NB: on va étudier cette notion parce que le but de ce cours est de former des développeurs informatique professionnel.

Et c'est en comprenant cette notion de complexité algorithmique qu'on va être capable d'écrire des algorithmique et les perfectionnés pour les rendre

Plus performant (plus rapide, moins gourmand en espace mémoire) ...

## Complexité algorithmique

* Le but de l'étude de la complexité algorithmique est de pouvoir comparer des algorithmes qui résous le même problème...

Ceci pour savoir quel algorithme est plus performant, quel algorithme est le mieux approprié selon l'environnement

* pour comparer 2 algorithmes qui résout le même problème, le premier reflex est toujours d'exécuter les 2 algorithme et de comparer

Le temps que met chacun pour terminer la tâche.

Ce qui ne fonctionne pas vraiment vu que plusieurs paramètres entre en compte:

@ le langage de programmation utiliser pour implémenter ces algorithmes

@ le processeur de l'appareil utilisé

@ même pour un même appareil, même langage, et même compilateur, il y'aura toujours une dépendance vis a vis de la

Disponibilité du processus qui exécute ces algorithmes

Avec tous cela, nous comprenons que pour comparer 2 algorithmes, on doit utiliser d'autre mécanisme

* il existe 2 types de complexité algorithme:

@ Complexité en espace

La complexité en espace est la taille de la mémoire nécessaire pour l'exécution de l'algorithme.

@ Complexité en temps:

Réaliser un calcul de complexité en temps revient à décompter le nombre d'opérations élémentaires (comme celles détaillé dans le précèdent Cours sur l'algorithmique) effectuées par l'algorithme.

Cependant, il y'a des algorithmes dont la complexité peut varier selon les données utiliser lors de l'implémentation. Néanmoins, on distingue 3 cas de Figures:

. La complexité dans le meilleur des cas : situation la plus favorable.

Exemple: trier d'un tableau trié

Recherche dans un tableau d'un élément situer à la première position

. La complexité dans le pire des cas : situation la plus défavorable. (On va généralement considérer le pire des cas)

Exemple: trier d'un tableau préalablement trié dans l'ordre inverse

Recherche dans un tableau d'un élément qui n'y est pas

. La complexité en moyenne

* exemple:

ex1: algorithme qui permet de permuter les valeurs de 2 variables

```
val1 : Entier
val2 : Entier
val3 : Entier
val1 = 5;
val2 = 10;
algo1:
        val3 = val1
        val1 = val2
        val2 = val3
algo2:
        val1 = val1 + val2
        val2 = val1 - val2
        val1 = val1 - val2
```

algo1...complexite en temps T=3, en memoire M=3(Entier)

algo2...complexite en temps T=6, en memoire M=2(Entier)

/*a vous de juger*/

ex2: algorithme qui recherche un element dans un tableau

```
n : Entier
T : TABLEAU [0...n] : Entier
x : Entier
i : Entier
resultat : Boolean
resultat = false
algo:
        pour i allant de 0 a n faire
                si (x==T[i])
```

resultat = true

finsi

finpour

algo...complexité en temps T(n) = 4n

/*car a la première line on:

Pour i allant de 0 à n faire <=> i=i+1, i<=n (ce qui fait 3 instruction élémentaire).

Et a l'intérieur de la boucle on a x==T[i] (ce qui est une autre instruction élémentaire)

D'où au final on a 4 instructions élémentaires par tour de boucle. Or on boucle n fois dans le pire des cas*/

Ainsi, vous pouvez maintenant réfléchir sur un autre algorithme donc la complexité en temps ou en espace sera réduit, selon vos exigences

Pour mieux approfondir ces connaissances dans la complexité algorithmique regarder ce cours