FORMATION

IT - Digital - Management

m2iformation.fr

Algorithmique

Christian LISANGOLA
Software Engineer & Technical Product Manager

1.

Introduction à l’algorithmique et programmation

C’est quoi un algorithme

C’est un ensemble ordonné d’instructions/opérations finies dans le but de produire un résultat.

- Il s’agit d’un ensemble ordonné d’opérations, ce qui signifie qu’il s’agit d’une chaîne d’instructions précises qui doivent être suivies dans l’ordre. Une bonne façon de l’illustrer est avec l’exemple d’une recette de cuisine, qui reste un algorithme simple.

- Son objectif est de résoudre un problème, c'est-à-dire il a un objectif délimité. Il ne s’agit pas seulement d’écrire une belle série d’ordre qui ne mènent nulle part, mais plutôt de le faire de manière rationnelle et dans un but précis..

C’est quoi un programme

Lorsqu’un développeur crée un programme, il crée essentiellement un ensemble d’algorithmes.

Un programme informatique est un ensemble de commandes données à la machine, **écrites dans un langage spécifique**, pour effectuer une série d’opérations déterminées afin d’obtenir un résultat.

Introduction L'algorithmique

- La programmation permet de résoudre un problème de manière **automatisée** grâce à l'utilisation des algorithmes.

- Les instructions utilisées dans le programme représentent **le code source**.

- Pour que le programme puisse être exécuté par la machine, il faut utiliser un langage que celle-ci peut comprendre : **le langage binaire.**

Introduction à la programmation

Le langage binaire est le langage à 2 états représentés par des 0 et des 1, ce qui correspond à ce que l'ordinateur peut comprendre.

Lorsque le développeur écrit son programme, il le fait grâce à un langage de programmation **qui est ensemble de règles de vocabulaire et de grammaire compréhensible par un humain dans le but de donner des instructions à une machine**.

Pour que la machine arrive à comprendre ces instructions, il faut que ce code soit **traduit** en langage binaire par un **compilateur** ou **un interpréteur**.

Introduction à la programmation

Langage **bas niveau** vs langage de **haut niveau**.

- Un langage de bas niveau est un langage qui est considéré comme plus proche du langage machine (binaire) plutôt que du langage humain. Il est en général plus difficile à apprendre et à utiliser mais offre plus de possibilités d'interactions avec le hardware de la machine.

- Un langage de haut niveau est le contraire, il se rapproche plus du langage humain et est par conséquent plus facile à appréhender. Cependant les interactions se voient limiter aux fonctionnalités que le langage met à disposition.

Introduction à la programmation

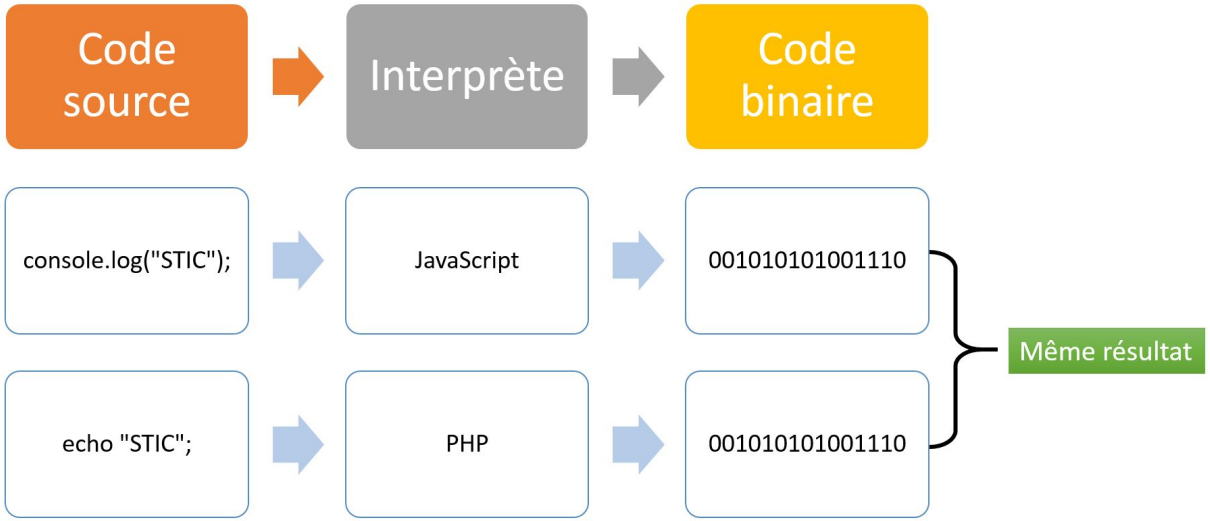
Compilation vs Interprétation

- La compilation : consiste à transformer toutes les instructions en langage machine avant que le programme puisse être exécuté. Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source.

- L'interprétation : consiste à traduire les instructions en temps réel (on run time). Dans ce cas le code source est lu à chaque exécution et par conséquent les changements apportés au code seront pris en compte directement.

Introduction à la programmation

- Alors c'est pas des 0 et des 1 le code ?



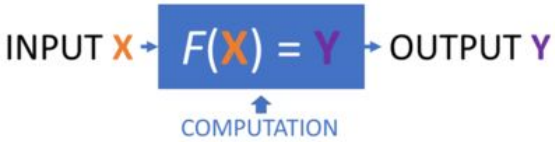
Introduction à la programmation

Et les mathématiques dans tout ça ?

- Développer ne consiste pas à écrire des formules mathématiques (sauf si vous travailler sur des logicielles destinées aux sciences évidemment).

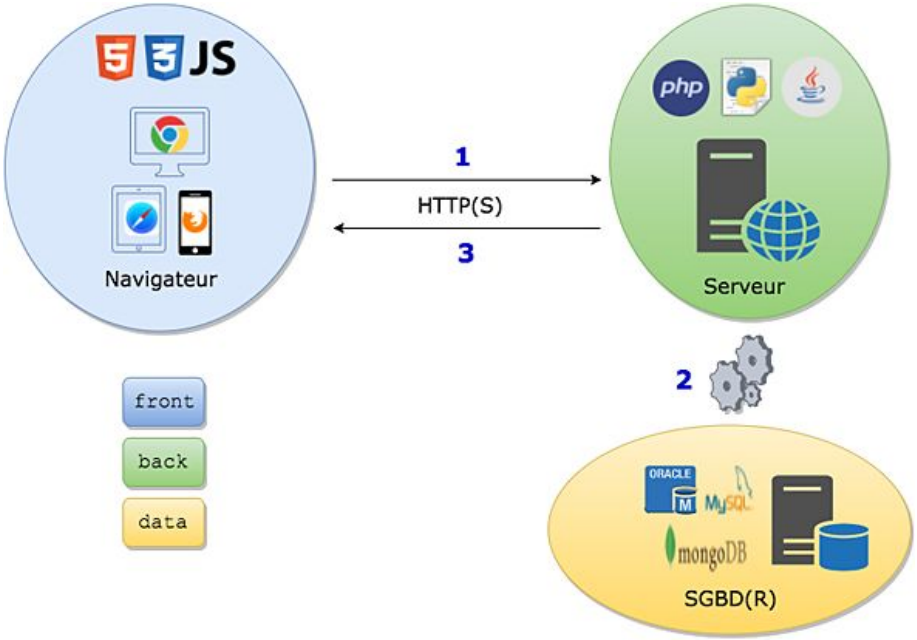
- Même si les langages sont différents, l'objectif principal d'un langage de programmation reste celui d'instruire la machine pour qu'elle produise des outputs conformes aux objectifs de l'application.

- On résume souvent par :
  - F(X) = Y, où :
    - X représente l'input
    - Y représente l'output
    - F() représente la fonction qui permet de transformer X en Y



Introduction à la programmation

- Anatomie d'une application interactive



Introduction à la programmation

Les éléments fondamentaux :

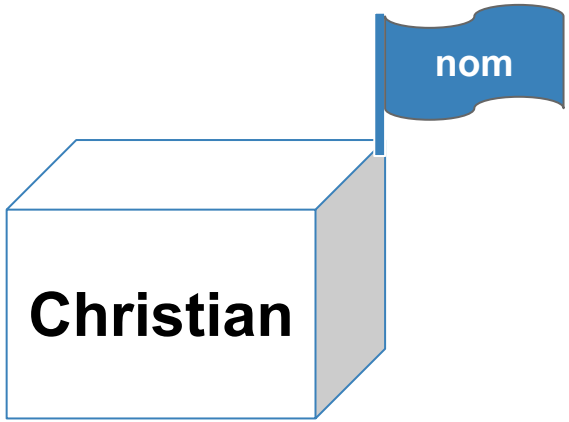
- Les variables
- Les opérateurs
- Les structures conditionnelles
- Les structures itératives
- Les fonctions
- Les tableaux (ou array)
- Les objets

Installation de Algobox

Afin de mettre en pratique les algorithmes, nous allons utiliser le logiciel algobox :

https://www.xm1math.net/algobox/download.html

Les variables



- Nom de la variable ou **identificateur** : **nom**
- Contenu : **Christian**

Une variable est une zone que l'on réserve en mémoire pour stocker des données.Par opposition aux constantes, la valeur contenue dans une variable peut changer au cours du temps.

Type de données

Si l'on doit le comparer variable à un récipient, elle peut contenir des données possédant un type bien défini.

Par exemple, on ne doit pas stocker ou garder des baskets dans une marmite.

Les types de données pouvant être contenu dans une variable sont :

- les suites de caractères (string) : elles sont utilisées pour représenter du texte ;
- les chiffres (nombre entier, à virgule flottante, etc.) : ils sont utilisés surtout avec des opérateurs mathématiques ;
- les valeurs booléennes (en anglais : booleans) : soit vrai, soit faux ;
- les tableaux (array) : ils sont utilisés pour une collection d'éléments
- les objets : ils sont des conteneurs qui peuvent inclure souvent tout type de données, y compris de sous-objets, des variables (i.e. des propriétés), ou des fonctions (i.e. méthodes)

Déclarations d’une variable

Déclarer une variable, c’est réserver un espace dans la mémoire pour y stocker des données.

```
VARIABLES
  nom EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
```

Affectation et affichage

L’affection est l’opération qui consiste à attribuer une valeur à une variable.

```
VARIABLES
  nom EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
  nom PREND_LA_VALEUR "Christian Lisangola"
  AFFICHER nom
FIN_ALGORITHME
```

Affectation et affichage

L’affection n’est pas la même chose que l’égalité, car avec l’égalité la relation est maintenue au cours du temps.

Opérateurs arithmétiques

Les opérateurs arithmétiques sont : l'addition, la soustraction, la division, la multiplication, etc.

```
VARIABLES
  nombre1 EST_DU_TYPE NOMBRE
  nombre2 EST_DU_TYPE NOMBRE
  somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  nombre1 PREND_LA_VALEUR 5
  nombre2 PREND_LA_VALEUR 10
  somme PREND_LA_VALEUR (nombre1 + nombre2)/2
  AFFICHER somme
FIN_ALGORITHME
```

Lecture des valeurs à partir du clavier

Il est possible que les valeurs des variables proviennent de la saisie de l'utilisateur.

```
VARIABLES
  nombre1 EST_DU_TYPE NOMBRE
  nombre2 EST_DU_TYPE NOMBRE
  somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE nombre1
  LIRE nombre2
  somme PREND_LA_VALEUR (nombre1 + nombre2)/2
  AFFICHER somme
FIN_ALGORITHME
```

Structures conditionnelles

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la vérification d'une ou plusieurs conditions.

La version sémantique la plus répandue des structures de contrôle est **« si… alors… »**.

Par exemple :

- Si la note est inférieure à 5, alors la note est insuffisante
- Si l'âge est supérieur ou égal à 18, alors on est majeur
- Si le mot de passe choisi contient moins de 6 caractères, alors il est trop court

Opérateurs Logiques et de comparison

Les opérateurs logiques sont : ET/AND, OU/OR, etc.

Les opérateur de comparaisons : >,<, >=,<=, <>,==

```
FONCTIONS_UTILISEES

VARIABLES
age EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE age
  SI (age>=12 ET age<=17) ALORS
    DEBUT_SI
      AFFICHER "Vous êtes ado"
    FIN_SI
FIN_ALGORITHME
```

```
FONCTIONS_UTILISEES
VARIABLES
  pays EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
LIRE pays
  SI (pays=="France" OU pays=="Belgique") ALORS
    DEBUT_SI
      AFFICHER "Vous pouvez participer au sondage"
    FIN_SI
  SINON
    DEBUT_SINON
    AFFICHER "Vous n'êtes pas éligible pour participer au sondage"
    FIN_SINON
FIN_ALGORITHME
```

Opérateurs Logiques

Les opérateurs logiques sont : ET(and), OU(or), etc.

Opérateur OU

- Retourne True quand au moins une des conditions est True
- Retourne False seulement si toutes les conditions sont false

Opérateur ET

- Retourne True seulement si toutes les conditions sont évaluées à True
- Retourne False si au moins une des conditions est évaluée à False

Les structures itératives ou boucles

Les boucles sont à la base d'un concept très utile en programmation : l'itération.

L'itération permet d'exécuter de manière répétitives des instructions. Elles peuvent être très utile par exemple pour appliquer un traitement à une liste d'éléments.

Exemple d'utilisation :

Tant que la liste n'est pas totalement parcouru :

    Modifier un élément de la liste

Ou encore :

Tant que ce chiffre ne dépasse pas 16 :

    Réalise un calcul

Boucle tant que : anatomie

Généralement, cette boucle est utilisée quand le nombre d'itérations n'est pas connu à l'avance.

```
Tant que (condition) faire
   Debut tant que
   Instruction 1
   Instruction 2
   …
   Fin tant que
```

Boucle tant que : exemple

```
FONCTIONS_UTILISEES

VARIABLES
code_secret EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
  code_secret PREND_LA_VALEUR ""
  TANT_QUE (code_secret!="xyz") FAIRE
    DEBUT_TANT_QUE
      LIRE code_secret
      SI (code_secret!="xyz") ALORS
        DEBUT_SI
          AFFICHER "Code incorrect, veuillez ressayer"
        FIN_SI
        SINON
            DEBUT_SINON
              AFFICHER "Félicitations,vous avez trouvé le code secret"
            FIN_SINON
    FIN_TANT_QUE
FIN_ALGORITHME
```

Boucle pour : anatomie

Cette boucle est utilisée quand le nombre d'itérations est est connu à l'avance.

```
Pour compteur allant de départ à limite
   Debut pour
   Instruction 1
   Instruction 2
   …
   Fin pour
```

Boucle pour i...

```
VARIABLES
nbr EST_DU_TYPE NOMBRE
compteur EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
LIRE nbr
POUR compteur ALLANT_DE 0 A 12
  DEBUT_POUR
  AFFICHER nbr
  AFFICHER " x "
  AFFICHER compteur
  AFFICHER " = "
  AFFICHERCALCUL* nbr*compteur
  FIN_POUR
FIN_ALGORITHME
```

Manipulation des chaines

- **nomChaine.substr(positionDebut, positionFin)** : Retourne une sous-chaîne
- **nomChaine.length** : Taille de la chaîne
- **nomChaine.charCodeAt(pos)** : permet d'obtenir le nombre égal au code ascii de la lettre figurant à la position
- **nomChaine.toString()** : Transforme un nombre en chaine

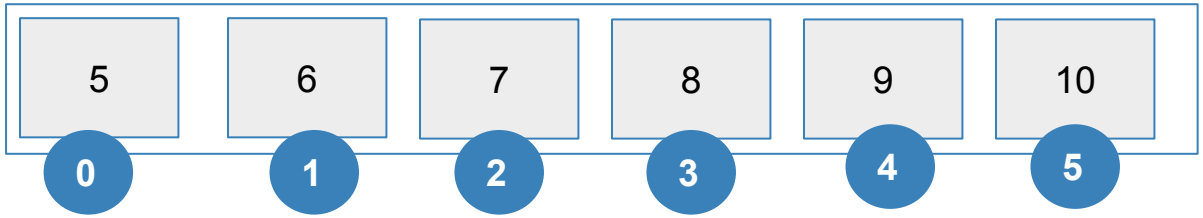
Les tableaux

Les tableaux (array) sont des listes indexées d'éléments qui partagent normalement une certaine relation sémantique pour appartenir à la liste. Un exemple tout simple d'array est la liste des courses : on peut indexer cette liste en fonction de l'ordre des articles à acheter :

1. Lait
2. Farine
3. Pommes
4. Fromage

Attention : l'index d'un tableau commence à 0. En prenant l'exemple précédent on dira que l'élément "Lait" est dans la première case du tableau qui est à l'index 0.
De la même façon l'élément "Fromage" est l'élément dans la 4ème case du tableau qui se trouve à l'index 3.

Les tableaux : index



Les tableaux

```
FONCTIONS_UTILISEES

VARIABLES
nombres EST_DU_TYPE LISTE
n EST_DU_TYPE NOMBRE
i EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME
LIRE n
POUR i ALLANT_DE 0 A n - 1
  DEBUT_POUR
  LIRE nombres[i]
  FIN_POUR
POUR i ALLANT_DE 0 A n - 1
  DEBUT_POUR
  AFFICHER nombres[i]
  FIN_POUR

FIN_ALGORITHME
```

Les fonctions

Les fonctions représentent une sorte de "programme dans le programme",donc des des “sous programmes”.On utilise des fonctions pour regrouper des instructions et les appeler sur demande : chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions. Pour accomplir ce rôle, le cycle de vie d'une fonction se divise en deux :

1. Une phase unique dans laquelle la fonction est déclarée
   On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité.

2. Une phase qui peut être répétée une ou plusieurs fois.

Les fonctions dans la vie courante

Il existe plusieurs objets de tous les jours, qui peuvent nous de bien visualiser le comportement des fonctions

- Hachoir à viande : Reçoit un input(de la viande), puis produit de la viande hachée

- Le moule : Reçoit du ciment , puis produit des risques.

Les fonctions

```
FONCTIONS_UTILISEES
FONCTION addition(a,b)
VARIABLES_FONCTION
  DEBUT_FONCTION
    RENVOYER a+b
  FIN_FONCTION
VARIABLES
a EST_DU_TYPE NOMBRE
b EST_DU_TYPE NOMBRE
somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
LIRE a
lire b
somme PREND_LA_VALEUR addition(a,b)
AFFICHER somme

FIN_ALGORITHME
```

Complexité algorithmique

La complexité algorithmique nous permet de mesurer l'efficacité d'un algorithme.

S'il faut mesurer l'efficacité ou performance d'un algorithme en se basant sur son temps d'exécution, alors les temps d'exécutions seront différents selon les ordinateur en fonction de la configuration matérielle.

Donc, utiliser le temps d'exécution comme metric ne peut pas donner une idée claire sur les performance d'un algorithme.

Pour avoir un standard qui soit indépendant de la configuration, on se base sur la Big O notation(Notation grand O).

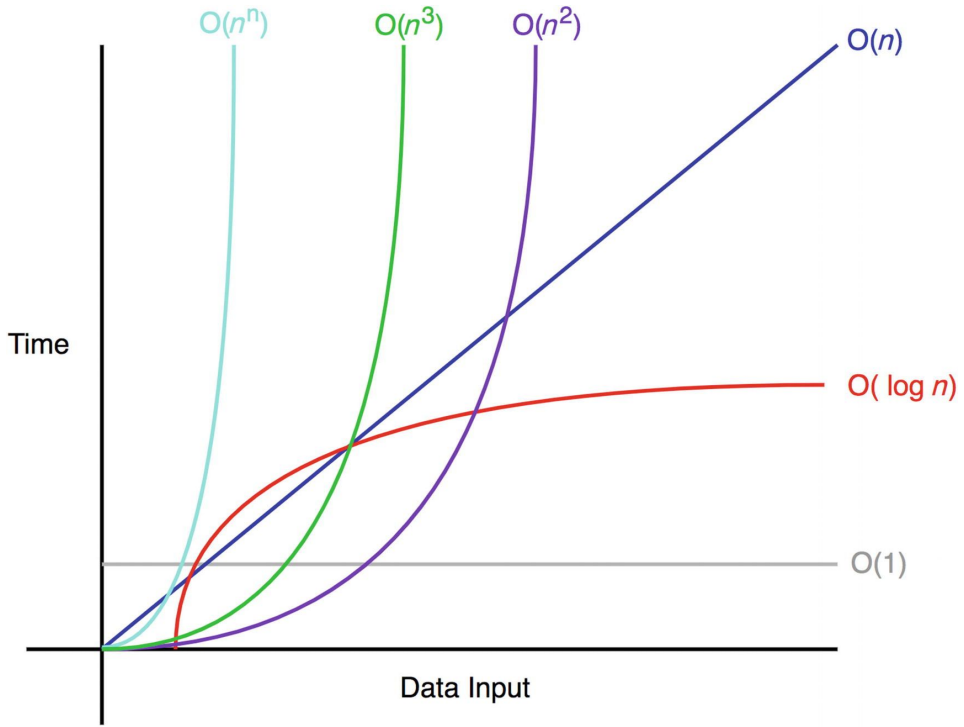
Big O

La Big O, nous permet d'évaluer les performances d'un algorithme sur base de :

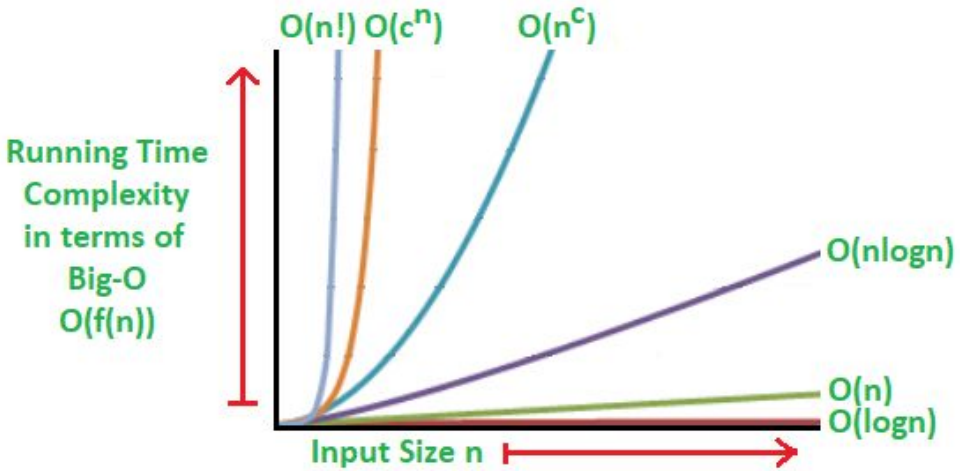
- Runtime complexity : Nombre d'opération exécutées par le CPU
- Space complexity : Quantité de mémoire utilisé

L'expression utilisée sera O(n) où n constitue le nombre d'opérations pour le runtime complexity et le space complexity.

Big O



Big O



O(n!),O(c^n),O(n^c) - Worst

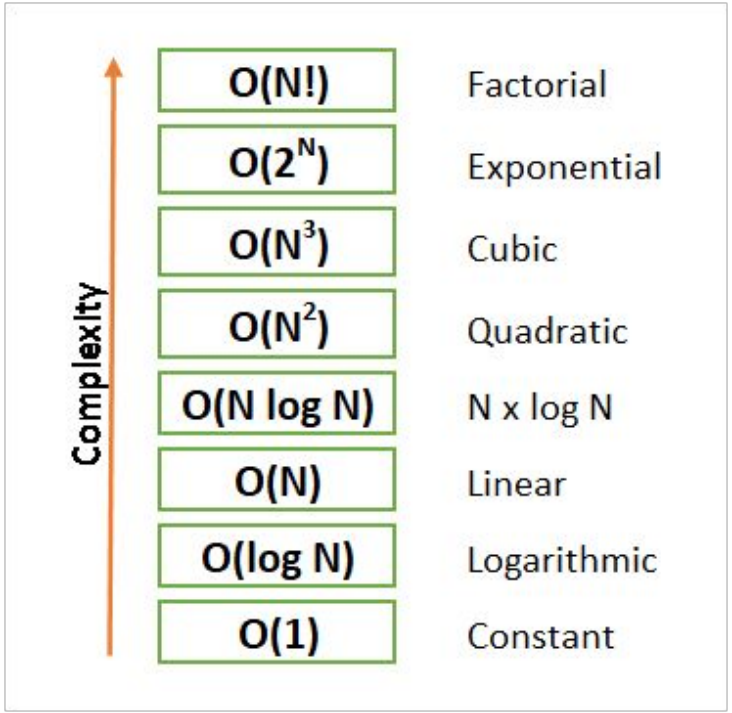
O(nlogn) - Bad

O(n) - Fair

O(logn) - Good

O(1) - Best

Big O


Complexity
O(N!)
Factorial
O(2^N)
Exponential
O(N^3)
Cubic
O(N^2)
Quadratic
O(N log N)
N x log N
O(N)
Linear
O(log N)
Logarithmic
O(1)
Constant

Big O

- Calcul de la somme des éléments du tableau : O(n)
- Calcul de la somme de 2 nombres : O(1)
- Calcul de de la somme des diagonales d'une matrice???
- Retirer un valeur au début d'un tableau : O(n)
- Retirer un valeur à la fin du tableau : O(1)
- Fibonacci(recursif) sans mémorisation : 2^n
- Fibonacci(recursif) avec mémorisation : O(n)
- Tri à bulles : $O(n^2)$