit aussi naïf, en ce sens qu’il fait partie des tris que l’on applique naturellement seul.

Par exemple, on utilise ce tri pour classer un jeu de cartes ou pour classer des copies dans un ordre croissant (ou décroissant, voire alphabétique). Dans tous les cas l’idée est alors de prendre une carte et de la classer par rapport aux autres (on cherche sa place dans le tas déjà trié).

* On prend les deux premières cartes que l’on range dans l’ordre croissant ;
* On prend la troisième carte et on la range par rapport aux deux premières. On a donc un ensemble de trois cartes rangées ;
* On prend une quatrième carte que l’on place par rapport aux trois premières ;
* On continue jusqu’à la dernière carte.

### Algorithme

Comment modéliser l’insertion d’une carte ?

On dispose, après plusieurs tris, de quatre cartes triées et on en a une cinquième en main (on la « met » de côté).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 15 | 34 | 38 | 35 |

On a donc maintenant devant nous 5 espaces dont les quatre premiers sont occupés.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 15 | 34 | 38 |  |

On compare la nouvelle carte avec la dernière. Comme on va décaler le 38 vers la droite (il y a une place de disponible puisque l’on a cinq cartes qui devront être triées).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 15 | 34 |  | 38 |

On compare désormais 35 et 34, et , donc le 35 doit être à droite du 34 et avant le 38.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 15 | 34 | 35 | 38 |

Une solution est donc le décalage à droite pour créer un espace libre entre la carte à laquelle on compare la carte en main et la prochaine carte avec laquelle on aurait à la comparer.

Lorsque l’on regarde le principe du tri, on répète un même type d’instructions un nombre de fois identique au nombre de cartes :

Il faut prendre une carte dans la main (au tour, on prend la carte numérotée, il s’agit de la i+1-ème carte du jeu). On utilise pour cela une variable temporaire.

On va commencer par comparer avec la carte la plus à droite. Il faut donc une autre variable qui va prendre les différentes positions des cartes dans la liste triée.

Il reste maintenant à comparer la carte en main aux autres suivant les explications ci-dessus. On répète alors une suite d’instructions tant que la carte en main (la valeur val) est inférieure à la carte à laquelle on la compare soit une boucle conditionnelle :

Il faut aussi s’assurer qu’il reste des cartes auxquelles comparer la carte en main pour faire le test.

La boucle conditionnelle devient alors :

Quand le test précédent est VRAI, on décale la carte d’un rang vers la droite.

Et on compare avec la carte à gauche :

Quand le test précédent est FAUX, on insère la carte val à sa place.

### Python

def tri(carte:list):

    carte2=carte.copy()

    for i in range(len(carte)):

        temp=carte2[i]

        carte.remove(temp)

        carte.append(None)

        a=len(carte)-2

        while temp<carte[a] and a>=0:

            carte[a+1],carte[a]=carte[a],carte[a+1]

            a-=1

        carte[a+1]=temp

    return carte

### Exemple

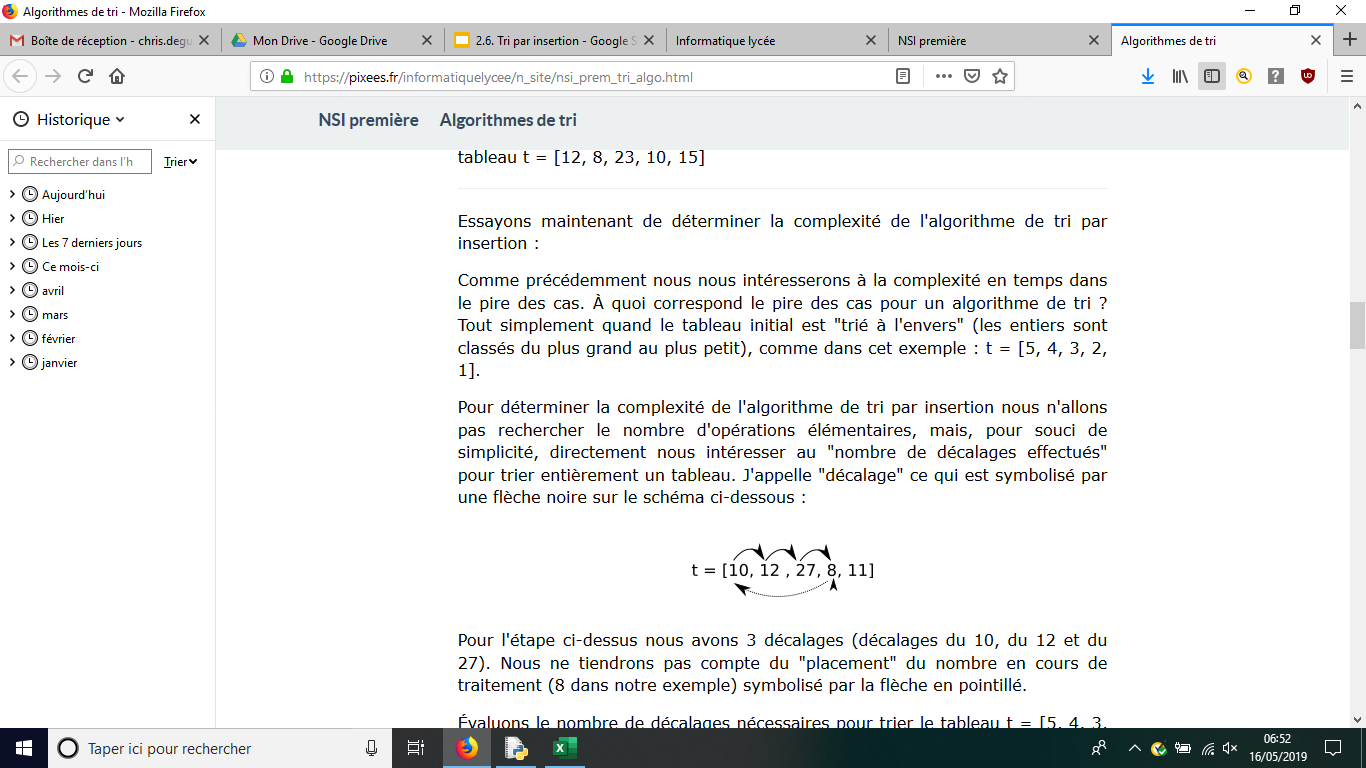
On peut le faire fonctionner sur un exemple, en complétant un tableau

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T | i | val | j | while |
| 5, 3, 7, 2 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### Complexité

#### Exercice

1. En complétant un “grand tableau” faire fonctionner l’algorithme sur la liste[12, 8, 23, 10, 15]
2. Afin de déterminer la complexité de notre algorithme on va compter le nombre de décalage. Par exemple : On peut compter 4 décalages dans l’exemple ci-dessous



1. Déterminer le nombre de décalages pour la liste [6,5,4,3,2,1].
2. Déterminer le nombre de décalages pour un tableau de taille n rangé dans l’ordre croissant.
3. Déterminer le nombre de décalages pour un tableau de taille n rangé dans l’ordre décroissant.
4. À quoi correspond le pire des cas pour l'algorithme de tri ?
5. À quoi correspond le meilleur des cas pour l'algorithme de tri ?

#### Complexité de l’algorithme de tri par insertion

Dans notre cas, on va s’intéresser à la complexité en temps et on va définir 2 opérations élémentaires :

* Test ou comparaison de deux nombres
* Affectation

Pour chaque tour de la boucle POUR :

* Il y a 3 affectations (et 2 comparaisons correspondant à la dernière comparaison de la conditionnelle while) ;
* On se place à l’étape i, l’algorithme effectue (boucle while)
  + 2\*i comparaisons
  + 2\*i affectations

Donc une complexité quadratique. On note la complexité

### Preuve de l’algorithme de tri par insertion

#### Variant de boucle

Un variant de boucle est une valeur entière qui répond à deux critères.

La valeur doit :

* être positive ou nulle ;
* être strictement décroissante.

#### Invariant de boucle

On appelle invariant d’une boucle une propriété :

1. Qui est vérifiée avant d'entrer dans la boucle : **initialisation**
2. Qui si elle est vraie avant l’exécution d’une itération le demeure après l’exécution de celle-ci : **propagation**
3. Qui lorsqu'elle est vérifiée en sortie de boucle, permet d'en déduire que le programme produit le résultat attendu : **terminaison**

On veut démontrer que l’on a un tableau trié. Notre invariant de boucle est donc que le tableau est trié à la fin de chaque tour.

On va alors vérifier ces trois éléments dans le cas de notre algorithme.

* Avant la première itération de la boucle POUR (i=1), on dispose d’un tableau à 1 élément, il est donc trié.
* On suppose que l’on dispose d’un tableau à k élément trié. On ajoute un élément (tableau non trié). La boucle TANT QUE permet de déplacer …d’une position vers la droite jusqu'à ce que l’on trouve la bonne position pour Une fois la place trouvée, la valeur est insérée). Le tableau en sortie est donc trié après une itération de la boucle POUR
* La boucle POUR se termine lorsque. En substituant par dans l’invariant de boucle, on a donc au final un tableau trié à n éléments. Notre tableau à éléments de départ est donc trié.

### Stabilité d’un algorithme de tri

On souhaite trier ces bouteilles par ordre croissant de contenance.

Une image contenant capture d’écran

Description générée automatiquement

On peut avoir les deux résultats ci-dessous :

|  |  |
| --- | --- |
| Tri Stable | Tri Non Stable |
| Une image contenant capture d’écran  Description générée automatiquement | Une image contenant capture d’écran  Description générée automatiquement |

Questions

* Quelle(s) différence(s) y a-t-il entre les deux ?
* De quel type est l’algorithme de tri par insertion ?

### Exercices

#### Tri par insertion décroissant

Écrire l’algorithme du tri par insertion qui permet de ranger les valeurs dans l’ordre décroissant.

#### Complexité

1. Quelle est la complexité au pire cas du tri du singe ? Au meilleur cas ? pire : n^2 meilleur : 1
2. Quelle est la complexité « au pire cas » du tri par insertion ?

## Tri à bulle

On considère l’algorithme de tri suivant :

def tri\_bulle(liste):

n = len(liste)

for i in range(n-1):

for j in range(n-i-1):

if liste[j] >liste[j+1]:

liste[j], liste[j+1] = liste[j+1], liste[j]

1. Expliquer les deux dernières lignes du script Python.
2. On considère la liste . Faire fonctionner cet algorithme sur cette liste. Déterminer le nombre d’opérations élémentaires effectuées (comparaison et affectation)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liste | i | j | test | affectation |
|  | 0 | 0 | 1 | 2 |
| 2,5,7,3,1 | 0 | 1 | 1 | 0 |
| 2,5,7,3,1 | 0 | 2 | 1 | 2 |
| 2,5,3,1,7 | 0 | 3 | 1 | 2 |
| 2,5,3,1,7 | 1 | 0 | 1 | 0 |
| 2,3,5,1,7 | 1 | 1 | 1 | 2 |
| 2,3,1,5,7 | 1 | 2 | 1 | 2 |
| 2,3,1,5,7 | 2 | 0 | 1 | 0 |
| 2,1,3,5,7 | 2 | 1 | 1 | 2 |
| 1,2,3,5,7 | 3 | 0 | 1 | 2 |

Total : 10 tests, 14 affecations (+2 si on compte def et n) => 24 opération

1. Expliquer « comment » fonctionne cet algorithme.

Cet algorithme compare 2à 2 les valeurs adjacentes et les ordonne si elle sont dans le mauvais ordre

1. Correction de l’algorithme :
   1. Justifier que l’algorithme se termine.

On prend comme variant de la boucle le nombre d’éléments restants à trier soit j = n-i-1. A chaque itération de la 1ère boucle for i est incrémenté de 1 jusqu’à n-1, donc les valeurs de la boucle j diminue jusqu’à atteindre 0. L’algorithme se termine.

* 1. Montrer que l’algorithme est correct(On pourra remarquer qu’à chaque itération de i, l’élément le plus grand se trouve à droite !).

Preuve de la correction du tri à bulles :

Terminaison : On prend comme variant de boucle le nombre d’éléments restants à trier soit n-i.

À chaque itération de la 1ère boucle, i est incrémenté de 0 à n-1.

Hypothèse ; À la fin de ‘i’ itérations de la boucle extérieure « for », les « n -i» éléments les plus élevés du tableau sont triés et occupent les index de ‘n-i-1’ à ‘n-1’.

Initialisation : Pour ‘i=0’, l’hypothèse dit qu’à la fin de la première itération de la boucle extérieure « for », l’algorithme donne l’élément le plus élevé à l’index « n ». En effet à la fin de l’itération pour i = 0, on a le plus grand nombre d’élément de la liste en position n-1. C’est une liste à 1 élément, elle est donc triée.Propagation : à la fin de ‘i’ itérations de la boucle extérieure « for », les « n-1 » éléments les plus élevé du tableau sont triés et occupent les ndices de « n-i-1 » à « n-1 ». Démontrons que pour la valeur i+1 la liste de n-i-2 à n est triée : on raisonne sur une liste allant de 0 à n-i-2

A la fin de la boucle intérieur (celle sur j), l’élément le plus grand sse trouvera en position n-i-2 . par conséquent, la liste allant de n-i-2 à n-1 est triée. L’hypothèse se propage.  
Conclusion : à la fin de « n-1 » itérations de la boucle « for » extérieure, tous les éléments du tableau sont trié et occupent les index de « 0 » à « n-1 »

Complexité :Pour une liste de 1 valeur, on compte 0 opérations élémentaires. Pour une liste de 2 valeurs classées en ordre décroissant, on compte 3 opérations (1 test et 2 affectations). Déterminer le nombre d’opérations élémentaires (comparaison et affectation) pour :

* 1. une liste de 3 valeurs classées dans l’ordre décroissant.

9 opérations => 4 tests, 6 affectations

* 1. une liste de 4 valeurs classées dans l’ordre décroissant.

18 opérations => 3 tests, 6 affectations + travail sur une liste de 3 valeur

* 1. une liste de 5 valeurs classées dans l’ordre décroissant.

30 opérations

* 1. une liste de n valeurs classées dans l’ordre décroissant.

## Tri par sélection

### Principe

La liste de n éléments est toujours divisée en 2 parties : une liste triée et une liste non triée.

Une image contenant capture d’écran, horloge

Description générée automatiquement

A chaque étape (itération)

* on recherche le plus petit élément de la liste non triée
* on place cet élément à la fin de la liste triée (pratiquement on échange la position du premier élément de la liste non trié et le plus petit élément trouvé)

La listenon triée contient n-i éléments, où i est le nombre d'itérations de recherche du minimum.

### Appréhension de la méthode

En prenant comme exemple la liste tab = [9, 5, 8, -2, 6, 4].

Écrire, en complétant les 2 tableaux, les étapes du tri de la liste

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Numéro de l'itération | Indice de la 1èrevaleur de la liste non triée | Valeur minimum | Indice de la valeur minimum | | Tab[indiceMin] | indiceMin | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  | | 3 |  |  |  | | 4 |  |  |  | | 5 |  |  |  | | Étapes du tri |

Combien avez-vous fait d'itérations pour trier votre liste ? 5

### Programme du tri par sélection

#### Étape 1

Écrire une fonction indice\_mintelle que:

def indice\_min(T: list, debut: int)->**int**:

"""

renvoie l’indice de la plus petite valeur dans la liste T

à partir de l’indice début.

"""

#### Étape 2

Écrire une fonction selection telle que:

def selection (T: list , debut: int) -> None:

"""

Modifie T en permutant la valeur de l’indice début et la valeur

la plus petite à partir de l’indice début dans la liste T.

"""

#### Étape 3

Écrire une fonction tri\_selectiontelle que:

def tri\_selection (T: list)-> None:

"""

Modifie T en la triant dans l’ordre croissant.

"""

### Complexité

Pour évaluer la complexité de cet algorithme on détermine le nombre de comparaisons lors de l’exécution de l’algorithme.

Le pire des cas (le nombre de comparaisons le plus élevé) correspond à une liste totalement non triée, comme celle donnée en exemple.

#### Pour commencer

Combien devez-vous faire de comparaisons pour la liste à 6 éléments [9, 5, 8, -2, 6, 4] :

* à l’itération 1: 1
* à l’itération 2 : 1
* à l’itération 3 : 1
* à l’itération 4 : 1
* à l’itération 5 : 1
* à l’itération 6 : 1

Au total combien avez-vous fait de comparaisons (itérations) ?

Comparez ce résultat à où n est le nombre d'éléments de la liste.

#### Généralisation

Combien devez-vous faire de comparaisons pour une liste de n éléments triée dans l'ordre décroissant?

* à l’itération 1 : ………………
* à l’itération 2 : ………………
* ….
* à l’itération n-1 : ………………
* à l’itération n : ………………

Le nombre de comparaisons est la somme des termes d’une suite arithmétique. Déterminer l’expression de cette somme :

#### Conclusion

### Preuve de l’algorithme de tri par sélection

#### Variant de boucle

Les boucles étant des boucles for, on connait le nombre d'itération à l'avance.

donc la boucle est bornée

donc la boucle s'arrête et le résultat attendu est atteint.

#### Invariant de boucle

L'invariant est ici le fait que la partie liste triée est toujours triée.

Initialisation

i=0, la liste triée est vide 🡪 triée

i=1 on parcourt la liste non triée pour trouver le minimum

on met le minimum trouvé à la fin de la liste triée

la liste triée contient 1 élément🡪 triée

il reste (n-1) éléments dans la liste non triée

i=2 on parcourt la liste non triée pour trouver le minimum

on met le minimum trouvé à la fin de la liste triée

la liste triée contient 2 éléments qui sont en ordre croissant 🡪 triée

il reste (n-2) éléments dans la liste non triée

Propagation

i quelconque on parcourt la liste non triée pour trouver le minimum

on met le minimum trouvé à la fin de la liste triée

la liste triée contient i éléments qui sont en ordre croissant 🡪 triée

il reste (n-i) éléments dans la liste non triée

donc la condition est vraie pour tout i

Terminaison

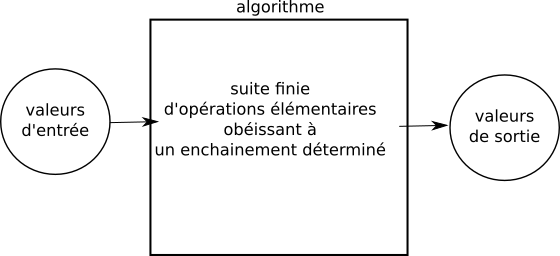
## Algorithmes et programmes

### Définition d’un algorithme

Le nom algorithme vient de Al-Khawarizmi, un mathématicien perse du IXe siècle. Il n'inventa pas les algorithmes (le plus ancien algorithme alors connu était celui d'Euclide, et on en découvrira au XXe siècle dans les anciennes tablettes de Babylone servant au calcul de l'impôt) mais décrit des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations.

Un algorithme est une liste d’instructions élémentaires ou une suite finie d’opérations élémentaires, à appliquer dans un ordre déterminé. Ces instructions fournissent, en un nombre fini d’étapes, des résultats permettant de donner la réponse à un problème.

Écrire un algorithme consiste donc à donner une méthode détaillée décrivant toutes les étapes d’une tâche à accomplir.



### Structure d’un algorithme

Un algorithme comporte 3 phases :

* Les entrées :
* Le traitement :
* Les sorties :

### Définition d’un programme (ou script)

Quand on écrit un algorithme, on utilise un langage dit "langage naturel" ("tant que", "si"...). Ce langage naturel permet de passer facilement à un langage de programmation (Python, Java...), on dit alors que l'on implémente l'algorithme.

Un programme (ou implémentation) est la traduction d’un algorithme dans un langage exécutable par une machine.

**Un exemple :** Un magasin offre une remise de 10% dès 50 € d’achat. L’algorithme et le programme en langage Python renvoie le montant à payer à partir du total des achats.

|  |  |
| --- | --- |
| Algorithme en langage naturel | Programme Python |
| Fonction Montant (*total*)  Si total > 50 alors  renvoyer *total*\* 0,9  Fin Si  renvoyer *total*  Fin Fonction | defMontant(total) :  if total > 50 :  return total \* 0,9  return total |

### Séquences d’instructions

Les instructions élémentaires d'un algorithme sont :

* l’affectation de variables ;
* la séquence d’instructions, appelée aussi bloc d’instructions, c’est-à-dire série d’actions contenant des mots clés et des connecteurs logiques nécessaires à la construction d’un algorithme.

Pour relier les différentes instructions élémentaires, on peut utiliser :

* une condition (if… elif … else)
* une boucle de répétitionbornée(for) ou conditionnelle non bornée (while).

En pratique, on rédige les algorithmes en "pseudo-code" car l’objectif est de donner les principales étapes nécessaires pour résoudre un problème, indépendamment des particularités du langage utilisé.

Les instructions d’un programme s’exécutent les unes après les autres dans l’ordre dans lequel elles ont été écrites à l’intérieur du script.

### Complexité

Une fois que le programmeur est convaincu que son algorithme est correct, il va essayer d'en évaluer l'efficacité. Il veut alors savoir si son algorithme est efficace : par exemple est ce qu’il va vite ?

Pour répondre à cette question la première idée que l’on pourrait avoir est d’implémenter l’algorithme sur un ordinateur ou des ordinateurs. Mais est-ce une bonne solution ?

Si on implémente deux algorithmes différents ayant le même objectif sur deux ordinateurs différents, celui qui a l'ordinateur le plus puissant risque de penser qu'il a l'algorithme le plus rapide, même si ce n'est pas vrai.

Afin de pouvoir comparer ou parler de l’efficacité d’un algorithme (on part quand même du principe que l’algorithme répond au problème posé) les informaticiens ont créé la notion de complexité[[1]](#footnote-1).

Pour faire simple la complexité algorithmique, c'est :

l'ordre de grandeur, en fonction de N (taille de l’entrée), du nombre d'opérations que mon programme va effectuer.

La complexité permet donc de quantifier (c’est-à-dire établir une formule mathématique) la relation entre les conditions de départ et le « temps » effectué par l'algorithme :

On compte le « nombre d’opérations élémentaires ».

L’avantage de ce critère est qu'il permet de comparer efficacement les algorithmes entre eux, indépendamment de l'ordinateur qui les exécute :Si un algorithme fait plus d'opérations que le mien, il sera plus lent chez moi, et chez mon voisin ; même si le nouvel algorithme est plus rapide chez mon voisin que l'algorithme actuel chez moi, je saurais qu'il est moins bon.

Enfin il faut savoir qu’il existe deux types de complexité :

* Complexité en temps : Réaliser un calcul de complexité en temps revient à décompter le nombre d’opérations élémentaires (affectation, calcul arithmétique ou logique, comparaison…) effectuées par l’algorithme. Pour rendre ce calcul réalisable, on émettra l'hypothèse que toutes les opérations élémentaires sont à égalité de coût. En pratique ce n'est pas tout à fait exact mais cette approximation est cependant raisonnable. On pourra donc estimer que le temps d'exécution de l'algorithme est proportionnel au nombre d’opérations élémentaires.
* Complexité en mémoire : la taille de la mémoire nécessaire pour stocker les différentes structures de données utilisées lors de l'exécution de l'algorithme

Pour comparer des algorithmes, nous allons uniquement nous intéresser à ce que l'on appelle "l'ordre de grandeur asymptotique" (notation O).

### Preuve d’un algorithme (terminologie)

#### Tests

Nous avons un algorithme qui semble fonctionner, du moins sur les exemples donnés. On est cependant en droit de se demander :

* fonctionne-t-il toujours ?
* se termine-t-il ?
* fait-il ce qui est demandé ?

On pourrait alors tester sur différentes entrées et contrôler que le résultat est conforme à nos attentes.

Ce procédé est :

* facile ;
* permet de trouver des bugs ;
* présuppose la terminaison ;
* formellement, ne garantit que les valeurs testées.

Mais on ne peut pas se satisfaire de cette démarche. Pour PROUVER qu'un algorithme est « correct », il faut DÉMONTRER qu'il est correct.

#### Terminaison

L’exécution de l’algorithme produit-elle un résultat en temps fini quelles que soient les données fournies ?

#### Correction partielle

Lorsque l’algorithme s’arrête, le résultat calculé est-il la solution cherchée quelles que soient les données fournies ?

#### Correction totale

Terminaison et correction partielle = correction totale

Quelles que soient les données fournies, l’algorithme s’arrête et donne une réponse correcte.

### Preuve d’un algorithme utilisant une boucle

Un algorithme utilisant une boucle est prouvé si :

1. On peut prouver que la boucle se termine : il existe un variant de boucle.
2. Il existe un invariant de boucle qui prouve que l’algorithme est correct à chaque itération.

#### Variant de boucle

Un variant de boucle est une quantité dépendant des paramètres de la boucle :

|  |  |
| --- | --- |
| entière ;  qui décroît strictement à chaque itération ;  qui possède une valeur minimale ou limite inférieure. | entière ;  qui croît strictement à chaque itération ;  qui possède une valeur maximale ou limite supérieure. |

Pour montrer la terminaison d’une boucle on procède ainsi : on choisit une quantité, dépendant des paramètres de la boucle, qui soit un entier et qui décroît strictement à chaque occurrence de la boucle jusqu’à une valeur minimale. Puisqu’il n’existe pas de suite infinie strictement décroissante dans N, cela prouve que la boucle se termine. Cette quantité s’appelle un variant de boucle.

#### Invariant de boucle

Un invariant de boucle est une propriété P des variables en début de boucle, qui est vérifiée à chaque passage dans la boucle.

La démonstration doit se faire en 3 étapes :

* INITIALISATION : on doit montrer que l'invariant de boucle est vrai avant la première itération de la boucle.
* PROPAGATION : on doit montrer que si l'invariant de boucle est vrai avant une itération de la boucle, il le reste après cette itération.
* TERMINAISON : une fois la boucle terminée, l'invariant fournit une propriété utile qui aide à montrer la correction de l'algorithme.

### Exercices d'application

#### Complexité

On rappelle que

1. Qu'appelle-t-on la « complexité », ou le « coût », d'un algorithme ?
   1. Le nombre d'opérations arithmétiques (additions, soustractions, ...) élémentaires effectuées lorsqu'on l'exécute.
   2. Le nombre de comparaisons effectuées lorsqu'on l'exécute.
   3. La place mémoire nécessaire pour l'exécuter.
   4. Le temps effectif nécessaire à son exécution.
   5. Toutes les propositions ci-dessus sont correctes.
2. On appelle tri par force brute un tri où l’on essaye toutes les combinaisons possibles. Par exemple pour la liste L=[3,5,1] l’algorithme (au pire cas) va regarder [3,5,1], [3,1,5], [5,1,3], [5,3,1] , [1,5,3] et [1,3,5]. Quelle est sa complexité ?

1. Le terme de « complexité » est un peu trompeur parce qu'on ne parle pas d'une difficulté de compréhension, mais d'efficacité : « complexe » ne veut pas dire « compliqué ». [↑](#footnote-ref-1)