

Stratégies de Résolution de problèmes

ECL - 2A - MI

2021-2022

Chapitre 1 (transparents)

Analyse d'algorithmes

Éléments de calcul de complexité

Alexandre Saidi (ECL-LIRIS)

I- Quelques références bibliographiques

1. *Optimisation Combinatoire* : M. Sakarovitch, Hermann, 1984
2. *Design and Analysis of Algorithms : cours notes* : Samir Khuller, 1994
3. *Graphes & Algorithmes* : M. Gondran & M.Minoux, Eyrolles, 1995
4. *Foundation of Algorithms* : R. Neapolitan, K. Naimipour, D.C. Heath & Co.1996
5. *The algorithm design Manual* : S.S. Skiena, Springer Verlag, 1997
6. *Complexity of Algorithms* : Peter Gacs, LNCS 1999
7. *The Art of Computer Programming* : 2nd edition, Knuth, 1999
8. *Data structures & Algorithm Analysis in C++* : M.A; Weiss, Addison-Wesley, 1999
9. *The algorithm design* : G. Kleinberg & E. Tardos, Addison Wesley 2005
10. *The Standard Template Library*, Alexander Stepanov (le site du SGI)
11. *Algorithm + Data structure = Programs*, N. Wirth, Prentice Hall 1976
12. *The Design and Analysis of Computer Algorithms*, Aho, Hopcroft, Ullman, 1974 Ad.w.
13. Notes personnelles "*Cahier de Complexité*" (A.S.) 1995-2005

II- Objectifs du cours et éléments abordés

Rigueur dans les algorithmes (preuve)

Calcul du coût (*complexité*) des algorithmes

On entend souvent : **" Mon programme marche donc il est juste !"**

⇒ Pouvez-vous le prouver ?

La Connaissance de quelques méthodes importantes aident.

Éléments abordés dans ce module :

- Sensibilisation aux exigences des (bons !) algorithmes
- Analyse, preuve, calcul de **Complexité** des algorithmes, outil de mesure de complexité
- Différents outils et stratégies de résolution de problèmes :
 - ✓ Stratégies de résolution : Diviser / séparer pour Régner, Gloutonne, Programmation Dynamique, B&B, A*, etc.
 - ✓ Graphes et algorithmes remarquables, TAS & Arbres, Dijkstra, MST, etc.
 - ✓ Récursivité & analyse récursive
- A travers l'Étude de quelques problèmes combinatoires connus.

III- Pourquoi faut-il des algorithmes ?

- Certains problèmes (p. ex. TSP, tournée de véhicules, ...) n'ont pas de formulation Mathématique.
- Ils Peuvent avoir une 'Modélisation Mathématique (p.ex. **Coloration minimale**),
- Pour d'autres, on n'a pas de 'formule' pour calculer une sortie / décision / etc.
 - Il faut les résoudre par un algorithme (+ heuristiques dans certains cas)
- Cet algorithme va décrire la démarche de résolution du problème.
- Il doit satisfaire quelques conditions (pour être 'utile' et 'effectif').
- Il décrira le **QUOI** (données, méthode) : **La Logique**
et le **COMMENT** (description des étapes) : **Le Contrôle**

IV- Qu'est-ce qu'un algorithme ? Exemples

Exemple : trier une séquence de N entiers $S[1..N]$ dans l'ordre ascendant.

→ La solution pour S de taille 6 = $[10, 7, 11, 5, 13, 8] \rightarrow [5, 7, 8, 10, 11, 13]$

→ S est une **instance** (1 exemplaire) du problème traité par un algorithme de tri.

Un **algorithme** de TRI décrit la méthode sous forme d'une suite d'instructions précises exécutée pour S de taille N .

Parmi une pléthore de méthodes, on dira par exemple (*Tri Insertion*):

- Soit S_i , $i=1..N$ un élément de S , commencer à partir de S_1 jusqu'à $S_{(N-1)}$
 - Itérer en comparant S_i aux S_j , $j=i+1..N$ et permuter S_i et S_j si $S_i > S_j$.
 - Recommencer avec $S_2 \dots S_N$
- Notion d'**assertion/invariant** : ici, après une itération sur i : $S_i = \min(S_{i..N})$

IV.1- Exemple : Tours de HANOI



- Un **algorithme** pour HANOI décrit la méthode de déplacement de N disques :
--> Les N disques au départ sur le pieu 'A' doivent aller sur B en s'aidant de C

Exemple d'algorithme de solution : $Hanoi(N, Dep, Arr, Aux)$

- Commencer par déplacer N-1 disques depuis Dep vers Aux en s'aidant de Arr
- Il restera un disque sur Dep : déplacez-le sur Arr
- Recommencer : déplacer les N-1 disques restants de Aux vers Arr à l'aide de Dep

Appel initial : $Hanoi(N, Dep=A, Arr = B, Aux = C)$

IV.2- Exemple : mariages stables (stable matching)

Construction / vérification d'un couplage stable dans un graphe.

Un couplage est *instable* s'il contient 2 personnes A et B qu'on n'a pas connectées (ensemble) alors qu'ils se préféreraient (p/r aux conjoints que l'on leur a affectés).

P. Exemple, on a 'fêté' ces mariages (f,F,g,G):

F est mariée avec g

G est mariée avec f

Mais 'ça' ne tiendra pas (instable) car :

F préfère G à g

G préfère F à f

Questions:

- Comment vérifier qu'un couplage est stable?
- Un couplage stable existe-il toujours? Si oui, peut-on le trouver (par un algo?)

👉 Voir solutions et applications en Annexes. [Mariages stables \(stable matching\)](#)

✓- Ce que l'on peut attendre des algorithmes

- Un algorithme a un domaine de définition :

l'ensemble potentiellement infini des instances (In/Out) du problème.

- Un algorithme doit être **juste** (et au besoin complet)

Un algorithme juste doit fonctionner sur toutes les instances d'un problème.

--> Par Ex. : un algorithme de recherche d'un élément dans un tableau

- Un algorithme **complet** doit trouver toutes les solutions justes.

- Un algorithme doit "**finir**" (terminaison) en un temps fini.

- Selon les cas, on pourra apporter les preuves de :

Existence / Unicité (éventuelle) de la solution, un Délai donné, Etc...

VI- Autres remarques

Algorithme = Logique + Contrôle (une logique + une stratégie)

Programme = Algorithme + Structures de données

Un algorithme décrit une solution dans un espace d'états (initiaux / finaux).

- ☞ Un état final = une solution
- ☞ Le **contrôle** dit : comment aller d'**initial** au **final** (étant donné la Logique).
- ☞ Parfois, une **logique juste** ne donne pas forcément un algorithme :

$$\text{Exemple : } (n + 1)! = (n + 1) \cdot n! \text{ d'où } n! = \frac{(n+1)!}{n+1}$$

Même si le "Contrôle" de l'algo. du calcul de ce $n!$ respecte sa logique.

Autres :

- Notion de décidabilité / *calculabilité* (traçabilité, en anglais : *tractable*)

☞ *Ex. : un programme pour détecter si un programme boucle ?*

- **Itératif / Récursif** : indépendant de la complexité, transformable

☞ Souvent on a le choix entre récursif/itératif, parfois pas (e.g. Hanoi, ...).

VII- Idée de la complexité

- Il y a souvent plusieurs méthodes (algorithmes) pour résoudre un problème.

Dans **Logique + Contrôle** : plusieurs Contrôles possibles pour une même logique.

Exemple de TRI de S : **on peut permuter S jusqu'à tomber juste !!**

Un Exemple (trivial):

On a 15 boîtes de vis de différentes longueurs rangées dans un bloc de rangement

avec 15 tiroirs.

→ **Comment faut-il ranger ces vis afin de les retrouver facilement ?**

1- Naïve : ranger n'importe comment, rechercher de gauche à droite.

--> En moyenne : 8

1'-Variante : ranger les vis selon la fréquence des demandes.

→ tjs 8 en moyenne.

2- Naïve 'Las Vegas' : Naïve mais plus *démocratique*

Recherche aléatoire (au lieu de gauche-droite) → tjs 8 en moyenne.

3- Tri : on range les vis selon la longueur de la plus petite à la plus grande.

On cherche par la méthode **Dichotomique**.

→ En moyenne 3,26 comparaison

 Voir détails de cet exemple en annexes [Détails Ex. "Idée de la complexité"](#)

VIII- Complexité et Contrôle

Définition : la **complexité** d'un algorithme A est une définition $C_A(N)$ donnant le nombre d'instructions **caractéristiques** exécutées par A (dans le pire des cas, pour une donnée de taille N).

Cette mesure donne un ordre de croissance de l'algorithme A .

N.B. la fonction de complexité d'un algorithme en temps est noté **$T(N)$** .

- L'analyse de la complexité peut être

Pessimiste (le cas pire : $W(n)$) : $\rightarrow T_{\max}(n) = \max(\text{Temps}(d) \mid d \text{ donnée de taille } n)$

Optimiste (meilleur cas : $B(n)$) : $\rightarrow T_{\min}(n) = \min(\text{Temps}(d) \mid d \text{ donnée de taille } n)$

Moyenne ($A(N)$) $\rightarrow T_{\text{moy}}(n) = \sum p(d) * \text{Temps}(d) \mid p(d) : \text{proba de } d$

- La connaissance du **cas pire** est **critique** pour les applications **Temps Réels** (contrôle aérien, robots, automatismes, freinage, systèmes d'alarme, etc.) où il faut **borner le temps nécessaire aux calculs**.

N.B. : accompagné de la complexité du cas pire, la **complexité moyenne** (\neq la moyenne des complexités) donne une indication intéressante.

N.B. : Problème **Q-sort** (Tri rapide)

Dans Q-sort, le cas *favorable* est $N \log(N)$ mais le cas pire est $O(N^2)$.

--> On doit considérer quand-même le **cas pire** (e.g. si le tableau est déjà trié).

VIII.1- Exemple 1 : Comparatif recherche d'une "vis"

- Recherche séquentielle ou Dichotomique dans l'ensemble S (de vis) :

Taille de S	Recherche séquentielle	Recherche binaire
128 (2^7)	128	8
1024 (2^{10})	1024	11
1.048576 (2^{20})	1048576	21
4.294.967.296 (2^{32})	4.294.967.296	33

L'efficacité d'un algorithme de recherche binaire semble évidente.

VIII.2- Exemple 2 : séquence de Fibonacci

$$\begin{aligned}\text{fib}(0) &= 0, \\ \text{fib}(1) &= 1, \\ \text{fib}(n) &= \text{fib}(n-1) + \text{fib}(n-2)\end{aligned}$$

1 - La solution réursive naïve (cf. la déf.) avec beaucoup de calculs redondants.

Le nombre de termes calculés $T(n)$ est de l'ordre de $2^{n/2}$ (voir plus loin).

2- il existe une solution réursive (de complexité $O(2^n)$, voir + loin) :

$$\text{fib}(2n) = \text{fib}(n)^2 + 2\text{fib}(n).\text{fib}(n-1) \quad \text{cas pair}$$

$$\text{fib}(2n+1) = \text{fib}(n)^2 + \text{fib}(n+1)^2 \quad \text{cas impair}$$

3- La solution utilisant un tableau pour stocker (Pr. D) :

Le coût $T(n) = n+1$

VIII.3- Exemple 3 : Comparatif Fibonacci

Comparatif du **temps d'exécution** de Fib sur une machine qui calcule chaque terme en une nanoseconde (10^{-9} sec.) pour les deux méthodes précédentes

n	$2^{n/2}$	Temps Algorithme Linéaire (n+1 termes)	Temps Algorithme Récursif (naïf, non Pr. D.)
60	$1.1 * 10^9$	61 ns	1 s
80	$1.1 * 10^{12}$	81 ns	18 min.
100	$1.1 * 10^{15}$	101 ns	13 jours
120	$1.2 * 10^{18}$	121 ns	36 années
200	$1.3 * 10^{30}$	201 ns	$4 * 10^{13}$ années

- ☞ Remarque : dans ce cours, **on ne traitera pas la complexité en espace.**
- ☞ **Certaines stratégies** peuvent mettre à mal les capacités (ressources, temps, espace) des machines récentes :

Exemples :

fib(N),

prime(p) : $(2^{p-1} \% p == 1)$ pour $p > 2$,

Tri par **permutation** successives !,

$$\mathbf{n!} = \frac{(n+1)!}{n+1}$$

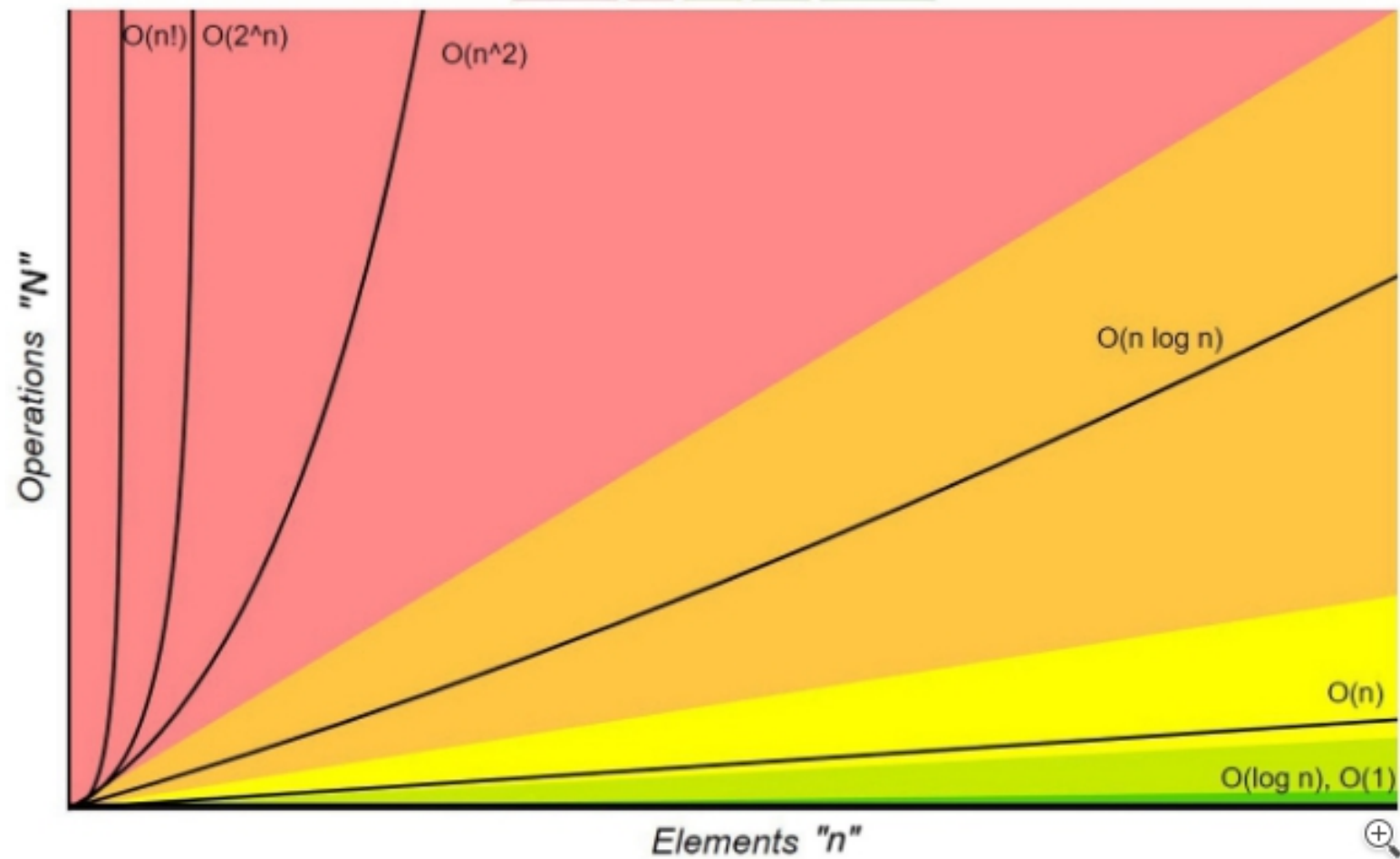
...

IX- Tableau des croissances relatives

Dans l'ordre croissant avec des exemples :

c	<i>constante</i>	→ accès à un élément dans un tableau
$\log N$	<i>logarithmique</i>	→ couper un ensemble en 2 parties égales, recouper
\sqrt{N}		→ utilisation dans le calcul des nombres premiers
$\log^2 N$	<i>log. au carré</i>	→ recherche dans un B-arbre, dans une forêt
N	<i>linéaire</i>	→ parcours linéaire d'un ensemble de données
$N \cdot \log N$		→ couper un ens. en 2 + parcours de chaque partie
N^2	<i>quadratique</i>	→ parcourir un ensemble une fois par élément d'un autre ensemble de la même taille (cf. tri bulle)
N^3	<i>cubique</i>	→ triple boucle (voir exemple réf. des sommes)
2^N	<i>exponentiel</i>	→ générer tous les sous-ens. d'un ens. de données
$N!$	<i>exponentiel</i>	→ toutes les permutations d'un ens. (ex. tri bête !)

IX.1- Comparaisons des courbes des croissances usuelles



IX.2- Comparaison de qq. ordres de de complexité

--> Le temps de calcul suppose une microseconde par instruction de haut niveau

(i.e. pas en assembleur) sur une machine avec un processeur $\geq 386/486$.

Les cases vides : > 1000 milliards d'années (> l'age estimé de l'univers).

complexité \ taille	20	50	100	200	500	1000
$10^3 \cdot n$	0.02 s	0.05 s	0.1 s	0.2 s	0.5 s	1 s
$10^3 \cdot n \log_2 n$	0.9 s	0.3 s	0.6 s	1.5 s	4.5 s	10 s
$100 n^2$	0.04 s	0.25 s	1 s	4 s	25 s	2 mn
$10 n^3$	0.02 s	1 s	10 s	1 mn	21 m	27 h
$n^{\log n}$	0.4 s	1.1 h	220 j	12500 ans	$5 \cdot 10^{10}$ ans	
$n^{n/3}$	0.001 s	0.1 s	2.7 h	$3 \cdot 10^6$ ans		
2^n	1 s	36 ans				
3^n	58 m	$2 \cdot 10^{11}$ ans				
$n!$	77100 ans					

N.B. : pour se faire une idée, lancer le calcul récuratif de **fib(100)** --> $O(5/3)^{100}$

X- Éléments d'analyse de la complexité

- Comment un algorithme se comporte et quelles ressources (temps et espace) il demande.
- Sous quelle forme le temps d'exécution augmente si la taille des données en entrée augmente.
- Cette analyse doit être **indépendante** des aspects d'un **contexte** particulier.
- Dans la **complexité** d'un algorithme, on :
 - > ne calcule pas le nombre de cycles du CPU (dép. d'une machine particulière).
 - > ne compte pas le nombre d'instructions exécutées dépendant d'un langage.
 - > ne tient pas compte de la classe du langage (impérative / fonctionnelle / logique) ni du compilateur employé. \Rightarrow
- **Mais** on décide en général d'une **opération de base caractéristique**.

--> L'opération de base peut changer d'un algorithme à un autre.

Exemples :

- l'ajout d'un élément à un tableau pour *fib* linéaire : $T(n)=n$
 - la comparaison de deux éléments pour le *tri* par échange : $T(n) = n.(n-1)/2$
 - la multiplication de 2 valeurs pour la *multiplication de matrices* : $T(n) = n^3$
- On peut rendre un algorithme plus efficace (par une cst.) sans forcément modifier sa complexité (Exemple BE1, AES et la fonction *prometteur*).
 - La complexité notée $T(n)$ doit tenir compte de tous les cas possibles
(*Every-case complexity* ≠ *Average-case complexity*).

XI- Sensibilité à la puissance des machines

Principe d'invariance (*opinionem dissimilis*) :

malgré les différences technologiques, la complexité d'un même algorithme sur deux machines différentes ne varie que par un facteur constant.

Exemple (ce n'est pas ce que l'on croit !)

Soit T = le temps nécessaire pour exécuter un programme sur une machine $M1$.

Supposons disposer du même temps T pour exécuter le même programme sur une machine $M2$ 10 fois plus rapide que $M1$.

--> De quel facteur (p/r à $M1$) peut-on augmenter la taille des données traitées sur $M2$?

Question légitime : la complexité dépend de la taille des données (n).

● Soit n la taille des données sur M1, n' celle sur M2 pour la même durée T .

- Pour une complexité linéaire, on a $n' = 10n$ (10 fois plus de données)

- Pour une complexité en n^2 , on a $n'^2 = 10n^2$

$$\text{d'où } n' = \sqrt{10}n = 3,16n$$

- Pour une complexité 2^n , on a $2^{n'} = 10 \cdot 2^n$

$$\text{d'où } n' = n + \log 10 = n + 3,3$$

👉 Pour ce cas, on peut augmenter les données de seulement 3 !!

Détaillons :

- l'évolution de n pour M2 (pour le même temps t)
- l'évolution du temps t si $n \rightarrow 10n$ (sur la même M1)

Complexités	1	$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n
Évolution du temps t quand la taille des données $n \rightarrow 10n$	t	$\log(10n) = t+3,3$	$10 t$	$(10+\varepsilon) t$	$10^2 t$	$10^3 t$	t^{10}
Évolution de la taille n quand le temps alloué $t \rightarrow 10t$	∞	n^{10}	$10 n$	$(10-\varepsilon)n$	$n\sqrt{10}=3,16 n$	$2,15 n$	$n+3,3$

La valeur de ε est négligeable devant la croissance de la fonction.

- dans la complexité $n \log(n)$, lorsque $n \rightarrow 10n$, on aura :

$$(10n)\log(10n)=10n[\log(n)+3,3]=10n.\log(n)+33n$$

$$=n\log(n)[10+ 33/\log(n)]=n.\log(n)[10+\varepsilon] = t (10+\varepsilon)$$

- dans $\log(n)$, lorsque $t \rightarrow 10t$, on aura : $\log(n) \rightarrow 10 \log(n) = \log(n^{10}) \rightarrow n' = n^{10}$
- dans $n \log(n)$, lorsque $t \rightarrow 10t$, on aura : $n.\log(n) \rightarrow 10 n.\log(n)$
 $= 10 n.\log[10 n / 10] = 10n\log(10n) -10n*3,3=10n\log(10n) -\varepsilon(n.\log(n))= (10-\varepsilon)t$
- le cas de la complexité n^2 a été traité plus haut...

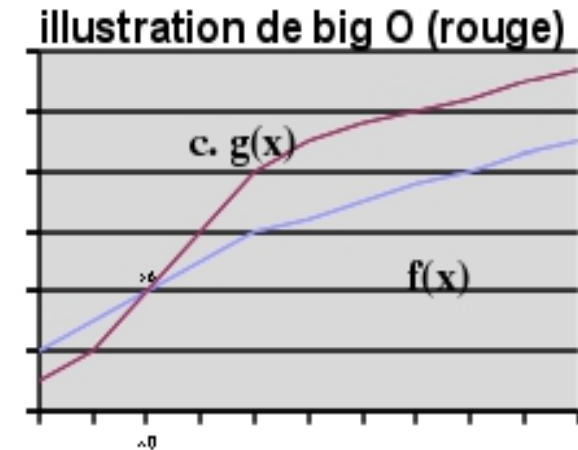
On constate que :

- Les progrès en puissance des machines sont **négligeables** face aux progrès en algorithmique
 - ⇒ Les algorithmes remarquables sont plus rares.
- Parfois, ces progrès valent *révolution* (e.g. Fourier)!

XII- Déf. de big-oh (upper bound, pessimiste)

Idee informelle : la limite supérieure d'une fonction (modulo un facteur constant)

Si $g(n)$ est une limite supérieure de $f(n)$, il est alors possible de trouver une valeur (n_0) telle que $f(n) \leq c \cdot g(n)$, pour tout $n \geq n_0$ et c une constante.



Définition : soit f et g deux fonctions de \mathbb{R} dans \mathbb{R} .

On dit que f est d'ordre inférieur ou égal à g (ou d'ordre au plus g) si l'on peut trouver un réel x_0 et un réel positif c tels que $\forall x \geq x_0, f(x) \leq c \cdot g(x)$.

- > g devient +grand que f à partir d'une certaine valeur x_0 à un facteur c près.
- > On remarque que des cas d'égalité sont possibles.

Remarques : on écrit

f est $O(g)$, f est en $O(g)$ ou $f=O(g)$ prononcé "grand O de g".

Pour le calcul de complexité, on utilise plutôt des fonctions de $\mathbb{N} \rightarrow \mathbb{R}$.

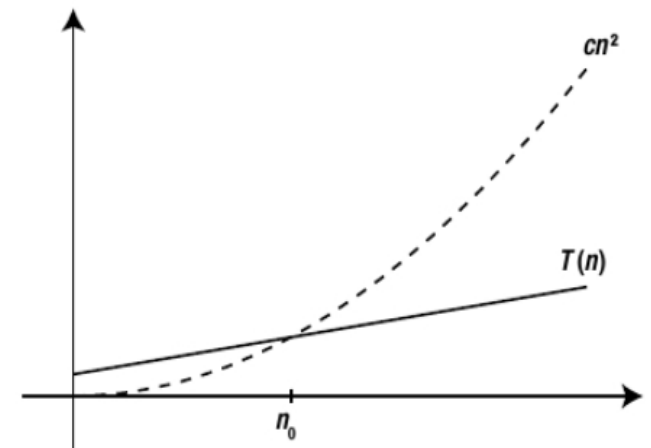
→ $O(g)$ est un ensemble de fonctions, celles d'ordre au plus g :

→ On préfère écrire $f \in O(g)$.

Un exemple simple : $5N = O(0.5 N^2)$ car $5N \leq 0.5 N^2$ à pd. $N \geq 10$.

$O(f(n))$ veut dire qu'une fonction est \leq à une autre fonction modulo une constante au sens "asymptotique", lorsque n croît (par f) vers l'infini (cf. $n \geq n_0$).

$$T(n) = O(cn^2) = O(n^2)$$



XII.1- Exemples de big-oh

I- Recherche du maximum dans un tableau de taille n

Cette recherche peut prendre $(2n-1)$ opérations de base (comparaisons).

On dira que $2n-1$ est $O(n)$ car on peut trouver une constante réelle $c > 0$ et $(n_0 \geq 1)$ tels que $2n-1 < c.n$, pour tout $n \geq n_0$.

--> Ici, on peut choisir par exemple $c=2$ et $n_0=1$.

Ce choix n'est qu'une possibilité car tout réel $c > 2$ et tout $n_0 \geq 1$ convient.

II- Montrer que $20n^3 + 10n \log n + 5$ est $O(n^3)$

→ $20n^3 + 10n \log n + 5 < c.n^3$ pour $n \geq 1$

→ Ici, $c=35$ et $n_0=1$ conviennent

III) Montrer que $n^2 + 2n + 1$ is $O(n^2)$

Il faut montrer : $n^2 + 2n + 1 \leq c * n^2$ avec $n \geq n_0$.

Avec $n_0 \geq 1$ un entier positif, posons $n_0=1$

--> $n^2 + 2n + 1 \leq c * n^2$ avec $n \geq 1$. Divisons par n^2

On a : $(n^2 + 2n + 1) / n^2 \leq c$ quand $n \geq 1$.

Sachant que $(n^2 + 2n + 1) / n^2 \leq (n^2 + 2n^2 + n^2) / n^2$ avec $n \geq 1$

On cherche c tel que $(n^2 + 2n + 1) / n^2 \leq (n^2 + 2n^2 + n^2) / n^2 \leq c$ avec $n \geq 1$.

--> $(n^2 + 2n^2 + n^2) / n^2 \leq c$ lorsque $n \geq 1$.

On simplifie sur n^2 : $(1 + 2 + 1) / 1 \leq c$ pour $n \geq 1$

--> $4 \leq c$ et $n \geq 1$

D'où : $n^2 + 2n + 1 \leq c * n^2$ pour $c = 4, n \geq 1$

IV) Montrer que $n^2/2 - 3n \in O(n^2)$

Il faut trouver un réel n_0 et un réel $c > 0$ tels que

$$\forall n \geq n_0, n^2/2 - 3n \leq c \cdot n^2$$

Divisons par n^2 (avec $n \geq 1$), on a :

$$\rightarrow 1/2 - 3/n \leq c \text{ pour } n \geq 1 \text{ et } c \geq 1/2$$

$$\text{Et } 0 \leq n^2/2 - 3n \leq c \cdot n^2 \text{ pour } n \geq n_0 = 6 \text{ et } c \geq 1/2$$

$$\text{D'où } n^2/2 - 3n \in O(n^2)$$

(V) Soit $f_1(n) = 5n^3 + 2n^2 + 22n + 6$.

Montrer que $f_1(n) = O(n^3)$.

Pour $c=6$ et $n_0=10$, on a $5n^3 + 2n^2 + 22n + 6 \leq 6n^3$ pour $n \geq 10$.

En plus, $f_1(n) = O(n^4)$ car $n^4 \geq n^3 \rightarrow n^4$ est une borne sup. Asymptotique pour f_1 .

N.B. : $f_1(n)$ n'est pas $O(n^2)$ car qq soit c et n_0 , la déf de $O(\cdot)$ ne se vérifie pas.

(VI) Montrer que $3n \log_2 n + 5n \log_2 \log_2 n + 2$ est $O(n \log n)$

Pour $n \geq 2$, $c=6$, on a $3n \log_2 n + 5n \log_2 \log_2 n + 2 \leq n \log n$

N.B. sachant que $\log_b n = \log_2(n) / \log_2(b)$, on peut omettre la base dans $O(n \log n)$.

tout polynôme de degré k est $O(n^k)$.

XIII- La fonction Ω (lower bound : optimiste)

La fonction Oméga (Ω) place une borne inférieure asymptotique.

Définition de la fonction oméga :

Pour une fonction de complexité $f(x)$, $\Omega(f(x))$ est un ensemble de fonctions de complexité $g(x)$ pour lequel il y a un réel positif c et une constante non négative x_0

tels que $\forall x \geq x_0, f(x) \leq c \cdot g(x)$.

N.B. : on dira que $f(n) = \Omega(g(n))$ si $g(n) = O(f(n))$

La notation est plus précise mais le big-oh est plus simple à calculer.

→ cherche à donner une meilleure estimation de la complexité.

Exemple : montrer que $n^2/2 - 3n \in \Omega(n^2)$

Par définition :

$$\Omega(n^2) = \{f : \mathbb{N} \rightarrow \mathbb{N} \text{ tq. } \exists n_0 \geq 0, \exists c > 0, \forall n \geq n_0, f(n) \geq cn^2 \geq 0\}$$

On peut trouver les constantes n_0 et c telles que :

$$0 \leq c \cdot n^2 \leq 1/2 n^2 - 3n \text{ pour } n \geq 7 \text{ et } c \geq 1/14$$

D'où :

$$1/2 n^2 - 3n \in \Omega(n^2)$$

XIV- La fonction Θ (égalité entre O et Ω)

- Si une fonction f est à la fois $O(n^2)$ et $\Omega(n^2)$, sa courbe se place à la fois au-dessous d'une certaine fonction quadratique **pure** (de la forme ax^2+bx+c) et au-dessus d'une certaine autre fonction quadratique (pure).

La fonction f est donc aussi bonne / mauvaise (que ces 2 fonctions).

Donc, l'accroissement de f est similaire à une fonc. quad. pure : **ordre Θ** :

L'ordre Θ dit que 2 foncs. sont asymptotiquement égales (modulo un facteur cst.).

Définition : pour une fonction de complexité $f(x)$, $\Theta(f(x)) = O(f(x)) \cap \Omega(f(x))$.

Ce qui veut dire que $\Theta(f(x))$ est l'ensemble de fonctions de complexité $g(x)$ pour lequel il y a les réels positifs c et d et une constante non négative x_0 tels que :

$$\forall x \geq x_0, c. g(x) \leq f(x) \leq d. g(x).$$

Exemples de Θ (Théta)

- $f(n) = n(n-1)/2$ est à la fois $O(n^2)$ et $\Omega(n^2)$

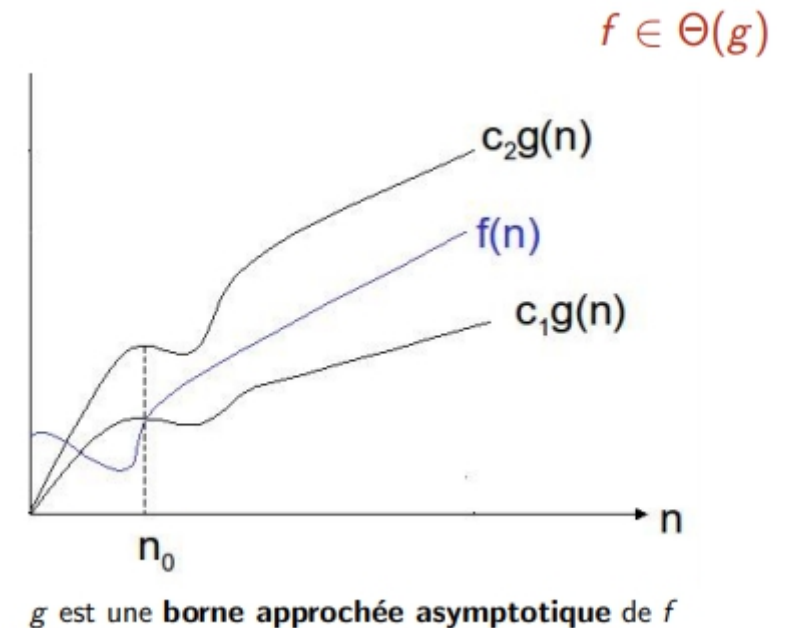
(cf. sections précédentes)

→ dans ce cas, $f(n) = \Theta(n^2)$

- $n^2 + 10n \in \Theta(n^2)$ car $n^2 + 10n \in O(n^2)$

et $n^2 + 10n \in \Omega(n^2)$ pour n grand.

On a également $n^2 \in \Theta(n^2 + 10n)$ v. (3) ci-dessous



Remarques :

- (1) Si $f(n) = \Theta(g(n))$, alors $f(n)$ est à la fois $O(g(n))$ et $\Omega(g(n))$.
- (2) Si $f(x) = \Theta(g(n))$, on dira que $g(x)$ est l'ordre de $f(x)$.
- (3) Si l'on a $O(N)$ et $N = \Theta(x)$, alors on a $O(x)$.
- (4) Si $g(n) \in \Theta(f(n))$ ssi $f(n) \in \Theta(g(n))$ (Θ porte l'égalité)

Exemple :

Montrer que $1/2n^2 - 3n$ est $\Theta(n^2)$.

On doit montrer que :

$$c_1 n^2 \leq n^2/2 - 3n \leq c_2 n^2 \text{ pour tout } n \geq n_0.$$

On divise par n^2 :

$$c_1 \leq 1/2 - 3/n \leq c_2$$

--> Pour $c_2 \geq 1/2$, on a l'inégalité à droite.

--> Pour la partie gauche, $n \geq 7$ et $c_1 \leq 1/14$.

Donc, avec $n_0=7$, $c_1=1/14$ et $c_2 = 1/2$, nous aurons

$$T(1/2n^2 - 3n) = \Theta(n^2).$$

Exemple :

Montrer que $6n^3 \neq \Theta(n^2)$.

Preuve par contradiction / réfutation.

Supposons que c'est le cas.

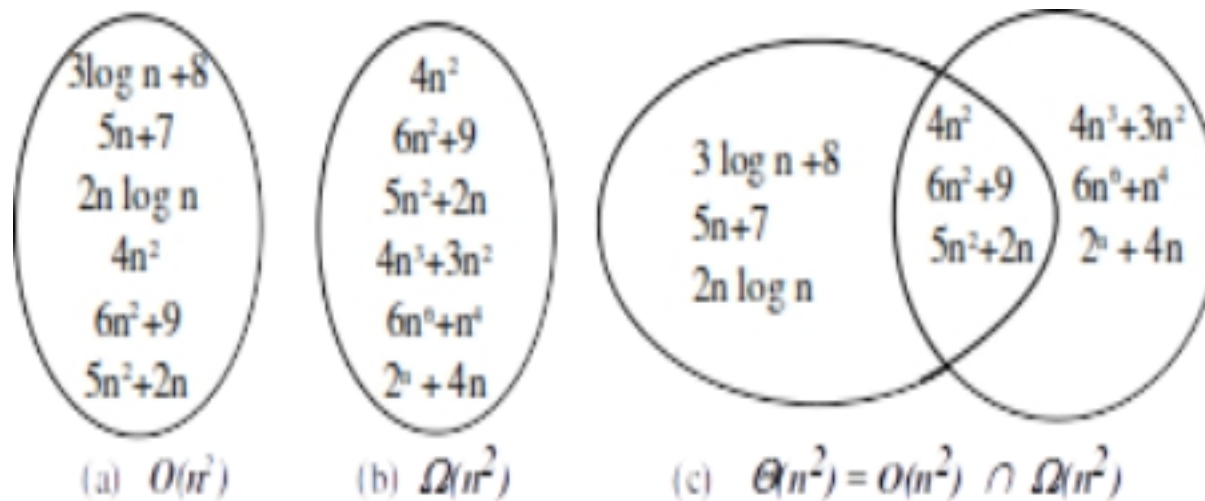
Il existe alors c_2 and n_0 tels que $6n^3 \leq c_2 n^2$.

En divisant par n^2 (pour $n \geq 1$), on aura $6n \leq c_2$.

Or, pour une constante c_2 donnée, la relation $6n \leq c_2$ n'est pas valide pour tout n .

XV- Illustration des 3 fonctions o , Ω , Θ

Quelques exemples courants de ces 3 fonctions de complexité pour n^2 .



- Voir Annexes pour les famille little-oh ($o(N)$) et little-oméga ($o(\omega)$)
- N.B. : Réflexivité :
 - On a $f \in O(f)$ et $f \in \Omega(f)$ donc $f \in \Theta(f)$.
 - Il n'y a pas de **réflexivité** pour little-oh : $f(n)$ n'est jamais $o(f(n))$

XVI- Calculs : le Modèle (de la machine)

Nécessité : il faut un modèle de calcul pour estimer la complexité.

En générale, on considère que les hypothèses suivantes sont vérifiées :

- **Ordinateur normal** (séquentiel) avec des instructions simples (+, -, *, /).
- **Chaque opération prend une unité de temps.**
- Les entiers sont de taille fixe (32 / 64 bits)
- L'opération caractéristique de base n'est pas compliquée comme la multiplication ou inversion de matrices ou le tri (qui demandent $\gg 1$ unité).
- La quantité de RAM est **illimitée** (dans la limite du raisonnable !).
- On est **indépendant du langage** : certains langages peuvent diminuer/augmenter le temps (paramètres par réf, copie de tableau en paramètre, etc.), voir + loin.

XVII- Règles basiques et empiriques de calcul

On repère les instructions *caractéristiques* (on dit aussi *fondamentales*)

- **Règle des opérations simples :**

```
if (A > B):  
    A = A - 1  
    B = 2 * B
```

Trois instructions (test+2 instruction) si $A > B$ et, une (le test) instruction sinon.

Le traitement est $O(1)$ car le nombre d'opérations est borné par une cste.

- **Règle des Séquences :**

On additionne les complexités on prend le maximum selon la règle de somme

- **Règle de la Conditionnelle :**

if (Cond) {S1} else {S2} -->

$$O(\text{conditionnelle}) = \max (O(S1), O(S2)) + \text{cout}(\text{Cond})$$

Règle de la boucle (for, while, ...):

Le temps d'exécution des instructions à l'intérieur * le nombre d'itérations.

Boucles imbriquées :

Le temps de la boucle interne multiplié par le nombre d'itérations externes

Exemple (de complexité = $O(N^2)$)

```
for i in range(n):  
    for j in range(n):  
        k = k + 1
```

Autres règles générales :

- On analyse de l'intérieur vers l'extérieur.
- Pour un appel de fonction, on analyse d'abord la fonction

Cas de la récursivité :

Parfois, une fonction récursive est en fait une itération «cachée»

Exemple ($O(N)$) où l'opération caractéristique est "*" ou un appel à "fact"

```
def fact(n: int) → long
  if (n <= 1): return 1
  return n * fact(n-1)
```

Remarque : le calcul de complexité formelle des algorithmes récursifs est souvent plus simple !

--> Leur preuve de justesse aussi !

XVIII- Quelques exemples de calculs simples

XVIII.1- Exemple : calcul de la médiane d'une suite

Calculer la médiane M dans une suite d'entiers S de taille N (on note $|S| = N$) telle que :

La moitié des nombres de S soient plus petits que M et l'autre moitié plus grande.

Sol 1- Pour calculer la médiane, il suffit de trier S puis de choisir le milieu.

→ Complexité : celle de l'algorithme de tri (au mieux $O(N \cdot \log(N))$).

Sol 2- Construire un TAS (voir + loin, min_heap, $O(N)$) puis retirer k éléments ($k = \lfloor N / 2 \rfloor$).

→ Complexité : $O(N \cdot \log_2 N)$ pour $N \geq 4$

Sol 3- Meilleure méthode pour trouver le $k^{\text{ième}}$ plus petit élément de S (ici $k = \lfloor |S| / 2 \rfloor$).

→ Le point fort de cette méthode est de ne trier que la plus petite sous séquence de S contenant le $k^{\text{ième}}$ plus petit élément.

→ Voyons quelques détails



XVIII.1.1- Solution : un stratégie Diviser pour régner

Pour une séquence S , la méthode se généralise pour $k = 1..|S|$

($k = |S|/2$ pour la médiane)

Pour une valeur v appartenant à S , on peut scinder S en 3 sous tableaux :

- S_g : contenant les éléments plus petits que v
- S_v : contenant les éléments = v (contenant v lui-même)
- S_d : contenant les éléments de S plus grands que v

Exemple : $S = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$, $|S|=11$ et $v = 5$ (p. ex.) :

→ $S_g = \langle 2, 4, 1 \rangle$, $S_v = \langle 5, 5 \rangle$, $S_d = \langle 36, 21, 8, 13, 11, 20 \rangle$

$v \in S$ sera choisi **aléatoirement** (mais $V =$ le 1^{er} élément est plus simple).

Pour trouver le k ième plus petit élément, il suffit de repérer le sous tableau susceptible de contenir cet élément et de traiter celui-ci.

Exemple : si $k=8$, on sait que le 8e plus petit élément ne peut être que dans S_d car pour $v = 5$, la somme des tailles de S_g et S_v : $|S_g| + |S_v| = 5$.

On notera : $selection(S, k=8) = selection(S_d, k=3)$ sachant $|S_g| = 3$ et $|S_v|=2$.

Généralisation : $selection(S, k) = selection(S_g, k)$ si $k \leq |S_g|$

$= v$ si $|S_g| < k \leq |S_g| + |S_v|$

$= selection(S_d, k-(|S_g| + |S_v|))$ si $k > |S_g| + |S_v|$

On répétera ce traitement (récursif) sur le sous-tableau concerné jusqu'à arriver à un singleton qui est le résultat recherché (voir l'algorithme en Annexes).

👉 Le découpage déséquilibré des S_d et S_g peut donner une complexité défavorable

👉 Aho & al montrent : en moyenne, 2 divisions suffisent pour trouver la médiane :

→ Voir [Aho & al], [Wirth] pour 2 algorithmes de **complexité moyenne $O(n)$**

pour obtenir le k ième plus petit élément de S .

XVIII.1.2- Détails du déroulement pour cet exemple

Trouver la médiane de $S = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ avec $|S|=11$ $k=6$

Appel initial de l'algo : **selection(S,6)** (6e plus petit élément)

selection(S,6) : **V1=2**, $Sg1 = \langle 1 \rangle$ $Sv1 = \langle 2 \rangle$ $Sd1 = \langle 36, 5, 21, 8, 13, 11, 20, 5, 4 \rangle$

$k > |Sg1| + |Sv1|$ \rightarrow selection($Sd1, 6 - |Sg1| - |Sv1|$) = selection($Sd1, 4$)

selection($Sd1, 4$) : $V2=36$, $Sg2 = \langle 5, 21, 8, 13, 11, 20, 5, 4 \rangle$ $Sv2 = \langle 36 \rangle$ $Sd2 = \langle \rangle$

$k < |Sg2|$ \rightarrow selection($Sg2, 4$)

selection($Sg2, 4$) : $V3=5$, $Sg3 = \langle 4 \rangle$ $Sv3 = \langle 5, 5 \rangle$ $Sd3 = \langle 21, 8, 13, 11, 20 \rangle$

$k > |Sg3| + |Sv3|$ \rightarrow selection($Sd3, 4 - |Sg3| - |Sv3|$) = selection($Sd3, 1$)

selection($Sd3, 1$) : $V4=21$, $Sg4 = \langle 8, 13, 11, 20 \rangle$ $Sv4 = \langle 21 \rangle$ $Sd4 = \langle \rangle$

$k < |Sg4|$ \rightarrow selection($Sg4, 1$)

selection($Sg4, 1$) : $V5=8$, $Sg5 = \langle \rangle$ $Sv5 = \langle 8 \rangle$ $Sd5 = \langle 13, 11, 20 \rangle$

$|Sg5| < k = |Sg5| + |Sv5|$ car $0 < 1 = 1$

\rightarrow **le résultat est = V5 = 8** (8 est le 6e plus petit élément de S)

Vérification :

si on trie S, on aura $S_trié = \langle 1, 2, 4, 5, 5, 8, 11, 13, 20, 21, 36 \rangle$ dont '8' est le 6e PP.

XVIII.2- Exemple : séquence de somme maximale

- Soit une suite d'entiers $A_1 \dots A_n$ (la suite peut contenir des entiers négatifs)
--> **But** : trouver la séquence contiguë de somme maximale.
- Hypothèse : on pose la somme maximum = 0 si tous les entiers sont négatifs.
- Exemples : pour (on commence à l'indice 1) :
 - la séquence : -2, 11, -4, 13, -5, -2 La réponse = 20 (A2..A4)
 - la séquence : 4, -3, 5, -2, -1, 2, 6, -2 --> La réponse = 11 (A1..A7)
- Ce problème a de multiples applications (dont le pb. d'appariement *de sous-intervalles*)
 - E.g., pour estimer le max de vraisemblance d'un motif dans une image.
- Cet ex. est intéressant car il y a plusieurs algorithmes possibles (4 cités ici) :
--> $O(N^3)$ à $O(N)$. Comparons les temps en fonction de N ⇔.

Rappel : le tableau suivant donne une idée comparative de ces temps en **secondes** :

↓N / T(N)→	$O(N^3)$	$O(N^2)$	$O(N \cdot \log N)$	$O(N)$
10	0.00103	0.00045	0.00066	0.00034
100	0.47015	0.01112	0.00486	0.00063
1000	448.77	1.12330	0.05843	0.00333
10000	NA	111.13	0.68631	0.03042
100000	NA	NA	8.01130	0.29832

- $O(n^3)$: naïf, basique sans opti. (aucun résultat/somme intermédiaire conservé)
- $O(n^2)$: optimisation dans les sommes partielles
- $O(n \log n)$: approche dichotomique :

la somme recherchée est soit dans la moitié gche, soit dte, soit au milieu des 2.

- $O(n)$: approche récursive (petit inconvénient : il manquera les indices de l'intervalle)

Exemple : le cas Dichotomique par la stratégie "Diviser-pour-régner" ($O(n \log n)$)

la somme recherchée est soit dans la moitié gche., soit dte., soit au milieu des 2.

- Les 2 premiers cas (soit à gauche, soit à droite) ont une solution récursive directe ;
- Le 3^e peut être obtenu en faisant la somme de
la plus grande somme dans la 1^e moitié qui inclut le dernier élément de cette
moitié et
la plus grande somme de la 2^e moitié qui inclut le 1^e élément de cette moitié
(continuité de la sous-séquence).
- Ces deux sommes pourront ensuite être additionnées (étape **Régner**).

Exemple1 :

la séquence A :

<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>↓</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u><== indices</u>
4	-3	5	-2	-1	2	6	-2		<== éléments


- Meilleure somme de la 1e moitié = 6 (A1..A3) et celle de la 2e moitié = 8 (A6..A7).
- Meilleure somme de la 1e moitié qui inclut le dernier élément de cette moitié est 4 (A1..A4) ;
- pour l'autre moitié, ce sera 7 (A5..A7)

On fait la somme des deux :

» la meilleure somme : $4+7=11$ (A1..A7).

● Exemple 2 :

la séquence précédente (tableau B):

<u>1</u>	<u>2</u>	<u>3</u>	<u></u>	<u>4</u>	<u>5</u>	<u>6</u>	<u><== indices</u>
-2	11	-4		13	-5	-2	<== éléments

- La meilleure somme de la 1^e moitié = 11 (B2) et celle de la 2^e moitié = 13 (B4).
 - La meilleure somme de la 1^e moitié qui inclut le dernier élément de cette moitié est 7 (B2..B3)
 - et pour l'autre moitié, c'est 13 (B4).
- »»» la meilleure somme = 20 (B2..B4).

● Exemple 3 :

nombre impaire de données (C) : 1 2 3 4 5 6 7 \Leftarrow indices

4 -3 5 -2 -1 2 -6 \Leftarrow élés.

- La meilleure somme de la 1^e moitié = 6 (C1..C3) et celle de la 2^e moitié = 2 (C6).
- La meilleure somme de la 1^e moitié qui inclut le dernier élément de cette moitié est 4 (C1..C4) et pour l'autre moitié, c'est 1 (C5..C6).

» la meilleure somme est dans la **première moitié** = 6 (C1..C3).

Et enfin, pour la 4^e solution (le cas linéaire) :

```
def max_sous_sequence(V):  
    max_finissant_ici = meilleur_max_jsq_ici = 0  
    for x in V :  
        max_finissant_ici = max(0, max_finissant_ici + x)  
        meilleur_max_jsq_ici = max(meilleur_max_jsq_ici, max_finissant_ici)  
    return meilleur_max_jsq_ici
```

Explication : considérons l'indice de chaque élément x dans V :

A chaque indice, on calcule le max (somme > 0) du sous-séquence finissant à cet indice.

Cette sous-séquence est soit vide (somme=0) soit contient un élément de plus que la sous-séquence finissant à l'indice précédent.

L'algorithme ci-dessus ne donne pas les indices. \Rightarrow

N.B. : L'algorithme ci-dessus ne donne pas les indices.

Pour pallier l'inconvénient de cette 4^e sol. :

- Modifier (légèrement) l'algorithme pour trouver le début et la fin de la sous-séquence de somme max.
 - Pour le début, on doit conserver le dernier indice de la dernière somme négative calculée.
- ⇒ Du fait de conserver la meilleur somme max jusqu'à la position précédente, on est en présence de la **PrD** (voir le chapitre 2).

Notes sur les applications de *max_sum* :

- * En dimension 2, l'intervalle de *max_sum* est un estimateur du MLE de certains motifs dans les images numériques.
- * Utilisation en méthodes de segmentation dans l'analyse des séquences de protéine (et l'ADN).
 - On y utilise (également) *min_sum* pour supprimer des séquences alignées de Protéine (source : plusieurs papiers recherche en bio-info).

XIX- A propos de la complexité moyenne $A(n)$

Un exemple : la recherche de X dans $S[1:n]$

- La probabilité pour que $S[k]=X$, $1 \leq k \leq n$ est $1/n$,
- Pour arriver à l'indice $k=1..n$, on aura fait les comparaisons :

$$A(n) = \sum_{k=1}^n \left(k \cdot \frac{1}{n}\right) = \frac{1}{n} \cdot \sum_{k=1}^n k = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} \text{ opérations de base } \underline{\text{si } X \text{ est dans } S}$$

Mais pour être complet, il faut tenir compte du cas où X n'est pas dans S :

- Si $\Pr(X \in S) = p$ alors $\Pr(X \notin S) = 1-p$ et pour k donné, $\Pr(S[k] = X) = p/n$
- On fait k comparaisons pour arriver à k tel que $S[k]=X$
et n comparaisons si X n'est pas dans S . ⇨

La complexité moyenne (**dépendra de p**) :

$$A(n) = \sum_{k=1}^n \left(k \cdot \frac{p}{n}\right) + n(1 - p) = \frac{p}{n} \cdot \frac{n(n+1)}{2} + n(1 - p) = n \left(1 - \frac{p}{2}\right) + \frac{p}{2}$$

$$\rightarrow \text{si } p=1, A(n) = (n+1)/2$$

$$\rightarrow \text{si } p = \frac{1}{2}, A(n) = 3n/4 + \frac{1}{4} \quad \rightarrow \text{seul } \frac{3}{4} \text{ de } S \text{ est recherché en moyenne}$$

$$\rightarrow \text{si } p=0, A(n) = O(n) = n.$$

Remarques sur $A(n)$:

On a supposé une **probabilité identique** pour chaque élément du tableau.

--> à changer si l'on connaît une distribution différente de taille/ valeurs :

↳ c-à-d. telle valeur est plus fréquente que telle autre...

👉 La complexité moyenne est un **très bon indicateur** mais son calcul est souvent **difficile** (et donc **rare**) car les probas nécessaires ne sont pas toujours connues/disponibles.

XX- Propriétés des limites de fonctions

- On peut calculer le taux relatif de croissance de 2 fonctions **f** et **g** en calculant

$$\text{Lim}_{n \rightarrow \infty} \frac{f(N)}{g(N)}$$

Si **f** et **g** admettent des limites, on a alors les propriétés suivantes :

- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = c > 0$, **f** et **g** sont du même ordre, $f=O(g)$ et $g=O(f) \rightarrow f= \Theta(g)$.
- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = 0$, alors $f = o(g)$ et **f** est d'ordre inférieur à **g**.
- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = +\infty$, **f** est d'ordre supérieur à **g**.
- On note $f=\Omega(g)$ qui est équivalent à $g=o(f)$ \Rightarrow

- On pourra (également) utiliser la règle de l'"Hôpital" :

Si f et g sont définies sur $[a, b[$, dérivables en a telles que $f(a)=g(a) \neq 0$

Alors $\lim_{n \rightarrow \infty} \frac{f(N)}{g(N)} = \lim_{n \rightarrow \infty} \frac{f'(N)}{g'(N)}$ si la limite existe.

- Plus précisément, si $g'(N) \neq 0$ Alors

Si $\lim_a f(N) = \lim_a g(N) = 0$ et $\lim_{n \rightarrow \infty} \frac{f'(N)}{g'(N)} = L$ Alors $\lim_{n \rightarrow \infty} \frac{f(N)}{g(N)} = L$

Si $\lim_a f(N) = \lim_a g(N) = +\infty$ et $\lim_{n \rightarrow \infty} \frac{f'(N)}{g'(N)} = L$ Alors $\lim_{n \rightarrow \infty} \frac{f(N)}{g(N)} = L$

Utilisation des limites

On peut utiliser les propriétés des limites pour trouver une idée de la complexité (lorsqu'elle est difficile à trouver).

XX.1- Exemple trivial d'utilisation des limites

- On a vu : pour $a > 0$, $a^n \in o(n!)$ car

avec les propriétés des limites : $\lim_{n \rightarrow \infty} \frac{a^n}{n!} = 0$

--> a^n est d'ordre inférieur à $n!$.

- On peut montrer que $\log n \in o(n)$:

$$\lim \frac{\log x}{x} = \lim \frac{\frac{d(\log x)}{dx}}{\frac{dx}{dx}} = \lim \frac{\frac{1}{x \log 2}}{1} = \lim \frac{1}{(x \log 2)} = \lim \frac{1}{x} = 0$$

XX.2- Application : approximation de la complexité par programme

- On utilise la règle "limite" pour calculer la complexité empirique par programme.

Pour montrer qu'un algorithme est $O(g(N))$, on calcule les valeurs de $T(N)/f(N)$ pour un intervalle de N habituellement espacé par un **facteur de 2**, puis d'étudier la limite de $T(N)/g(N)$.

Rappel des règles de la limite :

- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = c > 0$, f et g sont du même ordre, $f=O(g)$ et $g=O(f) \rightarrow f= \Theta(g)$.
- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = 0$, alors $f = o(g)$ et f est d'ordre inférieur à g .
- Si $\lim_{n \rightarrow \infty} \frac{f}{g} = +\infty$, f est d'ordre supérieur à g .
- On note $f=\Omega(g)$ qui est équivalent à $g=o(f)$

- Dans la méthode utilisée, $f(N) = T(N)$ = le temps empirique observé.
- On calcule également différentes fonc. de complexité $g(N)$ (e.g. $\log N$, N^2 , N^3 ...)
- Calculer ensuite $f(N)/g(N)$ et observez sa limite (empirique)
 - Si la limite converge vers une cste > 0 alors g est une estimation de la complexité \oplus
 - Si $f(N)$ est surestimée, les valeurs convergent vers 0, c-à-d. $f=o(g)$.
 - Si $f(N)$ est sous estimée, les valeurs divergent (tendent vers l'infini) : $g=o(f)$
- Le taux de convergence / divergence signale aussi le degré de justesse de $T(n)$.

XX.3- Exemple 1

1- **Montrer** que la probabilité pour que deux entiers distincts et aléatoires $I, J \leq N$ soient premiers entre eux approche $6/\Pi^2 = 0.608$ (pour N grand).

2- **Estimer la complexité de la solution.**

On propose l'algorithme (simplifié) suivant :

```
Fonction proba_prime(N)=           // renvoie un double
  Rel=0                             // Rel : nombre de fois où i et j sont premiers
  Total_essais=0                     // Total : nombre d'essais
  Pour (i=1; i <= N; i++)           // Simulation du tirage aléatoire sans remise
    Pour (j=i+1; j <= N; j++)
      Total_essais ++
      Si (pgcd(i,j) == 1) Rel++      // On sait : pgcd est O(log N)
  renvoyer Rel/Total_essais
```

- La complexité : 2 Boucles ($O(N^2)$) et la complexité de pgcd est $O(\log N)$
--> $N^2 \log N \rightarrow O(N^2 \cdot \log(N))$

XX.3.1- Estimation empirique de la complexité de l'exemple

- Le tableau obtenu sur une machine Centrino 1,6 avec 512Mo de mémoire

Taille	Ln	Ln * Ln	N	N ln	N+N Ln	N * N	N * N * N
50	0,0000702961	0,0000179692	0,0000055000	0,0000014059	0,0000011197	0,0000001100	0,0000000022
75	0,0001528667	0,0000354064	0,0000088000	0,0000020382	0,0000016549	0,0000001173	0,0000000016
112	0,0003401506	0,0000720887	0,0000143304	0,0000030371	0,0000025060	0,0000001279	0,0000000011
168	0,0008050408	0,0001571129	0,0000245536	0,0000047919	0,0000040094	0,0000001462	0,0000000009
252	0,0016952926	0,0003065945	0,0000371984	0,0000067274	0,0000056970	0,0000001476	0,0000000006
378	0,0175406665	0,0029555146	0,0002754021	0,0000464039	0,0000397125	0,0000007286	0,0000000019
567	0,0194570992	0,0030687692	0,0002175750	0,0000343159	0,0000296409	0,0000003837	0,0000000007
8500	0,7441067491	0,0822415378	0,0007920641	0,0000875420	0,0000788294	0,0000000932	0,0000000000
1275	0,0227007100	0,0031746130	0,0001273145	0,0000178045	0,0000156201	0,0000000999	0,0000000001
1912	0,0419171490	0,0055476013	0,0001656496	0,0000219232	0,0000193608	0,0000000866	0,0000000000
2868	0,0851187651	0,0106914718	0,0002362838	0,0000296788	0,0000263669	0,0000000824	0,0000000000
4302	0,1815649459	0,0217005521	0,0003531204	0,0000422048	0,0000376990	0,0000000821	0,0000000000
6453	0,3971829320	0,0452769414	0,0005399362	0,0000615501	0,0000552517	0,0000000837	0,0000000000
9679	0,8774637252	0,0956080934	0,0008320189	0,0000906564	0,0000817491	0,0000000860	0,0000000000
14518	1,9480021329	0,2032737924	0,0012858511	0,0001341784	0,0001214999	0,0000000886	0,0000000000
21777	4,4913626210	0,4496484276	0,0020600849	0,0002062434	0,0001874746	0,0000000946	0,0000000000

Rappel : la vitesse de la machine n'a pas d'effet car les rapports $T(N)/g(N)$ et les constantes C (de la définition de big-Oh) sont négligées.

Ici, chaque colonne représente $T(N) / g(N)$ et on étudie la limite.

Ici, $N^2 \log(N)$ n'est pas prise en compte, on sera $\Omega(N^2)$.

Deuxième tableau (des écarts types) : vers une décision automatique

Taille	Ln	Ln * Ln	N	N ln	N+N Ln	N * N	N * N * N
100	0,0002692626	0,0000584696	0,0000124000	0,0000026926	0,0000022122	0,0000001240	0,0000000012
200	0,0010877038	0,0002052923	0,0000288150	0,0000054385	0,0000045750	0,0000001441	0,0000000007
300	0,0186707681	0,0032734012	0,0003549800	0,0000622359	0,0000529522	0,0000011833	0,0000000039
400	0,0175603142	0,0029308884	0,0002630300	0,0000439008	0,0000376216	0,0000006576	0,0000000016
500	0,0195906159	0,0031523494	0,0002434960	0,0000391812	0,0000337504	0,0000004870	0,0000000010
600	0,0209661521	0,0032775336	0,0002235317	0,0000349436	0,0000302195	0,0000003726	0,0000000006
700	0,0204386137	0,0031198844	0,0001912786	0,0000291980	0,0000253313	0,0000002733	0,0000000004
800	0,0231905766	0,0034692481	0,0001937750	0,0000289882	0,0000252160	0,0000002422	0,0000000003
1000	0,0255741544	0,0037022380	0,0001766600	0,0000255742	0,0000223401	0,0000001767	0,0000000002
1200	0,0304775258	0,0042986171	0,0001800733	0,0000253979	0,0000222585	0,0000001501	0,0000000001
1500	0,0377177202	0,0051574707	0,0001838920	0,0000251451	0,0000221204	0,0000001226	0,0000000001
2000	0,0513148277	0,0067511493	0,0001950195	0,0000256574	0,0000226743	0,0000000975	0,0000000000
2500	0,0671867215	0,0085872094	0,0002102688	0,0000268747	0,0000238291	0,0000000841	0,0000000000
3500	0,1237624829	0,0151660077	0,0002885617	0,0000353607	0,0000315006	0,0000000824	0,0000000000
6500	0,4037839057	0,0459913735	0,0005453914	0,0000621206	0,0000557685	0,0000000839	0,0000000000
10000	0,9451901502	0,1026227167	0,0008705523	0,0000945190	0,0000852618	0,0000000871	0,0000000000
15000	2,0837667777	0,2167022598	0,0013358064	0,0001389178	0,0001258318	0,0000000891	0,0000000000
25000	5,7303253575	0,5658669007	0,0023211556	0,0002292130	0,0002086126	0,0000000928	0,0000000000
45000	18,5556198471	1,7318365074	0,0044180592	0,0004123471	0,0003771471	0,0000000982	0,0000000000
90000	76,7338421380	6,7265750823	0,0097260699	0,0008525982	0,0007838822	0,0000001081	0,0000000000
150000	213,5554605627	17,9181458481	0,0169682493	0,0014237031	0,0013134956	0,0000001131	0,0000000000

La colonne

$N*N$ représente le minimum d'écart type par rapport aux autres.

--> A titre d'indication, une fois ces valeurs harmonisées (toute valeur multipliée par $\frac{1}{min}$ où min est le minimum $\neq 0$ de la colonne), les écarts types seront :

30749140888, 4628790867, 103855, 15560, 19636, 9, 1361751

avec la valeur **9** pour la colonne $N*N$ d'où $O(N^2)$.

XX.4- Exemple 2

- Tableau Tri Bulle (Bubble-Sort) de complexité $O(N^2)$. Colonne N^2 quasi constante.

taille	Ln(N)	Ln^2(N)	N	N.Ln(N)	N+N.Ln(N)	N^2	N^2.Ln(N)	N^3	2^N
20	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
26	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
33	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
42	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
54	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
70	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
91	0.221687241324	0.049145232966	0.010989010989	0.002436123531	0.001994064805	0.000120758363	0.000026770588	0.000001327015	0.000000000000
118	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
153	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
198	0.189097864044	0.035758002186	0.005050505051	0.000955039717	0.000803163260	0.000025507601	0.000004823433	0.000000128826	0.000000000000
257	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
334	0.344166490267	0.059225286511	0.005988023952	0.001030438594	0.000879151372	0.000017928215	0.000003085145	0.000000053677	0.000000000000
434	0.329324112275	0.054227185463	0.004608294931	0.000758811319	0.000651529184	0.000010618191	0.000001748413	0.000000024466	0.000000000000
564	0.47355534160	0.074751614644	0.005319148936	0.000839637472	0.000725168316	0.000009431115	0.000001488719	0.000000016722	0.000000000000
733	0.606322822134	0.091906841160	0.005457025921	0.000827179839	0.000718299495	0.000007444783	0.000001128485	0.000000010157	0.000000000000
952	0.729015468197	0.106292710574	0.005252100840	0.000765772551	0.000668328227	0.000005516913	0.000000804383	0.000000005795	0.000000000000
1237	0.983084711277	0.138065078507	0.005658852061	0.000794732992	0.000696864827	0.000004574658	0.000000642468	0.000000003698	0.000000000000
1608	1.760862314384	0.238510468478	0.008084577114	0.001095063628	0.000964430591	0.000005027722	0.000000681010	0.000000003127	0.000000000000
2090	2.616116547153	0.342203289414	0.009569377990	0.001251730405	0.001106936642	0.000009431115	0.000000598914	0.000000002191	0.000000000000
2717	4.173367445764	0.527787752647	0.012145748988	0.001536020407	0.001363574971	0.000004470279	0.000000565337	0.000000001645	0.000000000000
3532	6.487450192701	0.794094528354	0.015005662514	0.001836763928	0.001636454208	0.000004248489	0.000000520035	0.000000001203	0.000000000000
4591	10.199418624677	1.209629538156	0.018732302331	0.002221611550	0.001986068066	0.000004080223	0.000000483906	0.000000000889	0.000000000000
5968	16.447808935155	1.891821110261	0.023961126005	0.002756000157	0.002471705476	0.000004014934	0.000000461796	0.000000000673	0.000000000000
7758	26.237986797065	2.929497664523	0.030291312194	0.003382055529	0.003042371667	0.000003904526	0.000000435944	0.000000000503	0.000000000000
10085	173.449822979231	18.814784922781	0.158552305404	0.017198792561	0.015515739260	0.000015721597	0.000001705383	0.000000001559	0.000000000000
13110	70.350259876718	7.420028582791	0.050877192982	0.005366152546	0.004854170322	0.000003880793	0.000000409318	0.000000000296	0.000000000000
17043	114.024794809692	11.702658714125	0.065188053746	0.006690418049	0.006067676739	0.000003824917	0.000000392561	0.000000000224	0.000000000000
22155	207.379336672363	20.725874351166	0.093658316407	0.009360385316	0.008509891068	0.000004227412	0.000000422495	0.000000000191	0.000000000000
28801	310.571546092015	30.246059969265	0.110725322037	0.010783359817	0.009826384174	0.000003844496	0.000000374409	0.000000000133	0.000000000000
37441	637.575253131689	60.545457760789	0.179322133490	0.017028798727	0.015551953250	0.000004789459	0.000000454817	0.000000000128	0.000000000000
48673	702.129568954109	65.054886724688	0.155692067471	0.014425442626	0.013202209375	0.000003198736	0.000000296375	0.000000000066	0.000000000000
63274	1117.841984700321	101.114314837251	0.195309289756	0.017666687497	0.016201208380	0.000003086723	0.000000279209	0.000000000049	0.000000000000
82256	1594.067060805795	140.848611182641	0.219327465474	0.019379340848	0.017806034844	0.000002666401	0.000000235598	0.000000000032	0.000000000000
106932	2352.946581879067	203.191456570499	0.254806793102	0.022004138910	0.020254995096	0.000002382886	0.000000205777	0.000000000022	0.000000000000
139011	3559.018965707158	300.534225407817	0.303191833740	0.025602426899	0.023608826823	0.000002181064	0.000000184176	0.000000000016	0.000000000000
180714	5407.664551152306	446.741970390010	0.362218754496	0.029923882771	0.027640431101	0.000002004376	0.000000165587	0.000000000011	0.000000000000
234928	8066.525657665044	652.260309202863	0.424636484370	0.034336161112	0.031767441665	0.000001807518	0.000000146156	0.000000000008	0.000000000000
305406	12292.510568139573	973.325191905950	0.508329895287	0.040249735002	0.037296579110	0.000001664440	0.000000131791	0.000000000005	0.000000000000
397027	19167.282689962205	1486.785616013455	0.622375808194	0.048277025719	0.044801798158	0.000001567591	0.000000121596	0.000000000004	0.000000000000
516135	30073.991311182079	2286.278595098182	0.766460325302	0.058267684445	0.054151026577	0.000001485000	0.000000112892	0.000000000003	0.000000000000
670975	47661.954466643336	3552.491341985035	0.953025075450	0.071033875281	0.066106608707	0.000001420359	0.000000105867	0.000000000002	0.000000000000
872267	76850.315251526772	5618.185043586380	1.205163097996	0.088104118637	0.082102005833	0.000001381645	0.000000101006	0.000000000002	0.000000000000

Tableau de Merge_sort : $O(N \cdot \log N)$: Colonne $N \cdot \log N$ quasi constante.

Taille	Ln(N)	Ln^2(N)	N	N.Ln(N)	N+N.Ln(N)	N^2	N^2.Ln(N)	N^3	2^N
20	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
26	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
33	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
42	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
54	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
70	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
91	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
118	0.209613520641	0.043937828035	0.008474576271	0.001776385768	0.001468556475	0.000071818443	0.000015054117	0.000000608631	0.000000000000
153	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
198	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
257	0.180210179983	0.032475708970	0.003891050584	0.000701206926	0.000594137331	0.000015140275	0.000002728432	0.000000058912	0.000000000000
334	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000
434	0.164662056138	0.027113592731	0.002304147465	0.000379405659	0.000325764592	0.000005309096	0.000000874207	0.000000012233	0.000000000000
564	0.157851844720	0.024917204881	0.001773049645	0.000279879157	0.000241722772	0.000003143705	0.000000496240	0.000000005574	0.000000000000
733	0.151580705534	0.022976710290	0.001364256480	0.000206794960	0.000179574874	0.000001861196	0.000000282121	0.000000002539	0.000000000000
952	0.145803093639	0.021258542115	0.001050420168	0.000153154510	0.000133665645	0.000001103383	0.000000160877	0.000000001159	0.000000000000
1237	0.280881346079	0.039447165288	0.001616814875	0.000227066569	0.000199104236	0.000001307045	0.000000183562	0.000000001057	0.000000000000
1608	0.406352841781	0.055040877341	0.001865671642	0.000252706991	0.000222560906	0.000001160244	0.000000157156	0.000000000722	0.000000000000
2090	0.523223309431	0.068440657883	0.001913875598	0.000250346081	0.000221387328	0.000000915730	0.000000119783	0.000000000438	0.000000000000
2717	0.505862720699	0.063974273048	0.001472211999	0.000186184292	0.000165281815	0.000000541852	0.000000068526	0.000000000199	0.000000000000
3532	0.734428323702	0.089897493776	0.001698754247	0.000207935539	0.000185258967	0.000000480961	0.000000058872	0.000000000136	0.000000000000
4591	0.830185236892	0.098458218222	0.001524722283	0.000180828847	0.000161656703	0.000000332111	0.000000039388	0.000000000072	0.000000000000
5968	1.150196429032	0.132295182536	0.001675603217	0.000192727284	0.000172846537	0.000000280765	0.000000032293	0.000000000047	0.000000000000
7758	1.451463099412	0.162057317612	0.001675689611	0.000187092434	0.000168301411	0.000000215995	0.000000024116	0.000000000028	0.000000000000
10085	1.952530840292	0.211798704572	0.001784828954	0.000193607421	0.000174661230	0.000000176979	0.000000019198	0.000000000018	0.000000000000
13110	2.320398376743	0.244738573945	0.001678108314	0.000176994537	0.000160107567	0.000000128002	0.000000013501	0.000000000010	0.000000000000
17043	2.873712200424	0.294936493245	0.001642903245	0.000168615396	0.000152920746	0.000000096398	0.000000009894	0.000000000006	0.000000000000
22155	3.697848412953	0.369569807708	0.001670051907	0.000166908076	0.000151742636	0.000000075380	0.000000007534	0.000000000003	0.000000000000
28801	4.674642274198	0.455255841494	0.001666608798	0.000162308332	0.000147904183	0.000000057866	0.000000005636	0.000000000002	0.000000000000
37441	5.792685499111	0.550085332649	0.001629229989	0.000154715032	0.000141297162	0.000000043515	0.000000004132	0.000000000001	0.000000000000
48673	7.690255241910	0.712530429948	0.001705257535	0.000157998382	0.000144600604	0.000000035035	0.000000003246	0.000000000001	0.000000000000
63274	9.497767308103	0.859119846084	0.001659449379	0.000150105372	0.000137653899	0.000000026226	0.000000002372	0.000000000000	0.000000000000
82256	12.635197034268	1.116421007656	0.001738475005	0.000153608211	0.000141137575	0.000000021135	0.000000001867	0.000000000000	0.000000000000
106932	15.889535400806	1.372159430725	0.001720719710	0.000148594765	0.000136782732	0.000000016092	0.000000001390	0.000000000000	0.000000000000
139011	149.379660482026	12.614066119687	0.012725611642	0.001074588777	0.000990913105	0.000000091544	0.000000007730	0.000000000001	0.000000000000
180714	25.279497580931	2.088408490014	0.001693283310	0.000139886769	0.000129212196	0.000000009370	0.000000000774	0.000000000000	0.000000000000
234928	31.939751148044	2.582652413668	0.001681366206	0.000135955489	0.000125784535	0.000000007157	0.000000000579	0.000000000000	0.000000000000
305406	117.503627658629	9.303977434594	0.004859105584	0.000384745642	0.000356516541	0.000000015910	0.000000001260	0.000000000000	0.000000000000
397027	52.669303708961	4.085501551085	0.001710211144	0.000132659249	0.000123109757	0.000000004308	0.000000000334	0.000000000000	0.000000000000
516135	85.144402683852	6.472829739634	0.002169974910	0.000164965373	0.000153310439	0.000000004204	0.000000000320	0.000000000000	0.000000000000
670975	86.162643502352	6.422146310825	0.001722865979	0.000128414089	0.000119506643	0.000000002568	0.000000000191	0.000000000000	0.000000000000
872267	183.494946159270	13.414500105973	0.002877559279	0.000210365572	0.000196034370	0.000000003299	0.000000000241	0.000000000000	0.000000000000
1133947	147.046006845825	10.547574697221	0.001807844635	0.000129676261	0.000120997163	0.000000001594	0.000000000114	0.000000000000	0.000000000000
1474131	179.884236229371	12.664711719696	0.001733224523	0.000122027307	0.000114001085	0.000000001176	0.000000000083	0.000000000000	0.000000000000
1916370	324.970165644709	22.464498736258	0.002453075346	0.000169575899	0.000158611427	0.000000001280	0.000000000088	0.000000000000	0.000000000000

XXI- Éléments du calculs de la complexité

XXI.1- Équation de récurrence

- Analyse de la complexité des fonctions non récursives est relativement simple.
→ cf. les règles simples de calcul vues plus haut.
- Cette analyse (& la preuve) devient plus simple avec les fonctions **récursives**.

Une définition nécessaire : on appelle une **récurrence** la forme :

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} + c = 0 \quad \text{où } c, k, a_i \text{ sont des constantes}$$

= l'équation de récurrence linéaire et homogène à coefficients constants.

Par exemple, dans le schéma **récuratif** :

```
def factorielle(n) :  
  if (n==1) : return 1  
  return n*factorielle(n-1)
```

L'équation de récurrence sera :

$$t_n - t_{n-1} - 1 = 0 \leftrightarrow T(n) = T(n-1) + 1$$

On retrouve :

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} + c = 0 \quad \text{où } c, k, a_i \text{ sont des constantes}$$

- Le but sera d'exprimer $T(n)$ sous forme de récurrence.
- Puis de résoudre cette équation.

XXI.2- Résolution de l'équation récurrente

- Pour "résoudre" une récurrence, on cherche **une forme fermée** (ou close).

→ Une **forme fermée** pour $T(n)$ est une équation pour $T(n)$ sans utiliser T .

- Par exemple, pour la spécification :

$$T(n) = T(n-1) + 1$$

$$T(1)=1$$

→ une forme fermée est **$T(n)=n$** .

- Pour cette résolution :

Il faut (au moins) une **condition initiale** pour trouver une telle solution.

Remarque : si absence de condition initiale, on aura une **famille** de formes closes.

→ Peu exploitable, voire non-exploitable.

→ Avec une condition initiale, une seule de ces formes satisfait la récurrence.

Pour l'exemple factoriel, selon la condition initiale :

→ $T(n)=T(n-1)+1$ avec $T(1)=0$ donne $T(n)=n-1$

→ $T(n)=T(n-1)+1$ avec $T(1)=1$ donne $T(n)=n$

→ $T(n)=T(n-1)+1$ avec $T(1)=2$ donne $T(n)=n+1$

Résultats obtenus par **substitution** ; voir pages suivantes.

Il existe plusieurs méthodes et techniques pour "résoudre" une récurrence.

I) Pour les cas simples :

- Par la définition de fonctions (voir TD)
- *Substitution ascendante*
- *Substitution descendante*
- Calculer, faire une hypothèse et vérifier ...
... avec une preuve simple pour les schémas non complexes via une :

II) Pour les cas moins simples, utiliser :

- Équation caractéristique
- Méthodes ad-hoc (Boîte à outils)
- Théorème ou **Méthode Principale** (ou MM : *master method*).

XXII- Cas simples : proposer et vérifier

Pour calculer la complexité dans les cas simples,

on peut faire une substitution **ascendante** ou **descendante**.

- Substitution **ascendante** :

Exemple factorielle où $T(n) = T(n-1) + 1$ (avec $T(1) = 1$)

→ $T(1) = 1$ supposons cette valeur arbitraire !

→ $T(2) = T(1) + 1 = 1 + 1 = 2$

....

→ $T(n-1) = \dots n-1$

→ $T(n) = T(n-1) + 1 \dots = (n-1) + 1 = n$ --> Semble être **$O(N)$**

La forme **fermée** pour la Factorielle sera **$T(n) = n$** .

Exemple chiffré : pour $N=5$ et la condition initiale $T(1)=1$

$$\rightarrow T(1)=1$$

$$\rightarrow T(2) = T(1) + 1 = 1+1 = 2$$

....

$$\rightarrow T(5) = T(4)+1 = 4+1=5$$

--> Cela suffit comme vérification

--> on peut au besoin faire une preuve par Induction

- Substitution descendante :

$$T(n) = T(n-1) + 1$$

$$= [T(n-2)+1]+1$$

$$= [[T(n-3)+1]+1]+1$$

...

$$= [...[[T(n-(n-1))+1]+1]...+1]+1 \quad \text{avec (n-1) fois '1'}$$

$$= T(n-(n-1)) + (n-1) = = n$$

→ La forme **fermée** pour la Factorielle sera **$T(n)=n$** .

Exemple chiffré : pour $N=5$ et condition initiale $T(1)=1$

$$\rightarrow T(5) = T(4)+1 = (T(3)+1)+1 = ... = (T(1)+1)+1...+1=1+1+1+1+1=5 \rightarrow O(N)$$

- La vérification des résultats pour ces cas simples peut être triviale.

Remarque : ces techniques basiques sont utilisées pour les cas simples.

Mais elles **ne marchent pas (ou sont fastidieuses)** pour tous les cas moins simples.

Exemple : pour

$$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2) \quad \text{et} \quad \text{Fib}(1..2) = 1.$$

- Dans tous les cas, il faudra apporter la preuve de nos calculs.
- Une preuve de la solution proposée nous dira si la solution est fausse (ou pas).

XXIII- Cas général : Preuve

On peut systématiser une preuve de l'hypothèse (une vérif.) de la complexité.

Comme toute preuve, celle de la complexité calculée peut être faite par différentes méthodes :

- *intuitivement (cas simples),*
- *"réécriture" (algébriquement),*
- *induction mathématique,*
- *etc...*

XXIII.1- Exemple de preuve : Hanoi

Action hanoi(entier n, Dep, Aux, Arr) :

Si ($n \leq 0$) rien

Sinon

hanoi($n-1$, Dep, Arr, Aux)

déplacer le disque sur Dep vers Arr

hanoi($n-1$, Aux, Dep, Arr)

$$\bullet T(n) = 2.T(n-1) + 1 = 2.(2.T(n-2) + 1) + 1 = 2.(2.(2.T(n-3) + 1) + 1) + 1 = \dots$$

$$= 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^0 \quad \text{substitutions ascendantes}$$

● Hypothèse : on a la forme close $T(n) = 2^n - 1$ avec $T(1) = 1$ (et $t(0) = 0$) : ok ?

Vérification : $T(n) = 2.T(n-1) + 1$ posons $S(n) = T(n) + 1$

$$S(n) = T(n) + 1 = [2.T(n-1) + 1] + 1 = 2[T(n-1) + 1] \quad \text{et donc } S(n) = 2.S(n-1), n > 0$$

On voit clairement que $S(n) = 2^n$ d'où $T(n) = 2^n - 1$.

XXIII.2- Exemple de preuve : recherche Dichotomique

Renvoie **-1** si $X \notin T$, sinon renvoie l'indice I tel que $T[I]=X$

```

fonction Recherche_binaire(indice Inf, indice Sup, val X, tableau T) =
  Si (Inf > Sup) Alors -1          // par convention, '-1' vaut  $X \notin T$ 
  milieu = (Inf+Sup)/2
  Si (X=T[milieu]) Alors milieu
  Sinon Si (X < T[milieu]) Alors Recherche_binaire(Inf, milieu-1, X, T)
        Sinon Recherche_binaire(milieu+1, Sup, X, T)

```

- On a (hyp. $n=2^k$) $T(1)=1$, $T(2)=T(1)+1 = 2$,
 $T(4)=T(2)+1=3$, $T(8) = T(4)+1=4$, ...
 $T(n) = T(n/2) + 1$

- On "devine" $T(n) = \lg(n) + 1$ pour $n=2^k$

Est-ce vrai ?

- **Vérification** intuitive : si l'hypothèse $\lg(n)$ est fondée, on aura

$$T(n) = T(n/2) + 1 = [\lg(n/2) + 1] + 1 = [\lg(n) - \lg(2) + 1] + 1 = \lg(n) + 1 \quad \leftarrow \text{Yes !}$$

XXIII.3- Preuve par Induction

- A partir de l'équation de récurrence
- Simplification des écritures : on note $T(n)$ par t_n
- Exemple :

```
def fact(n) :  
  if (n == 0) : return 1  
  return  $n * fact(n - 1)$ 
```

Si t_n = nombre de multiplications nécessaires pour un n donné (= nbr d'appels),

Alors on a $t_n = t_{n-1} + 1$

t_{n-1} est la valeur de t dans les appels récursifs et 1 le coût de la multiplication

$\Rightarrow t_n = t_{n-1} + 1$ est l'équation de récurrence recherchée.

La **condition initiale** (nécessaire à la résolution) est ici $t_0 = 0$

XXIII.3.1- Exemple 1 : Factorielle

```
def fact(n) :  
  if (n == 0) : return 1  
  return  $n * \text{fact}(n - 1)$ 
```

On décide $t_0=1$ mais $t_0=0$ ne modifiera la complexité (linéaire) de fact.

→ cette constante sera bien négligeable devant la limite de n .

On peut alors calculer t_n pour différentes valeurs de n (par une subs. desc.) :

$$t_1 = t_{1-1} + 1 = t_0 + 1 = 1,$$

$$t_2 = t_{2-1} + 1 = t_1 + 1 = 2,$$

.....

On constate (hypothèse) :

$t_n = \dots = n$ appelée **solution** de l'équation de récurrence

👉 **Maintenant, prouvons que $t_n=n$ est bien une solution (?)**

Vérification par Induction :

Base de l'induction :

$$t_0=0$$

Hypothèse d'induction : supposons avoir $t_n=n$

Étape d'induction : démontrer, par induction que : $t_{n+1} = n+1$

Facile ! : il existe plusieurs autres "preuves" triviales

selon l'équation $t_n = t_{n-1} + 1 \rightarrow T_{(n+1)} = T_{(n+1)-1} + 1 = T_n + 1$ CQFD !

Rappel : une induction ne peut pas trouver une solution mais permet de vérifier si une solution proposée en est une.

Notons cependant que l'**induction constructive** peut aider à trouver une solution.

XXIII.3.2- Exemple 2

$$t_n = t_{n/2} + 1 \text{ pour } n > 1, n=2^k \text{ (une puissance de 2)}$$
$$t_1 = 1$$

Quelques valeurs de t_n :

$$t_1 = 1$$

$$t_2 = t_{2/2} + 1 = t_1 + 1 = 2$$

$$t_4 = t_{4/2} + 1 = t_2 + 1 = 3$$

$$t_8 = 4$$

$$t_{16} = 5 \quad \dots$$

On propose une solution (constatée, flairée !) : $t_n = \log n + 1$.

Verifions :

Base : $t_1 = 1$

Hypothèse : $t_n = \log n + 1$

Étape d'induction : $t_{2n} = \log(2n) + 1$? si $n = 2^k$: la prochaine val. pour n est $2n$

--> De l'algorithme lui-même, on a : $t_{2n} = t_{(2n)/2} + 1 = t_n + 1$

--> et de l'hypothèse, on a : $t_n = \log(n) + 1$

$$\text{D'où } t_{2n} = t_{n+1} = \log(n) + 1 + 1 = \log(n) + \log(2) + 1 = \log(2n) + 1 \quad \text{CQFD.}$$

XXIII.3.3- Exemple 3 (à démontrer)

Soit la récurrence :

$$\begin{aligned} t_n &= 7t_{n/2} \quad \text{pour } n > 1, n \text{ une puissance de } 2 \\ t_1 &= 1 \end{aligned}$$

Quelques valeurs de t_n :

$$\begin{aligned} t_1 &= 1 & t_2 &= 7t_{2/2} = 7t_1 = 7 & t_4 &= 7t_{4/2} = 7t_2 = 7^2 \\ t_8 &= 7^3 & t_{16} &= 7^4 & \dots \end{aligned}$$

On constate une solution :

$$t_n = 7^{\log n} \quad \text{à démontrer par l'induction Math.}$$

Remarque : on sait $7^{\log n} = n^{\log 7}$

$$\rightarrow t_n = n^{\log 7} \simeq n^{2.81}$$

XXIII.3.4- Exemple 4 (cas défavorable)

$$\begin{aligned} t_n &= 2t_{n/2} + n - 1 \quad \text{pour } n > 1, n \text{ une puissance de } 2 \\ t_1 &= 0 \end{aligned}$$

On a quelques valeurs de t_n :

$$\begin{aligned} t_1 &= 0 & t_2 &= 2t_{2/2} + 2 - 1 = 2t_1 + 1 = 1 & t_4 &= 2t_{4/2} + 4 - 1 = 2 + 4 - 1 = 5 \\ t_8 &= 17 & t_{16} &= 49 & & \dots \end{aligned}$$

Il n'y a pas de solution candidate comme dans les exemples précédents.

--> On ne peut donc pas vérifier celle-ci puisque l'induction sert à vérifier si une solution candidate est juste.

NB : la complexité sera de la forme $t_n = n \cdot \log(n) - (n-1) = n \cdot \log(n) - n + 1$ (voir + loin)

XXIII.3.5- Un autre cas défavorable : Fib

Rappel Fib : $t_n = t_{n-1} + t_{n-2}$

$$t_0 = 0$$

$$t_1 = 1$$

On peut écrire la première équation par :

$$t_n - t_{n-1} - t_{n-2} = 0$$

- On est vite bloqué dans l'approche par équation de récurrence !
- Au mieux, une forme exponentielle se dégage (si on fait des substitutions).

La suite Fibonacci est définie par une **équation linéaire homogène**.

Comment résoudre ce type d'équation ?

- **On va le** voir après quelques exemples introductifs simples.

XXIV- Résolution de l'équation caractéristique

Supposons l'équation suivante (*l'exemple Fib est traité à la suite*) :

$$t_n - 5t_{n-1} + 6t_{n-2} = 0 \quad \text{pour } n > 1$$

$$t_0 = 0$$

$$t_1 = 1$$

Si l'on note $t_n = r^n$

On aura alors $t_n - 5t_{n-1} + 6t_{n-2} = r^n - 5r^{n-1} + 6r^{n-2}$

Dans ce cas, $t_n = r^n$ est une solution à cette récurrence si r est la racine de l'équation

$$r^n - 5r^{n-1} + 6r^{n-2} = 0 \quad \dots$$

Résolution :

On a $r^n - 5r^{n-1} + 6r^{n-2} = r^{n-2}(r^2 - 5r + 6)$

les racines : $r=0$ et celles de $r^2 - 5r + 6 = 0$

qui sont obtenues par $r^2 - 5r + 6 = (r-3)(r-2) = 0$

Les racines de l'équation : $r=0, r=3$ et $r=2$

C'est à dire :

$t_n=0$, $t_n = 3^n$ et $t_n = 2^n$ sont tous trois solutions à l'équation Réc.

../..

On note :

Si 0, 3^n et 2^n sont des solutions,

Alors toute solution de la forme générale (= une combinaison linéaire des t_n) :

$$t_n = c_1 3^n + c_2 2^n \quad (c_1 \text{ et } c_2 \text{ sont des constantes arbitraires})$$

Est aussi une solution (voir le théorème-1 suivant).

N.B. : On peut démontrer que ce sont les seules solutions.

- On a une infinité de solutions (suivant c_1 et c_2) mais laquelle choisir ?

--> Ce choix est déterminé par les conditions initiales :

$$t_0=0 \quad t_1=1. \quad \text{D'où}$$

$$t_0 = c_1 3^0 + c_2 2^0 = 0$$

$$t_1 = c_1 3^1 + c_2 2^1 = 1$$



L'ensemble se simplifie par :

$$c_1 + c_2 = 0$$

$$3 c_1 + 2 c_2 = 1$$

On obtient :

$$c_1=1 \text{ et } c_2 = -1$$

La solution à l'équation de récurrence sera :

$$t_n = 1 (3^n) - 1 (2^n) = 3^n - 2^n$$

N.B. : avec les conditions initiales $t_0=1$ et $t_1=2$, on aurait la solution $t_n = 2^n$

Cela veut dire que **la récurrence représente une classe de fonctions.**

NB : l'équation ($r^2 - 5r + 6 = 0$) est appelée l'équation caractéristique de la récurrence.

Cette équation est définie comme suit



XXIV.1- Théorème 1

Théorème 1 (preuve dans le support "long" du cours):

- L'équation caractéristique de l'équation de récurrence linéaire et homogène à

coefficients constants $a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = 0$

est définie par : $a_0 r^k + a_1 r^{k-1} + \dots + a_k r^0 = 0$

avec k racines distinctes $r_1, r_2 \dots r_k$

- L'unique solution à la récurrence sera alors (c_i constantes arbitraires) :

$$t_n = c_1 r_1^n + c_2 r_2^n + \dots + c_k r_k^n$$

Par exemple :

L'équation caractéristique pour $5t_n - 7t_{n-1} + 6t_{n-2} = 0$

est $5r^2 - 7r + 6 = 0$ avec un ordre $k=2$.

XXIV.1.1- Exemple

Résoudre l'équation de récurrence :

$$t_n - 3t_{n-1} - 4t_{n-2} = 0 \quad \text{pour } n > 1$$

$$t_0 = 0$$

$$t_1 = 1$$

L'équation caractéristique $r^2 - 3r - 4 = 0 = (r - 4)(r + 1)$

D'où $r = 4$ et $r = -1$

→ N.B. : la racine $r=0$ n'apporte aucune information.

La solution générale : $t_n = c_1 4^n + c_2 (-1)^n$

Trouver les constantes c_1 et c_2 en utilisant les conditions initiales :

--> D'où $c_1 = 1/5$ et $c_2 = -1/5$

On obtient la solution finale $t_n = 1/5 4^n - 1/5 (-1)^n$

XXIV.1.2- Une autre définition de Fib

Nous avons vu plus haut une autre définition récursive pour Fib :

$$\text{fib}(2n) = \text{fib}(n)^2 + 2\text{fib}(n).\text{fib}(n-1) \quad \text{pour tout entier } n \text{ pair}$$

$$\text{fib}(2n+1) = \text{fib}(n)^2 + \text{fib}(n+1)^2 \quad \text{pour tout entier } n \text{ impair}$$

Pour Simplifier, **on considère le premier cas.** Il est représentatif.

Le 2^e cas a un terme supplémentaire.

$$T_{2n} = (T_n)^2 + 2T_n.T_{n-1}$$

→ Hyp : le coût de $(T_n)^2$ est simplement le coût de $T_n + X$ (X disparaîtra !)

$$T_{2n} = T_n + X + 2T_n.T_{n-1}$$

$$D'où \rightarrow r^{2n} = r^n + X + 2r^n r^{n-1} \quad \dots/..$$

$$\rightarrow r^n \cdot r^n = r^n + X + 2r^n r^{n-1}$$

$$\rightarrow r^n \cdot r^n - r^n - X - 2r^n r^{n-1} = 0$$

$$\rightarrow r^n (r^n - X/r^n - 2 r^{n-1}) = 0$$

$$\rightarrow r = 0 \text{ est une racine} \quad \text{et } r^n - X/r^n - 2 r^{n-1} = 0$$

$$\rightarrow r^{n-1} (r - X/r^{2n-1} - 2) = 0$$

$$\rightarrow r = 0 \text{ (on le sait déjà), la quantité } X/r^{2n-1} \text{ est } \underline{\text{négligeable}}$$

$$\rightarrow r - 2 = 0$$

$$\rightarrow r = 2$$

Pour CETTE VERSION de Fib, on a une complexité $O(2^N)$.

Comparer $r=2$ à $3/2 < r < 5/3$ de ci-dessus.

XXV- Cas de racines multiples : théorème-2

- Le théorème-1 ci-dessus indique que les k racines de l'équation caractéristique sont **distinctes**.

Comment utiliser ce théorème dans le cas d'une équation caractéristique

de la forme $(r - 1)(r - 2)^3 = 0$

C-à-d. le terme $(r-2)$ est à la puissance **3** ?

- Dans ce cas, la racine $r=2$ est appelée la **racine de multiplicité 3** de l'équation.
- Le théorème suivant permet alors à une racine d'avoir une multiplicité.

XXV.1- Théorème 2

Si r est une racine de multiplicité m de l'équation caractéristique.

Alors :

$$t_n = r^n, \quad t_n = n r^n, \quad t_n = n^2 r^n, \quad t_n = n^3 r^n, \quad \dots, \quad t_n = n^{m-1} r^n$$

sont toutes des solutions à la récurrence.

C'est-à-dire, pour chacune de ces solutions, un terme est inclus dans la solution générale de la récurrence.

XXV.1.1- Exemple 1

Soit la récurrence :

$$t_n - 7 t_{n-1} + 15 t_{n-2} - 9 t_{n-3} = 0 \quad \text{pour } n > 2$$

$$t_0 = 0$$

$$t_1 = 1$$

$$t_2 = 2$$

● L'équation caractéristique $r^3 - 7 r^2 + 15 r - 9 = 0$

$$\rightarrow r^3 - 7 r^2 + 15 r - 9$$

$$= (r-1)(r-3)^2 = 0 \quad \text{où la racine } 3 \text{ est de multiplicité } 2.$$

→ La solution générale sera alors :

$$t_n = c_1 1^n + c_2 3^n + c_3 n 3^n$$

On introduit les termes 3^n et $n 3^n$ car la racine 3 est de multiplicité 2.

Les conditions initiales (à l'aide de t_0 , t_1 et t_2) donnent

$$c_1 = -1, \quad c_2 = 1 \quad \text{et} \quad c_3 = -1/3$$

d'où :

$$\begin{aligned} t_n &= (-1) 1^n + (1) 3^n + (-1/3) n 3^n \\ &= -1 + 3n - n3^{n-1} \end{aligned}$$

XXV.1.2- Exemple 2

Soit la récurrence :

$$t_n - 5 t_{n-1} + 7 t_{n-2} - 3 t_{n-3} = 0 \quad \text{pour } n > 2$$

$$t_0 = 0$$

$$t_1 = 2$$

$$t_2 = 3$$

On obtient l'équation caractéristique

$$r^3 - 5 r^2 + 7r - 3 = (r-3)(r-1)^2 = 0$$

→ La racine 1 est de multiplicité 2.

→ La solution générale sera alors: $t_n = c_1 3^n + c_2 1^n + c_3 n 1^n$

→ Conditions initiales : $c_1 = 0$, $c_2 = 1$ et $c_3 = 1$

$$\begin{aligned} \rightarrow t_n &= 0 \cdot 3^n + 1 \cdot 1^n + 1 \cdot n \cdot 1^n \\ &= 1 + n \end{aligned}$$

XXVI- Récurrence linéaire non homogène

Le cas où le terme à droite de l'équation n'est pas = 0 mais $f(n)$.

Une récurrence de la forme $a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = f(n) \neq 0$
où a_i sont des constantes et $f(n)$ une fonction non nulle appelée une
équation de récurrence linéaire non Homogène à coefficient constant.

- **Il n'y a pas de méthode générale connue** pour résoudre cette équation.

Par contre, on propose une solution pour une forme particulière où :

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = b^n p(n)$$

Deux exemples :

$$t_n - 3t_{n-1} = 4^n$$

$$t_n - 3t_{n-1} = 4^n (8n+7)$$

XXVI.1- Théorème 3

Une équation de récurrence linéaire non Homogène à coefficient constant de la forme

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = b^n p(n)$$

peut être transformée en $(a_0 r^k + a_1 r^{k-1} + \dots + a_k) (r-b)^{d+1} = 0$

où d = le degré de $p(n)$ et $P(n)$ est un polynôme en $\mathbb{N} \rightarrow \mathbb{R}$

→ N.B. : noter l'origine de k (en rouge), b sera une nouvelle racine.

Résolution : cette équation est alors composée de 2 parties :

- 1- l'équation caractéristique pour la partie homogène (premier terme à gauche)
- 2- un terme obtenu de la partie non homogène de la récurrence.

⇒ S'il y a plus d'un terme de la forme $b^n p(n)$ à droite, chacun contribuera à un terme de l'équation caractéristique.

XXVI.1.1- Exemple 1

Soit

$$t_n - 3 t_{n-1} = 4^n (2n+1) \quad \text{pour } n > 1$$

$$t_0 = 0$$

$$t_1 = 12$$

Démarche :

1- On obtient l'équation caractéristique pour la **partie homogène**

$$t_n - 3 t_{n-1} = 0 \quad \rightarrow \quad r^1 - 3 = 0$$

2- On obtient de la partie **non homogène** de la forme $b^n P(n)$:

$$4^n (2n+1) \quad \text{où} \quad b = 4 \text{ et } d = 1 (\text{degré de } p(n))$$

Le terme de la partie non homogène sera : $(r - b)^{d+1} = (r - 4)^{1+1} \quad \dots / \dots$

3- On applique le théorème 3 pour obtenir une équation caractéristique :

L'équation caractéristique sera $(r - 3)(r - 4)^2$

4- La résolution de l'équation $(r - 3)(r - 4)^2 = 0$

donne les racines $r=3$ et $r=4$ $r=4$ est de **multiplicité 2**

5- Par le théorème 2 : $t_n = c_1 3^n + c_2 4^n + c_3 n4^n$.

6- Calcul des constantes : ici, on n'a que deux cas basiques (t_0 et t_1)

→ Mais on peut calculer t_2 par $t_2 - 3 t_1 = 4^2 (2 * 2 + 1)$

Sachant $t_1 = 12$, on obtient $t_2 = 116$.

→ La forme générale de la solution sera : $t_n = 20(3^n) - 20(4^n) + 8 n 4^n$.

XXVI.1.2- Exemple 2

Soit la récurrence

$$t_n = 3 t_{n-1} + 2^n$$

$$t_0=1$$

Rappel :

Une équation de récurrence linéaire non Homogène à coefficient constant de la forme

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = b^n p(n)$$

peut être transformée en $(a_0 r^k + a_1 r^{k-1} + \dots + a_k) (r-b)^{d+1} = 0$

où d = le degré de $p(n)$ et $P(n)$ est un polynôme en $\mathbb{N} \rightarrow \mathbb{R}$

→ la racine de la partie homogène est $r=3$

→ la partie droite est de la forme $(r-2)^{0+1}$ et $p(n)=1$ d'où le degré $d = 0$.

La complexité sera de la forme $T(n) = c_1 3^n + c_2 2^n$

Et les calculs donneront $c_1=3$ et $c_2 = -2$

d'où $T(n) = 3 \cdot 3^n - 2 \cdot 2^n$.

Remarque :

Si l'équation de récurrence (non homogène) est de la forme

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = \text{Cste (une constante)}$$

Alors **Cste** représente $P(n)$ et non b^n

Car si $P(n)$ est constante, alors $n=0$.

Dans le cas contraire, on introduirait une racine supplémentaire $r=b$

→ incohérent !

Pour un exemple de ce cas : voir les tours de Hanoi.

XXVI.1.3- Exercice 1

Soit la somme $f(n) = \sum_{j=0}^n j^3 = 1^3 + 2^3 + 3^3 + \dots + n^3$

dont l'équation de récurrence est donnée par :

$$t_0=0, \quad t_1=1, \quad t_2=9, \quad t_3=36, \quad t_4=100$$

$$t_n = t_{n-1} + n^3$$

On est dans le cas d'une équation de récurrence non homogène car $t_n - t_{n-1} = n^3$

- Calculer les détails cette complexité.

- Indication :

le résultat devrait être $t_n = c_1 + c_2 n + c_3 n^2 + c_4 n^3 + c_5 n^4$

→ Calculer les coefficients et en déduire une relation entre les fonctions $f(n)$ des deux exemples précédents.

XXVI.1.4- Exercice 2

Calculer la complexité de

$$t_n = 7 t_{n-1} - 10 t_{n-2} + (2n+5) 3^n$$

et déduire qu'elle est $\Theta(5^n)$.

Indication :

dans la partie non homogène, on a $b^n = 3^n$ et $p(n)=2n+5$

d'où la partie homogène sera multipliée par $(r-3)^{1+1}$.

XXVII- Boite à outils (Cook Book)

- La généralisation de certains de ces calculs a donné lieu à une boite à outils.

Il s'agit d'une méthode dite "principale" (*main method*) qui permet de calculer (rapidement) une complexité Θ (Théta).

- Elles peuvent remplacer le calcul dans le cas d'équation non-homogènes.

XXVII.1- Méthode principale (MM) et la famille Thêta

- Cook-Book des cas **fréquents** calculés par différentes techniques.
- Utilisé (souvent) pour trouver la complexité Θ (mais aussi O et Ω)
- Permet d'éviter des calculs détaillés et donne directement la solution.

- Le but (de MM) : résoudre la **réurrence** de la forme générique :
 - $T(n) = a T(n/b) + cn^k$ avec $n > 1$, $n = b^m$ (n est une puissance de b , $m \geq 0$)
 - $T(1) = d$ (condition initiale)
 - $b \geq 2$, $k \geq 0$ sont des constantes entières,
 - $a > 0$, $c > 0$, $d \geq 0$ des constantes

Suivant l'algorithme étudié, on pioche dans le cook-book et on choisit un des cas :

I	$T(n) = \Theta(n^k)$	si $a < b^k$
II	$T(n) = \Theta(n^k \cdot \lg(n))$	si $a = b^k$
III	$T(n) = \Theta(n^{\lg_b(a)})$	si $a > b^k$

→ Remarque : fournit une complexité Theta

→ Pour *big-Oh* / Ω , remplacer " $T(n) =$ " par " $T(n) \leq$ " ou " $T(n) \geq$ "

XXVII.1.1- Exemples

Il suffit d'identifier simplement chaque cas de figure.

- $T(n) = 16 T(n/4) + 5n^3$

- $a=16, b=4, k=3$ et $16 < 4^3$ donc **cas (I)** : $T(n) = \Theta(n^k)$ si $a < b^k$

- $T(n) = \Theta(n^k) = \Theta(n^3)$

- $T(n) = 2 T(n/2) + n$

- $a=2, b=2, k=1$ et $2=2^1$ donc **cas (II)** : $T(n) = \Theta(n^k \cdot \lg(n))$ si $a = b^k$

- $T(n) = \Theta(n^k \cdot \lg(n)) = \Theta(n^1 \cdot \lg(n))$

- $T(n) = 8 T(n/2) + n$

- $a=8, b=2, k=1$ et $8 > 2^1$ donc **cas (III)** : $T(n) = \Theta(n^{\lg_b(a)})$ si $a > b^k$

- $T(n) = \Theta(n^{\lg_b(a)}) = \Theta(n^{\lg(8)}) = \Theta(n^3)$

XXVII.2- Méthode principale généralisée (MMG)

- Pour résoudre la récurrence : si on a les conditions suivantes
 - $T(n) = a T(n/b) + f(n)$ avec $n > 1$, $n = b^m$ (n est une puissance de b)
 - $T(1) = d$ (cond. initiale)
 - $b \geq 2$ une constante entière et $a > 0$, $d \geq 0$ des constantes

Suivant l'algorithme que l'on étudie, on choisit un des cas :

I') $T(n) = (f(n))$, si $f(n) = (n \lg_b (a + \varepsilon))$ et $a.f(n/b) \leq cf(n)$ pour $c \leq 1$ cste et $n \rightarrow \infty$

II') $T(n) = (n \lg_b (a) \cdot \lg(n))$, si $f(n) = (n \lg_b (a))$

III') $T(n) = (n \lg_b (a))$, si $f(n) = O(n \lg_b (a - \varepsilon))$ \lg_b : log en base b

Avec $\varepsilon > 0$, \lg_b est important car le paramètre b est présent dans $T(n)$

Rappel :

I') $T(n) = (f(n))$, si $f(n) = (n \lg_b (a+\varepsilon))$ et $a.f(n/b) \leq cf(n)$ pour $c \leq 1$ cste et $n \rightarrow \infty$

II') $T(n) = (n \lg_b (a) \cdot \lg(n))$, si $f(n) = (n \lg_b (a))$

III') $T(n) = (n \lg_b (a))$, si $f(n) = O(n \lg_b (a-\varepsilon))$ \lg_b : log en base b

Avec $\varepsilon > 0$, \lg_b est important car le paramètre b est présent dans $T(n)$

- Le cas (I') peut être ré-écrite en $n^{\lg_b (a+\varepsilon)} = O(f(n))$.

Car si $f(n) = \Omega(g(n)) \rightarrow g(n) = O(f(n))$

- Ce cas (I') est souvent ignoré à cause de sa condition complexe !

XXVII.2.1- Exemples

A partir de la récurrence $T(n)$, identifier $f(n)$ et extraire la complexité.

● $T(n) = 2 T(n/2) + n$

○ On souligne les cond. de la récurrence : $a = 2, b=2, f(n) = n, \lg_b(a) = \lg(2)$

→ **cas (II')** : $f(n) = n = n^1 = \Theta(n^{\lg_b(a)})$

II') → $T(n) = \Theta(n^{\lg_b(a)} \cdot \lg(n)) = \Theta(n^{\lg(2)} \cdot \lg(n)) = \Theta(n \cdot \lg(n))$

● $T(n) = 8 T(n/2) + n$

○ On a $a = 8, b = 2, f(n) = n, \lg_b(a) = \lg(8) = 3$ et $2=2^1$

→ **cas (III')** : $f(n) = n = O(n^{3-\varepsilon})$ avec $\varepsilon = 1$

III') → $T(n) = \Theta(n^{\lg_b(a)}) = \Theta(n^{\lg(8)}) = \Theta(n^3)$

Remarques :

- Les 3 cas de MM (I,II,III) sont des cas particuliers de MMG (I',II', III').
- P. ex. : Si $T(n) = a T(n/b) + cn^k$ satisfait la condition de (II) : $a=b^k$

Alors on peut écrire (en utilisant le cas II') :

$$a=b^k$$

$$\lg_b(a) = \lg_b(b^k) = k$$

$$\rightarrow f(n) = cn^k = cn^{\lg_b(a)} \in \Theta(n^{\lg_b(a)}) \quad \text{cf. la condition du cas général (II')}$$

XXVIII- Complément technique

Ci-dessous, quelques exemples qui montrent différentes techniques employées pour le calcul de la complexité.

XXVIII.1- Changement de variable (cas homogène)

Nous avons vu un premier exemple de changement de variable :

Preuve par induction de la complexité de Hanoi.

D'autres exemples ci-après.

XXVIII.1.1- Exemple : Recherche dichotomique

Pour un cas dichotomique tel que :

$$T(n) = T(n/2) + 1 \quad n > 1 \text{ et } n = 2^k$$

$$T(1) = 1.$$

● On pose $n=2^k \rightarrow k = \log n$ et donc $T(2^k) = T(2^k/2) + 1 = T(2^{k-1}) + 1$

● Poser ensuite $T(2^k) = t_k$ d'où $t_k = t_{k-1} + 1$

--> On sait résoudre cette équation par le théorème 3 et obtenir

$$t_k = c_1 + c_2 k \quad \text{et donc} \quad T(2^k) = c_1 + c_2 k$$

● Ensuite, on substitue n pour 2^k et $\log(n)$ pour $k \rightarrow T(n) = c_1 + c_2 \log(n)$

● On obtiendra $c_1=c_2=1$ par ailleurs et au final :

$$T(n) = 1 + \log(n)$$

XXVIII.1.2- Exemple : Tri Fusion

On a	$T(1)=T(0)=0$
	$T(n) = 2 T(n/2) + n$

- Soit $n=2^k$ d'où $k=\log(n)$

--> On aura $T(2^k) = 2 T(2^k/2) + 2^k$. -> $T(2^k) = 2 T(2^{k-1}) + 2^k$.

- Soit $t_k = T(2^k)$ pour tout entier k .

--> D'où l'équation de récurrence $t_k = 2t_{k-1} + 2^k$

→ l'éq. caractéristique $r^k = 2 r^{k-1} + 2^k$.

- En accord avec le théorème 3 (non homogène), on aura $(r^k - 2 r^{k-1})(r-2)=0$.

→ qui donne $(r-2)(r-2) :$ une racine double $r=2$.

● La forme générale de la solution $t_k = c_0 + c_1 2^k + c_2 k 2^k$.
--

→ Et donc $t_n = c_1 n + c_2 n \log(n)$.

- Les calculs des coefficients donnent $c_1=0$ et $c_2=1/2$.

→ D'où $t(n) = 1/2 n \log(n) = O(n \cdot \log(n))$

N.B. : une autre technique ad-hoc pour ce même exemple :

On peut utiliser une soustraction pour aboutir aux mêmes résultats :

- On avait (de la solution précédente) :

$$(1) \quad t_k = 2t_{k-1} + 2^k.$$

Et on pose

$$(2) \quad t_{k-1} = 2t_{k-2} + 2^{k-1}.$$

On multiplie (2) par 2 :

$$(3) \quad 2t_{k-1} = 4t_{k-2} + 2^k.$$

On soustrait (1) et (3) :

$$t_k = 4t_{k-1} - 4t_{k-2}.$$

$$\rightarrow r^k - 4r^{k-1} + 4r^{k-2} = 0$$

$$\rightarrow r^{k-2} (r^2 - 4r + 4) = 0 \quad \rightarrow \quad (r^2 - 4r + 4) = (r-2)(r-2).$$

Les racines : $r=0$, $r=2$ (racine double)

et la forme générale : $t_k = c_0 + c_1 2^k + c_2 k 2^k$.

Le reste est identique à la solution précédente.

XXVIII.2- Changement de variable dans une cas non homogène

La récurrence de la forme

$$t_n = a t_{n/b} + f(n)$$

est résolu pour obtenir $T(n)$ en posant

$$S(n) = T(b^k) \quad \text{et donc}$$

$$s_k = t_{b^k} = a t_{b^k/b} + f(b^k) = a t_{b^{k-1}} + f(b^k) = a s_{k-1} + f(b^k).$$

Ce qui nous ramène aux cas précédents.

Exemple : ../..

XXVIII.2.1- Exemple

Soit $t_n = 3 t_{n/2} + n \lg(n)$ pour $n > 1$ et $t_1 = 2$

On pose $S(n) = T(2^k)$ avec $n = 2^k \rightarrow k = \lg(n)$ et donc

$$s_k = t_{2^k} = 3 t_{2^{k-1}} + 2^k \cdot \lg(2^k) = 3 t_{2^{k-1}} + 2^k \cdot k = 3 s_{k-1} + k \cdot 2^k$$

La résolution de ce cas non homogène donnera :

$$(r-3)(r-2)^{1+1}=0 \quad \text{et} \quad \text{donc} \quad s_k = c_1 3^k + c_2 \cdot 2^k + c_3 k \cdot 2^k.$$

- Pour les conditions initiales sur S :

$$S(0) = T(2^0) = T(1) = 2$$

$$S(1) = T(2^1) = T(2) = 8 \quad \text{que l'on calcule à l'aide du système initial}$$

$$S(2) = T(2^2) = T(4) = 32$$

- On peut maintenant calculer les constantes et obtenir :

$$S(k) = 8 \cdot 3^k - 6 \cdot 2^k - 2 \cdot k \cdot 2^k.$$



On s'occupe maintenant de calculer $T(n)$:

Avec $S(n) = T(2^k)$ et $k = \lg(n)$, on a :

$$T(n) = S(\lg(n))$$

$$T(n) = 8 \cdot 3^{\lg(n)} - 6 \cdot 2^{\lg(n)} - 2 \cdot \lg(n) \cdot 2^{\lg(n)}.$$

$$\text{N.B. : } 3^{\lg(n)} = (2^{\lg(3)})^{\lg(n)} = (2^{\lg(n)})^{\lg(3)} = n^{\lg(3)}.$$

● La forme finale de la complexité sera donc $T(n) = 8n^{\lg(3)} - 6n - 2n\lg(n)$

→ Sachant que $1 < \lg(3) < 2$ (car $2^1 < 3 < 2^2$)

on a une complexité $O(n^2)$ pour $n = 2^k$.

Une remarque générale :

Une règle appelée la *règle de lissage* permet d'affirmer ceci.

si $T(n) = \Theta(f(n))$ pour n une puissance de $b \geq 2$

et si $f(n)$ est *nice* polynomiale

--> $f(n)$ est dite *nice* si elle n'est ni *exponentielle* ni *factorielle*

alors $T(n)$ est réellement $\Theta(f(n))$ pour toute valeur de n

Dans le cas de l'exemple présent, $n > 1$.

XXIX- Cas particulier : Racines imaginaires !

Il arrive (rarement) que l'on ait besoin de racines imaginaires.

XXIX.1- Un exemple

Soit $U_n = U_{n-1} - 2 U_{n-2}$

$$U_0 = \dots$$

Pas très simple : les racines seront complexes.

On obtient (avec l'aide de Maple) : $U_n = \left(\frac{U_0}{2} - i \left(\frac{2u_1 - u_0}{2\sqrt{7}} \right) \right) r_1^n + \left(\frac{U_0}{2} + i \left(\frac{2u_1 - u_0}{2\sqrt{7}} \right) \right) r_2^n$

La résolution donnera $r_1, r_2 = \frac{1 \pm i\sqrt{7}}{2}$

XXIX.2- Un autre exemple

● Soit $t_n = 2 t_{n-1} - 2 t_{n-2}$, $t_0=0, t_1 = 2$

Pour le cas $t_n = 2 t_{n-1} - 2 t_{n-2}$, dont l'équation caractéristique est $r^2-2r+2 = 0$

→ les racines seront $r=1+i$ et $r=1-i$ (et $r=0$ par ailleurs).

→ dans ce cas, on aura $T(n) = c_1 (1+i)^n + c_2 (1-i)^n$.

● On obtient des conditions initiales $c_1+c_2=0$ et $c_1-c_2=-2i$

→ et donc $T(n) = i \cdot [(1-i)^n - (1+i)^n]$

→ Cette fonction oscille entre $\sqrt{2^n}$ et $-\sqrt{2^n}$ avec une période de 4 ;

en particulier, on a $T(4n) = 0$ car $(1-i)^4=(1+i)^4 = -4$

N.B. : la présence de racine imaginaire laisse supposer un terme (ici la complexité) négatif, mais il se peut que $T(n) > 0$.

P. Ex. : si les racines d'une équation sont 2, $1+i$ et $1-i$ on aura une complexité oscillant autour de $c2^n$ (supposons des conditions initiales ad-hoc) qui sera toujours positif.

xxx- Table des matières

I- Quelques références bibliographiques	2
II- Objectifs du cours et éléments abordés	3
III- Pourquoi faut-il des algorithmes ?	5
IV- Qu'est-ce qu'un algorithme ? Exemples	6
IV.1- Exemple :Tours de HANOI	7
IV.2- Exemple : mariages stables (stable matching)	8
V- Ce que l'on peut attendre des algorithmes	9
VI- Autres remarques	10
VII- Idée de la complexité	12
VIII- Complexité et Contrôle	14
VIII.1- Exemple 1 : Comparatif recherche d'une "vis"	16
VIII.2- Exemple 2 : séquence de Fibonacci	17
VIII.3- Exemple 3 : Comparatif Fibonacci	18
IX- Tableau des croissances relatives	20
IX.1- Comparaisons des courbes des croissances usuelles	21
IX.2- Comparaison de qq. ordres de de complexité	22
X- Éléments d'analyse de la complexité	23
XI- Sensibilité à la puissance des machines	25
XII- Déf. de big-oh (upper bound, pessimiste)	29
XII.1- Exemples de big-oh	31
XIII- La fonction Ω (lower bound : optimiste)	35
XIV- La fonction Θ (égalité entre O et Ω)	37
XV- Illustration des 3 fonctions O , Ω , Θ	41
XVI- Calculs : le Modèle (de la machine)	42
XVII- Règles basiques et empiriques de calcul	43
XVIII- Quelques exemples de calculs simples	46

XVIII.1- Exemple : calcul de la médiane d'une suite	47
XVIII.1.1- Solution : un stratégie Diviser pour régner	48
XVIII.1.2- Détails du déroulement pour cet exemple	50
XVIII.2- Exemple : séquence de somme maximale	51
XIX- A propos de la complexité moyenne $A(n)$	60
XX- Propriétés des limites de fonctions	62
XX.1- Exemple trivial d'utilisation des limites	64
XX.2- Application : approximation de la complexité par programme	65
XX.3- Exemple 1	67
XX.3.1- Estimation empirique de la complexité de l'exemple	68
XX.4- Exemple 2	70
XXI- Éléments du calculs de la complexité	72
XXI.1- Équation de récurrence	72
XXI.2- Résolution de l'équation récurrente	74
XXII- Cas simples : proposer et vérifier	77
XXIII- Cas général : Preuve	81
XXIII.1- Exemple de preuve : Hanoi	82
XXIII.2- Exemple de preuve : recherche Dichotomique	83
XXIII.3- Preuve par Induction	84
XXIII.3.1- Exemple 1 : Factorielle	85
XXIII.3.2- Exemple 2	87
XXIII.3.3- Exemple 3 (à démontrer)	88
XXIII.3.4- Exemple 4 (cas défavorable)	89
XXIII.3.5- Un autre cas défavorable : Fib	90
XXIV- Résolution de l'équation caractéristique	91
XXIV.1- Théorème 1	95
XXIV.1.1- Exemple	96
XXIV.1.2- Une autre définition de Fib	97
XXV- Cas de racines multiples : théorème-2	99
XXV.1- Théorème 2	100
XXV.1.1- Exemple 1	101
XXV.1.2- Exemple 2	103
XXVI- Récurrence linéaire non homogène	104
XXVI.1- Théorème 3	105
XXVI.1.1- Exemple 1	106
XXVI.1.2- Exemple 2	108

XXVI.1.3- Exercice 1	110
XXVI.1.4- Exercice 2	111
XXVII- Boîte à outils (Cook Book)	112
XXVII.1- Méthode principale (MM) et la famille Thêta	113
XXVII.1.1- Exemples	115
XXVII.2- Méthode principale généralisée (MMG)	116
XXVII.2.1- Exemples	118
XXVIII- Complément technique	120
XXVIII.1- Changement de variable (cas homogène)	120
XXVIII.1.1- Exemple : Recherche dichotomique	121
XXVIII.1.2- Exemple : Tri Fusion	122
XXVIII.2- Changement de variable dans une cas non homogène	124
XXVIII.2.1- Exemple	125
XXIX- Cas particulier : Racines imaginaires !	128
XXIX.1- Un exemple	128
XXIX.2- Un autre exemple	129
XXX- Table des matières	130