Introduction à la complexité algorithmique

Complexité algorithmique

Introduction à la complexité algorithmique

Comment déterminer quel est l'algorithme le plus performant pour résoudre un problème donné ?

Avec quels critères ?

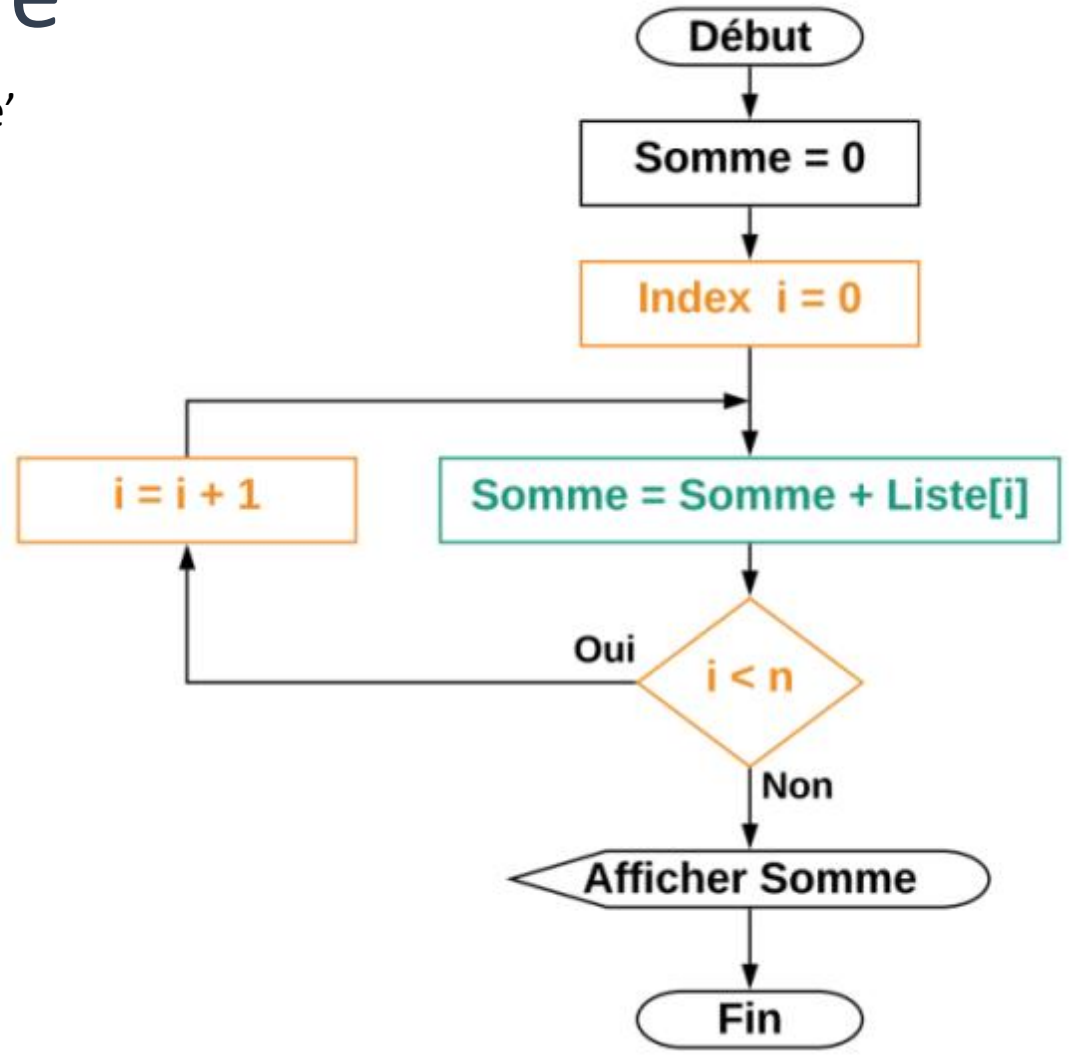
- ❑La rapidité
- ❑L'occupation mémoire
- ❑La bande passante utilisée

Un exemple pour comprendre

Calculons la somme de tous les éléments d'une liste 'Liste' de n nombres.

Voilà l'algorithme écrit en pseudo code :

```
Somme <— 0
Pour i parcourant tous les éléments de la liste
Faire
        Somme <— Somme + Liste[i]
Fin pour
Afficher Somme
```



Voilà l'algorithme sous forme d'algorigramme

Analyse de l'exemple

Quels sont les opérations présentes :

Des affectations de valeurs notées =

Des additions notées +

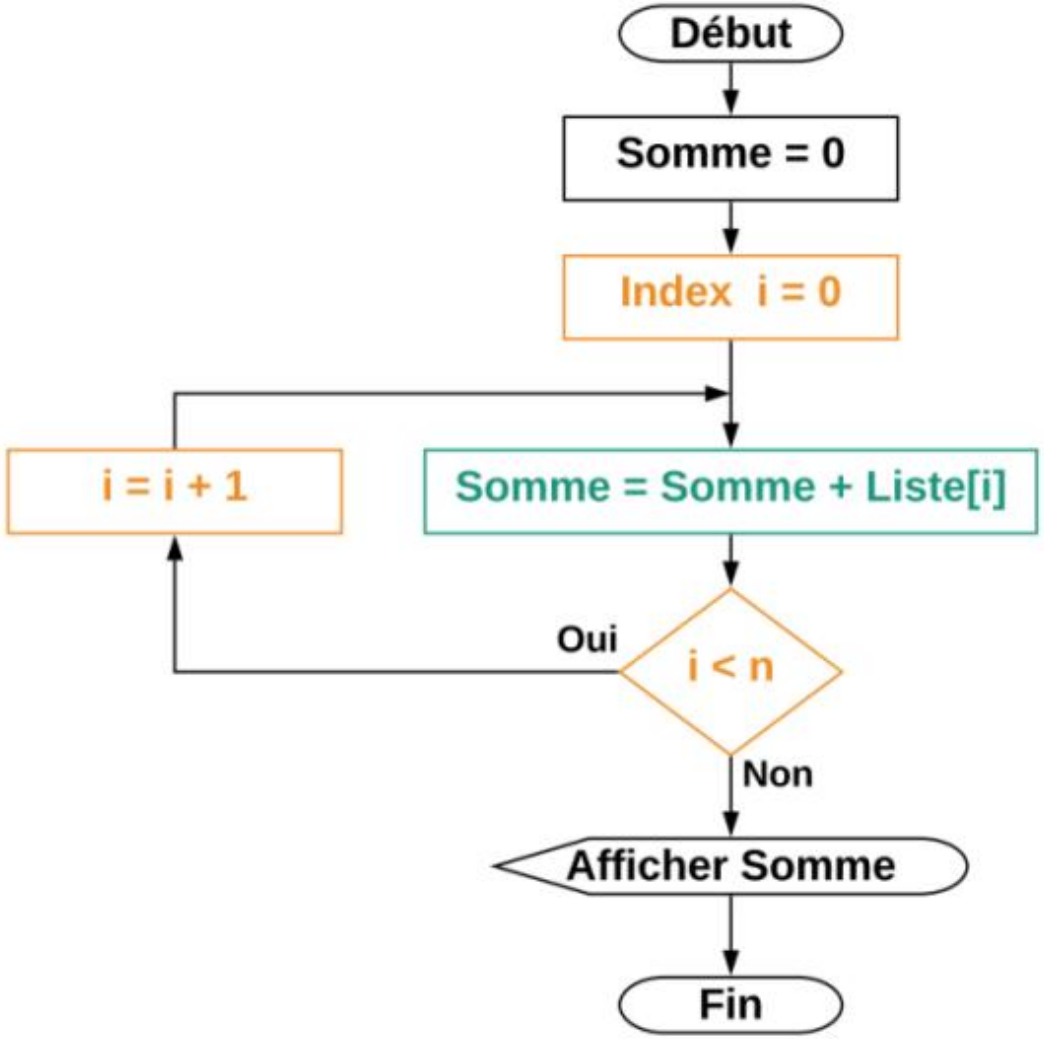
Des comparaisons ici <

Étude de l'algorithme de la boucle

La boucle est constituée de deux parties :

1) La gestion de la boucle permettant de la réaliser n fois avec :
   - l'initialisation **index i = 0**
   - Le test de fin de boucle **i < n**
   - L'incrémentation de l'index **i = i + 1**
2) Le corps de la boucle comprenant l'action effectuée à chaque tour

   somme = somme + Liste[i]



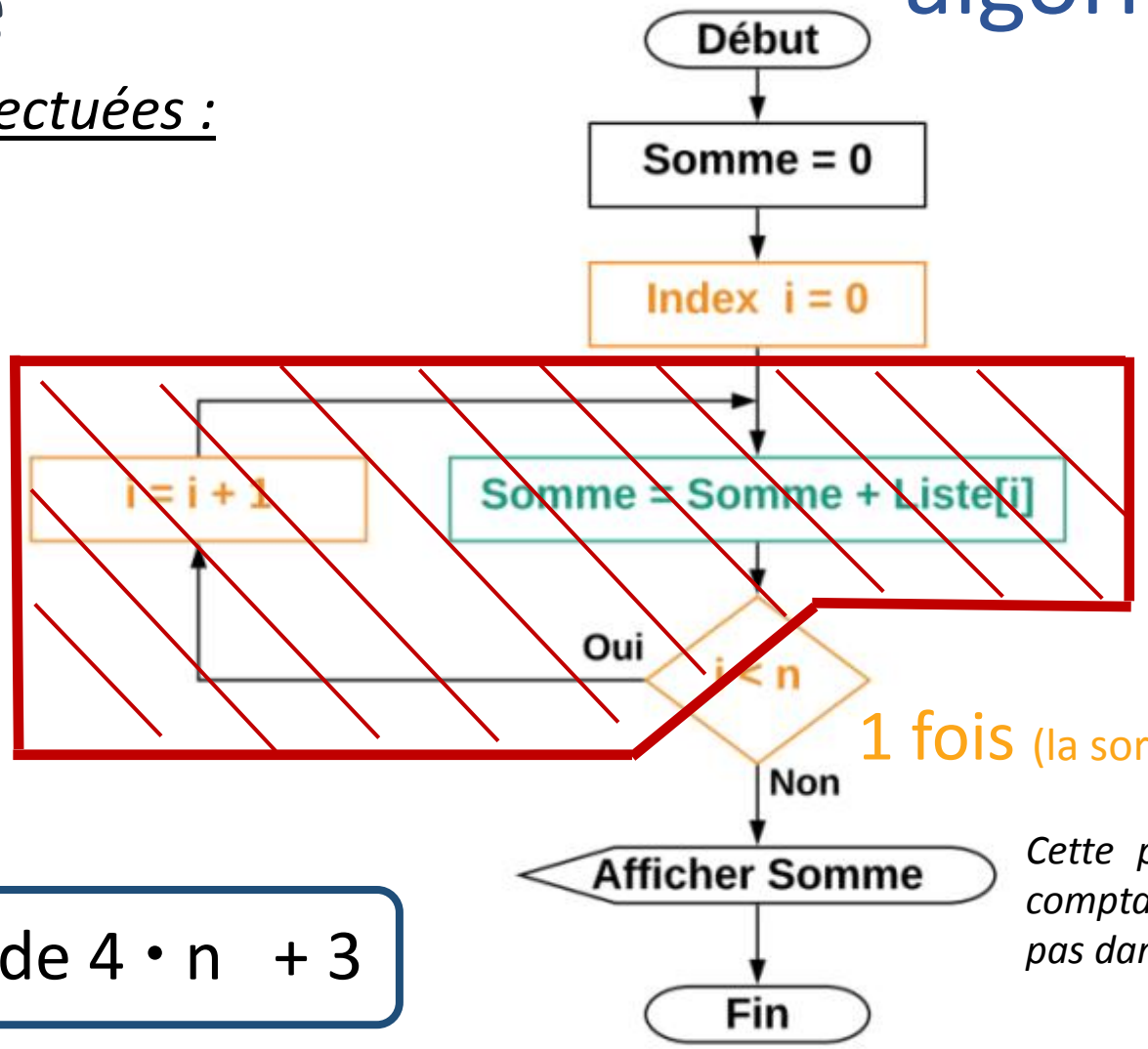
Analyse de l'exemple

Calcul du nombre d'opérations effectuées :

Pour faciliter l'analyse nous identifions les différentes parties de l'algorithme en indiquant le nombre de fois où chacune est exécutée.

Bilan du décompte :

Quantité d'instructions	Nombre d'exécutions
1	1
1	1
4	n
1	1



1 fois

1 fois

n fois

1 fois (la sortie de la boucle)

Cette partie n'est pas comptabilisée n'étant pas dans la boucle.

Le nombre d'opérations est de 4 • n + 3

Complexité de la boucle

Bilan du décompte :

Le nombre d'opérations est de $4 \cdot n + 3$

Conclusion sur la complexité de l'algorithme :

Quand le nombre de termes n devient très grand la constante est négligeable. Le nombre d'opérations est de l'ordre de grandeur de $4 \cdot n$.

Pour ces études on ne garde que la forme générale et on s'intéresse aux ordres de grandeur, donc le 4 disparaît, et on aboutit à la conclusion que cet algorithme est linéaire en n.

Autrement dis si la quantité de données n est doublée la durée de calcul est doublée également.

Notation mathématique de la complexité :

Il existe une notation mathématique pour indiquer la complexité d'un algorithme : la notation O (lire grand O).

Dans notre exemple la complexité est en : $O(n)$.

Résultats expérimentaux

Résultats obtenus pour différentes valeurs de n :

La complexité en $O(n)$ est-elle vérifiée ?

Il suffit de vérifier que la durée mesurée double bien quand n double. Plus particulièrement pour les grandes valeurs de n.

Qu'observez-vous ?

Votre conclusion ?

```
n =      100 valeurs  le temps d'éxécution :   0.000046 secondes
n =      200 valeurs  le temps d'éxécution :   0.000061 secondes
n =      400 valeurs  le temps d'éxécution :   0.000142 secondes
n =     1000 valeurs  le temps d'éxécution :   0.000332 secondes
n =     2000 valeurs  le temps d'éxécution :   0.000716 secondes
n =      400 valeurs  le temps d'éxécution :   0.000122 secondes
n =     1000 valeurs  le temps d'éxécution :   0.000335 secondes
n =     2000 valeurs  le temps d'éxécution :   0.000708 secondes
n =     4000 valeurs  le temps d'éxécution :   0.001314 secondes
n =    10000 valeurs  le temps d'éxécution :   0.003236 secondes
n =    20000 valeurs  le temps d'éxécution :   0.006700 secondes
n =    40000 valeurs  le temps d'éxécution :   0.014664 secondes
n =   100000 valeurs  le temps d'éxécution :   0.034079 secondes
n =   200000 valeurs  le temps d'éxécution :   0.068755 secondes
n =   400000 valeurs  le temps d'éxécution :   0.141120 secondes
n =  1000000 valeurs  le temps d'éxécution :   0.348514 secondes
n =  2000000 valeurs  le temps d'éxécution :   0.739935 secondes
n =  4000000 valeurs  le temps d'éxécution :   1.497242 secondes
```