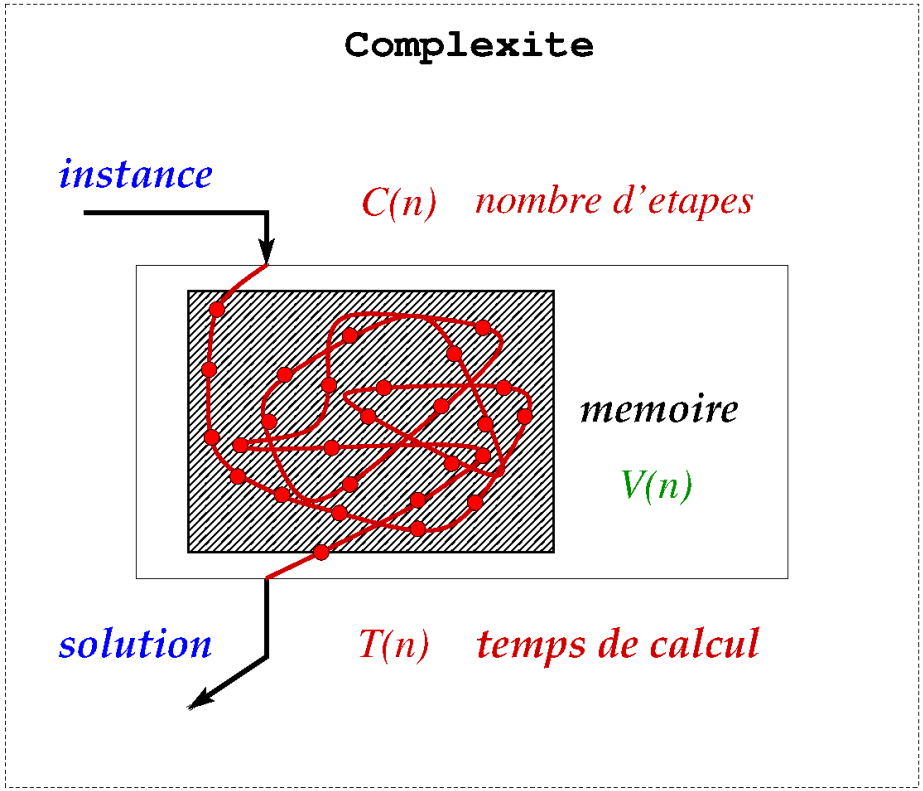
Temps de calcul, complexité des algorithmes

Philippe Langevin

IMATH, université de Toulon

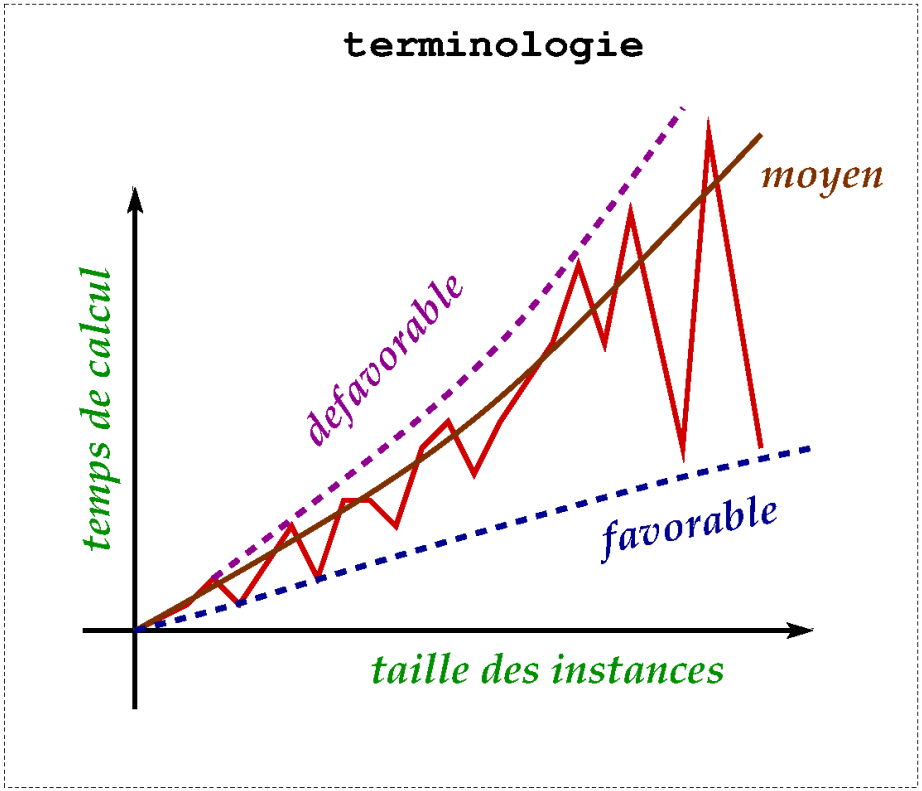
Février 2021

Complexité



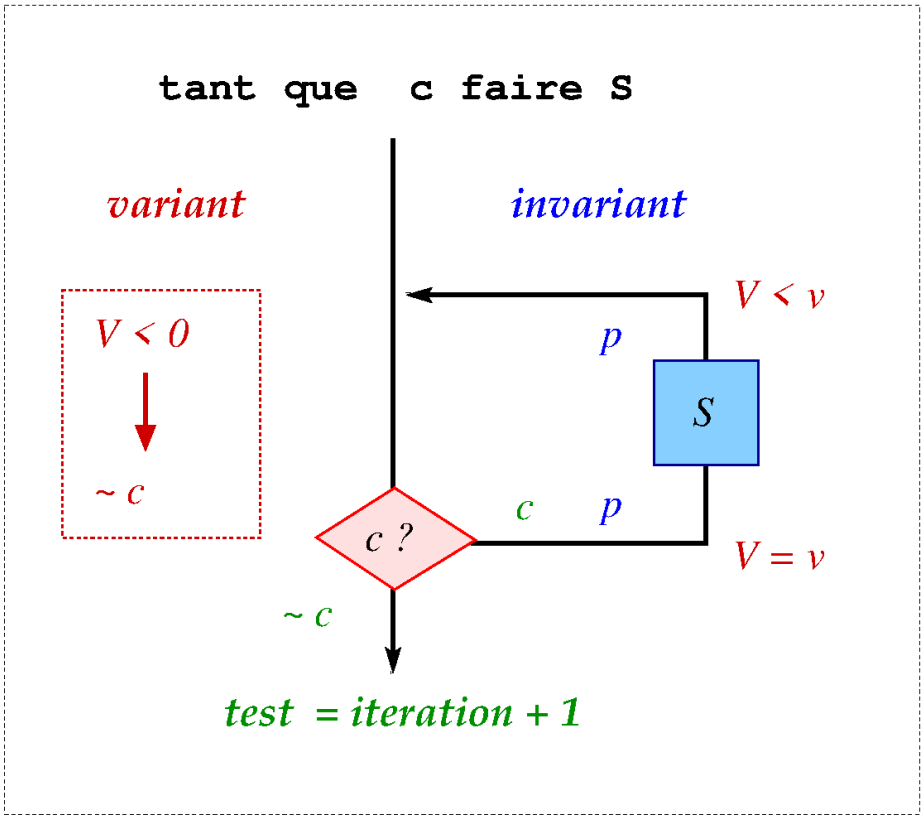
$$AC(n) \leq T(n) \leq BC(n), \quad V(n) \leq C(n).$$

Estimer les temps de calcul

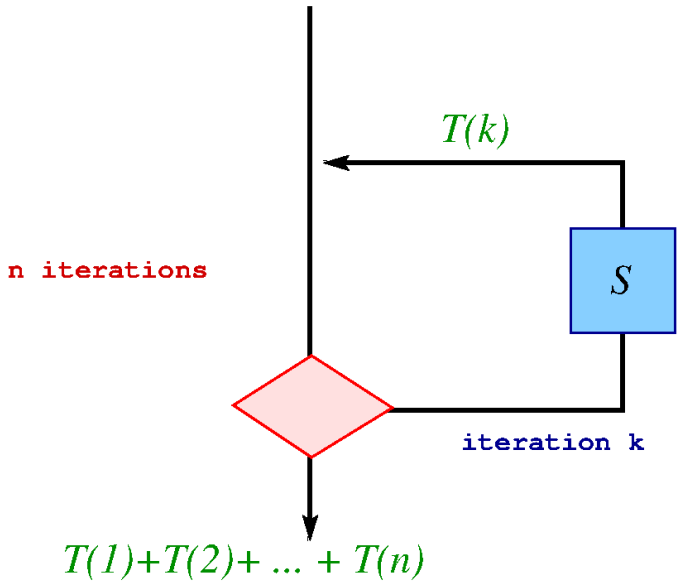


- $\check{T}(n)$, instance favorable ;
- $\tilde{T}(n)$, temps de calcul moyen ;
- $\hat{T}(n)$, instance défavorable.

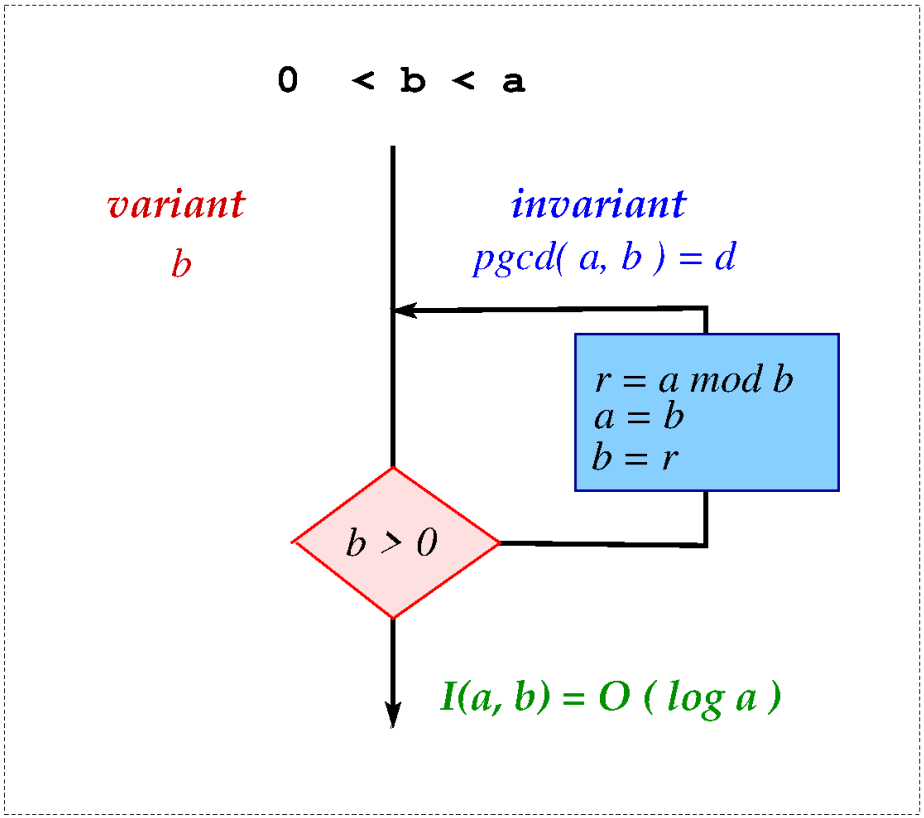
Analyse des boucles



Analyse des boucles



Algorithme d'Euclide (-300)



$I(\alpha, \beta)$: nombre d'itérations de l'algorithme pour l'instance $a = \alpha$, $b = \beta$.

Algorithme d'Euclide (-300)

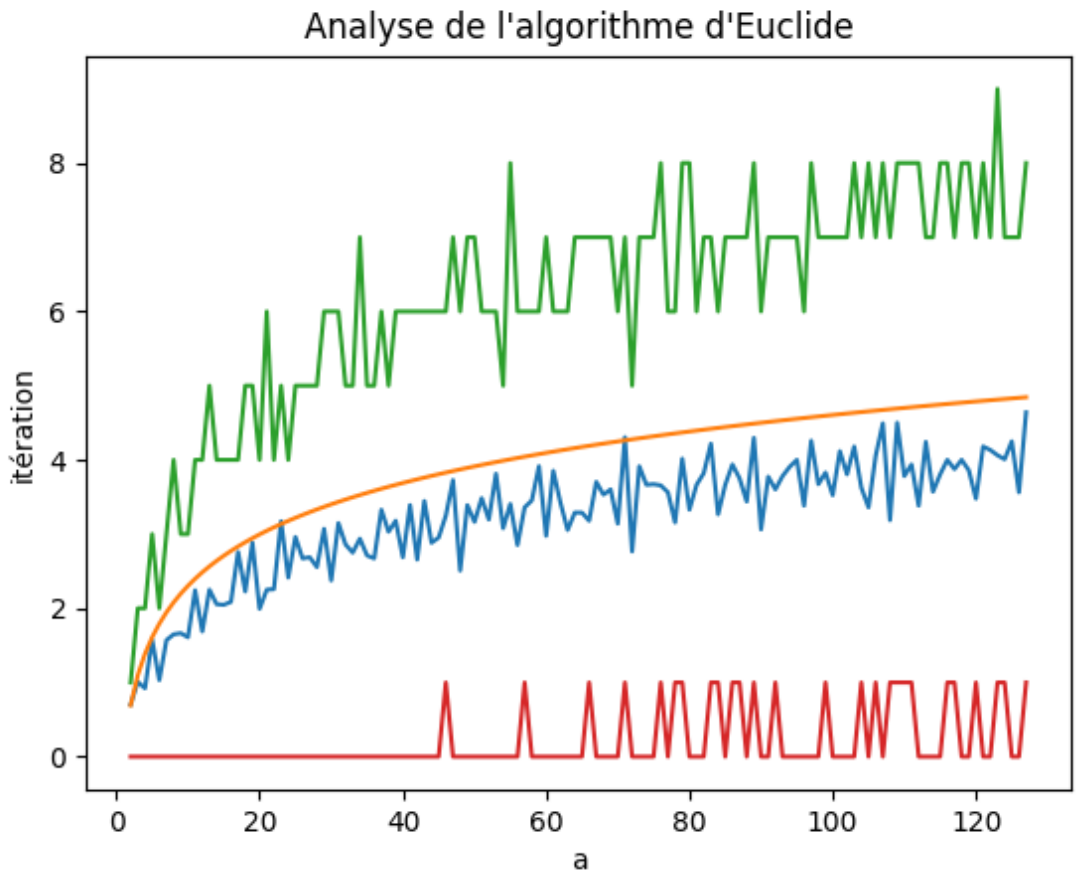
```
EUCLIDE( a, b : entier naturel )
variable     r : entier
debut
 tantque b > 0 faire
  r = a mod b
  a = b
  b = r
 ftq
 retourner a
fin
```

On suppose que $\beta < \alpha$.
Que peut-on dire du comportement des fonctions :

$$\check{I}(\alpha) = \min_{\beta<\alpha} I(\alpha,\beta) \quad \tilde{I}(\alpha) = \frac{1}{a}\sum_{\beta<\alpha} I(\alpha,\beta) \quad \hat{I}(\alpha) = \max_{\beta<\alpha} I(\alpha,\beta)$$

correspondant respectivement au temps de calcul du cas favorable, moyen et défavorable ?
Quand l'analyse mathématiques n'est pas évidente, il est possible de réaliser des mesures pour une interprétation des représentations graphiques qui peut nous aider à deviner un comportement asymptotique.

Mesure du nombre d'itération



On devine sur un comportement logarithmique du cas moyen et du pire des cas.

Un résultat de G. Lamé (1847)

Théorème (G. Lamé, 1847)

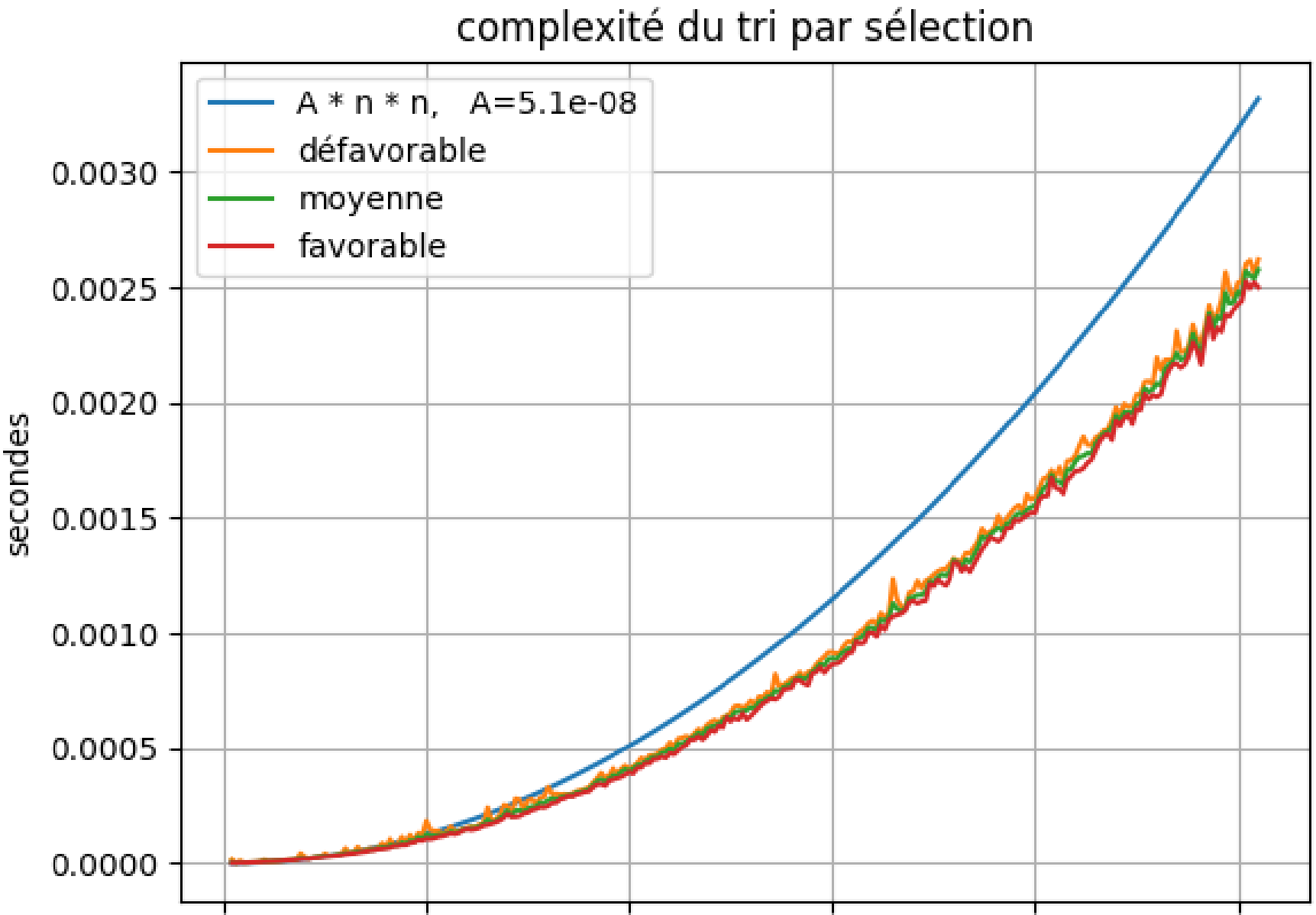
$$\hat{I}(a) = \mathrm{O}(\log a).$$

Démonstration.

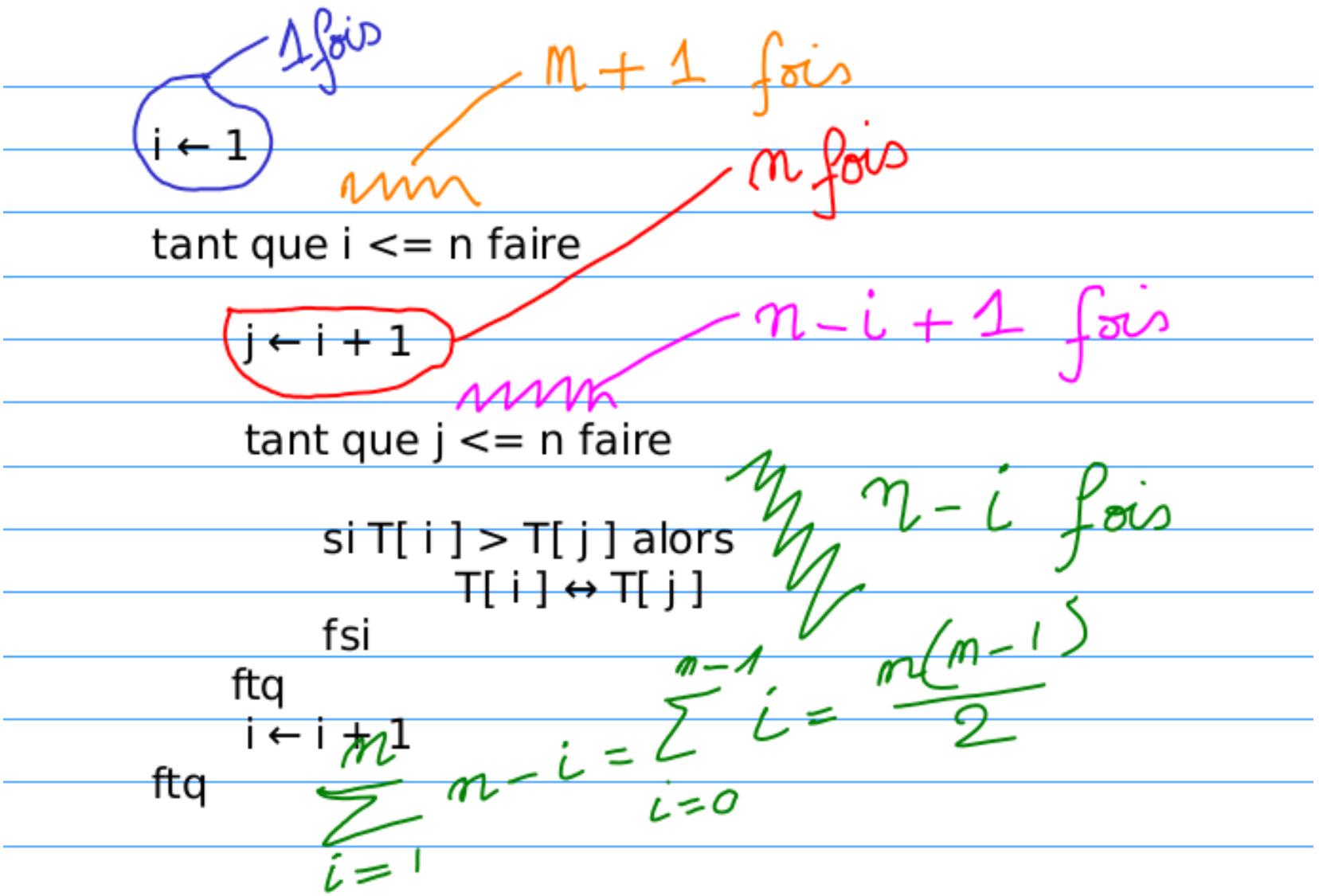
La preuve du mécanicien s'appuie sur le comportement exponentielle des suites de Fibonacci, à la portée d'un étudiant de première année, mais n'est pas au programme du I21. Le résultat analogue pour le cas moyen est beaucoup plus délicat. □

Tri par sélection

```
1  TRISELECTION( T : tableau de n nombres )
2  variable i, j : indice
3  debut
4   i ← 1
5   tantque i ≤ n faire
6    j ← i + 1
7    tantque j ≤ n faire
8     si T[i] > T[j] alors
9      T[i]⟷T[j]
10    fsi
11   ftq
12  ftq
13 fin
```


complexité du tri par sélection
A * n * n, A=5.1e-08
défavorable
moyenne
favorable
secondes
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000

Dénombrement des étapes



dénombrement des étapes

On introduit une fonction de comptage. Notons $C(n)$ le nombre d'étapes du tri par sélection pour traiter une instance de taille n sans compter les affectations de la ligne 9.

lignes		étape i	bilan
3, 4	1		
5	$n+1$		
6	n		
7		$n-i+1$	
8		$n-i$	$(n-1)+(n-2)+\cdots+0=\frac{1}{2}(n-1)n$
9		entre 0 et $n-i$	W(t)

$$C(n) = 1 + 1 + 2n + n + (n-1)n \qquad (1)$$

Estimation du temps de calcul

De cette formule, on déduit que le temps de calcul correspondant à ces lignes dans une implantation de l'algorithme sera de la forme

$$T(n) = \alpha + \beta n + \gamma n^2$$

Les trois constantes α, β et γ dépendront de la vertuosité du programmeur, de la qualité du compilateur et des performances du matériel. Le temps consommé par la ligne 9 dépend du tableau, il introduit la composante cahotique visible sur le graphique !

$$0 \leq W(t) \leq \delta n^2$$

Théorème (analyse tri par sélection)

Le temps de calcul du tri par sélection est quadratique.

Démonstration.

Le temps de calcul d'une instance t de taille n s'écrit :

$$T(n) + W(t)$$

D'une part,

$$T(n) = \Theta(n^2),$$

et d'autre part

$$0 \leq W(t) = \mathrm{O}(n^2).$$

□

Les notations sont expliquées plus loin.

Implantation en langage C

```
1  void triselect  ( int *t, int n )
2  {
3      int i, j, tmp;
4      for( i=0; i < n; i++ )
5          for( j = i+1; j < n; j++ )
6              if ( t[i] > t[j] ) {
7                  coeur:
8                  tmp = t[i];
9                  t[i] = t[j];
10                 t[j] = tmp;
11     }
12 }
```

Vérification expérimentale

```
1 pl@812/notes> /usr/bin/time --format "cpu=%U" ./a.out 10000
2 cpu=0.24
3 pl@812/notes> /usr/bin/time --format "cpu=%U" ./a.out 20000
4 cpu=0.96
```

Remarque

Dans le cas d'un algorithme de temps de calcul quadratique, doubler la taille d'une instance multiplie le temps de calcul par 4.

$$T(n) = An^2, \quad T(2n) = A(2n)^2 = 4An^2 = 4T(n)$$

Opération élémentaire du coeur de boucle

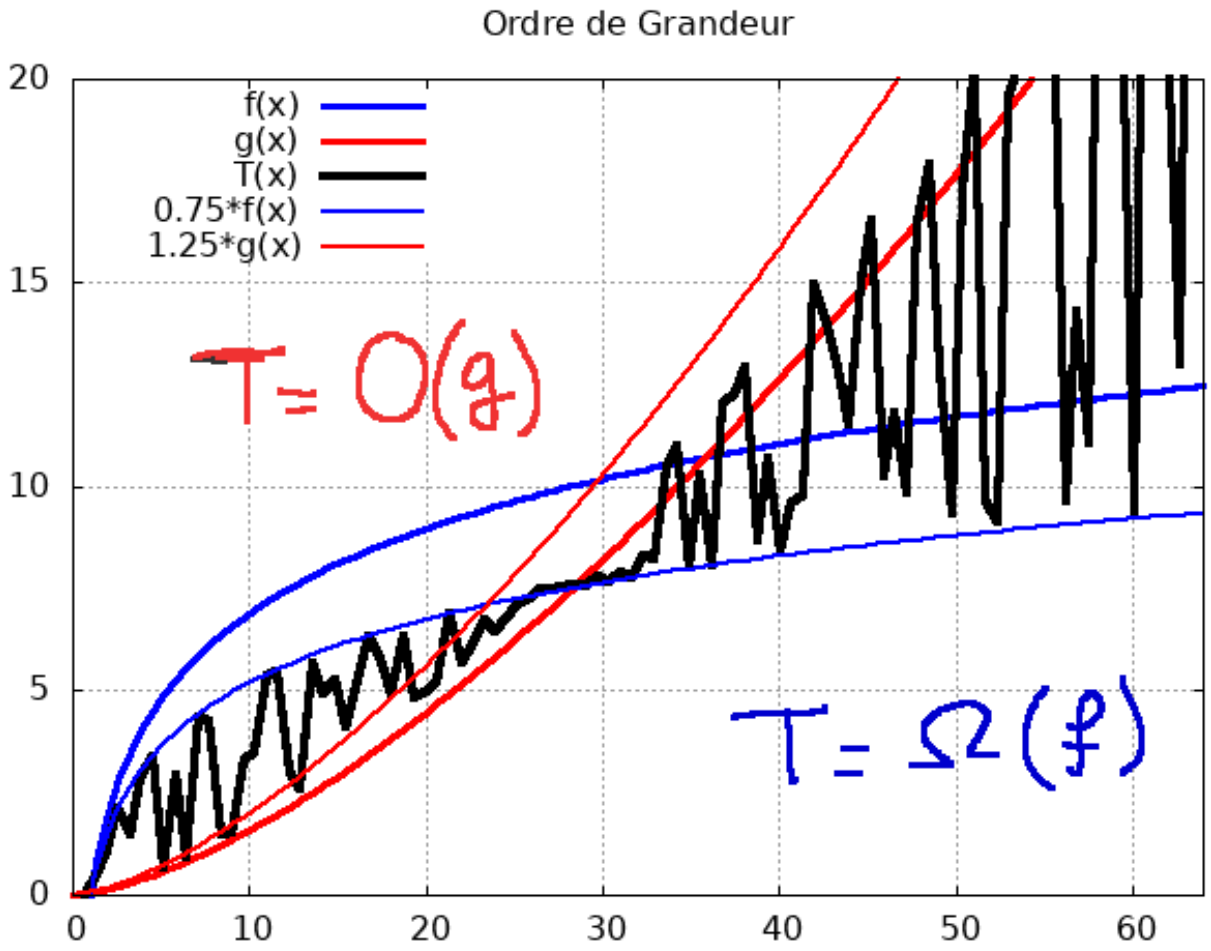
```
1 pl@812/notes> lscpu | grep -i hz
2 Nom de modele :  Intel(R) Core(TM) i5-4690S CPU @ 3.20GHz
3 Vitesse  du processeur  en MHz :          3192.556
4
5 pl@812/notes> bc -l
6 iter =20000*20000
7 3.2*0.96*10^9/iter
8 7.680000000000000000
```

Remarque

Ici comme dans la grande majorité des cas, le temps de calcul d'une structure itérative imbriquée est dominé par le temps d'exécution des instructions qui sont au coeur de la boucle interne.

Échelle de comparaison

forme	terminologie	propriété
1	constante	
$\log(x)$	logarithme	
$\log(x)^r$	polylogarithme	
x	linéaire	
$x * \log(x)^r$	quasi-linéaire	
x^2	quadratique	
x^3	cubique	
2^x	géométrique	
$\exp(x)$	exponentielle	
$x!$	factorielle	



Comparaison d'une fonction T avec des fonctions de références.

Domination

Definition

Soient T et g deux fonctions. On dit que T est est dominée par g s'il existe un constante $B \neq 0$ et un seuil x_0 à partir duquel, pour $x > x_0$:

$$|T(x)| \leq Bg(x).$$

Remarque

On note :

$$T = \mathrm{O}(g),$$

et on dit que T est un grand O de g.

$$\exists B > 0, \quad \exists x_0 \in R \quad \forall x, \quad x \geq x_0 \Longrightarrow |T(x)| \leq Bg(x)$$

Exemples

$$x^2 + 2^{1024}x = \mathrm{O}(x^2)$$

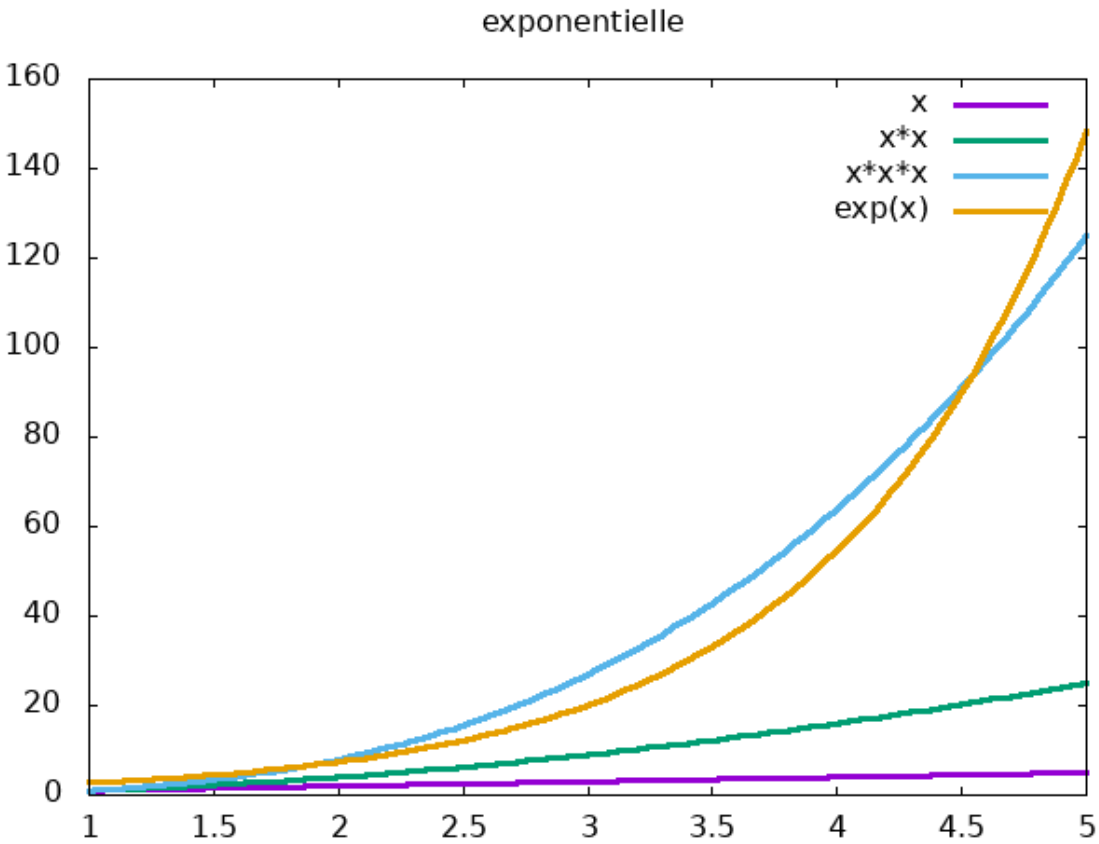
$$\sin(x) + \cos(x) = \mathrm{O}(1)$$

$$n! = \mathrm{O}(n^n)$$

$\mathfrak{s}(x)$ la somme des chiffres de l'entier x écrit en base 10, par exemple $\mathfrak{s}(2021) = 5$:

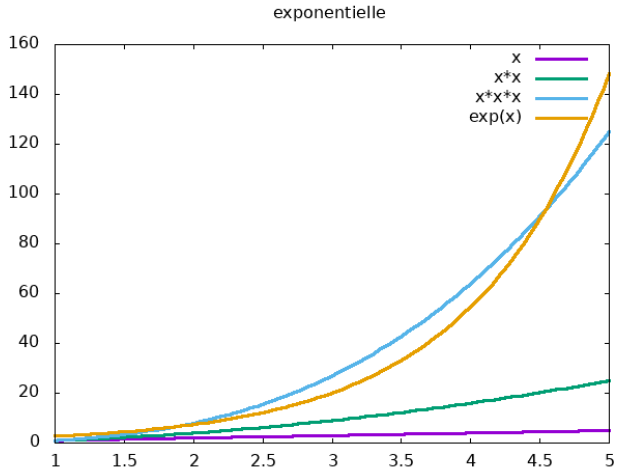
$$\mathfrak{s}(n) = \mathrm{O}(\log n)$$

Exponentielle vs linéaire, quadratique et cubique.



$$\forall s \in \mathbb{N}, \quad x^s = \mathrm{O}(\exp(x))$$

Exponentielle vs linéaire, quadratique et cubique.

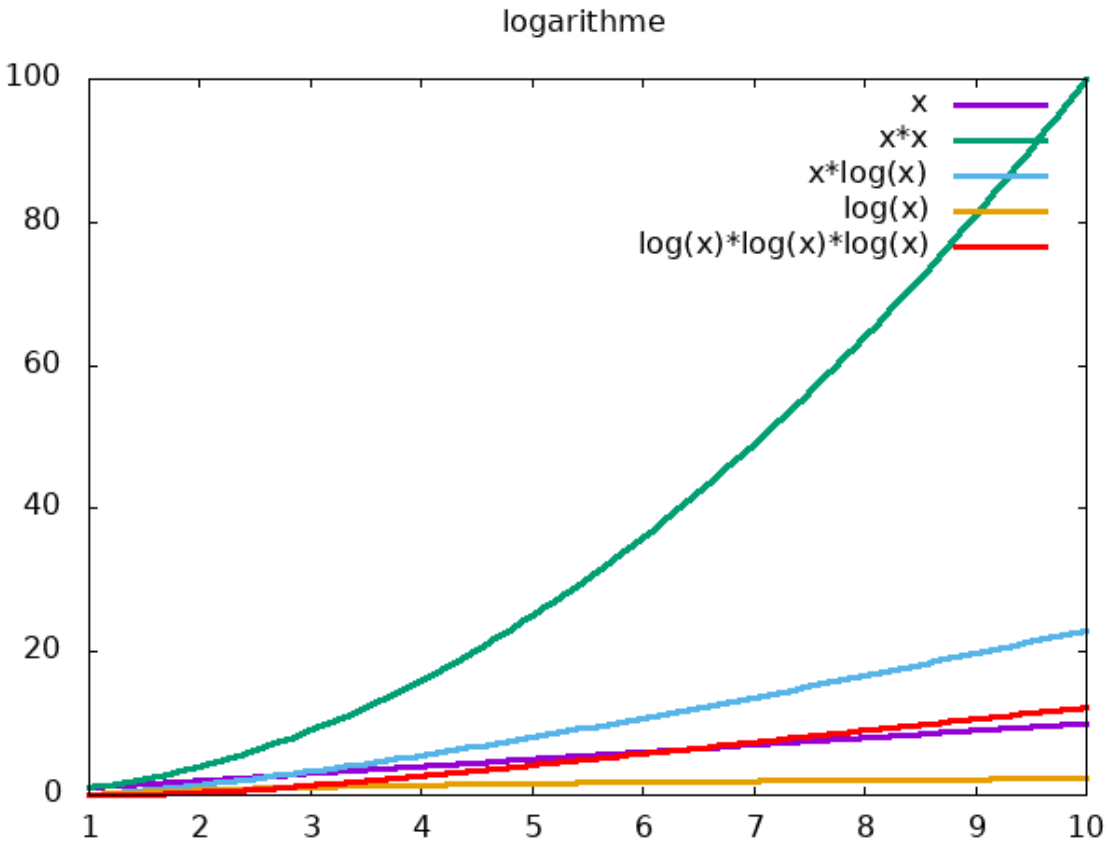


Théorème (polynôme vs exponentielle)

Les fonctions polynômes sont négligeables devant l'exponentielle. Pour tout entier $r \in \mathbb{N}$,

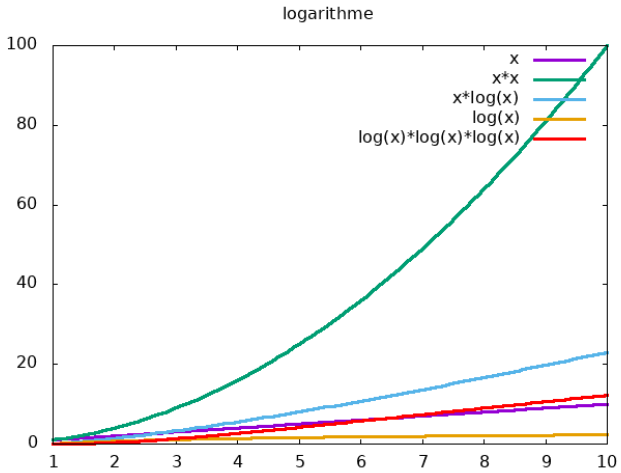
$$\lim_{x \to +\infty} \frac{\exp(x)}{x^r} = +\infty.$$

Polylogarithme, quasilinéaire, linéaire et quadratique.



$$\forall s \in \mathbb{N}, \quad \log(x)^s = \mathrm{O}(x)$$

Polylogarithme, quasilinéaire, linéaire et quadratique.



Théorème (polylogarithme vs linéaire)

Les fonctions polylogarithmes sont négligeables devant les fonctions polynômes. Pour tout entier $r \in \mathbb{N}$,

$$\lim_{x \to +\infty} \frac{\log(x)^r}{x} = 0 + .$$

Minoration

Definition

Soient T et f deux fonctions. On dit que T est (asymptotiquement) minorée par f (positive) s'il existe un constante positive $A > 0$ et un seuil x_0 à partir duquel, pour $x > x_0$:

$$Af(x) \leq |T(x)|\,.$$

Remarque

On note :

$$T = \Omega(f),$$

et on dit que T est un grand omega de f.

$$\exists A > 0, \quad \exists x_0 \in R \quad \forall x, \quad x \geq x_0 \Longrightarrow Af(x) \leq |T(x)|$$

Exemples et contre-exemples

$$x^2 - 2^{1024}x = \Omega(x^2)$$

$$\sin(x) + \cos(x) \neq \Omega(1)$$

$$|\sin(x)| + |\cos(x)| = \Omega(1)$$

$$n! \neq \Omega(n^n), \quad n^n = \Omega(n!)$$

$$\mathfrak{s}(n) \neq \Omega(\log n) \qquad \mathfrak{s}(n) = \Omega(1)$$

Équivalence asymptotique

Definition

Soient T et h deux fonctions. On dit que T est asymptotiquement du même ordre de grandeur que h quand :

$$T = \Omega(h), \quad \text{et} \quad T = \mathrm{O}(h).$$

Remarque

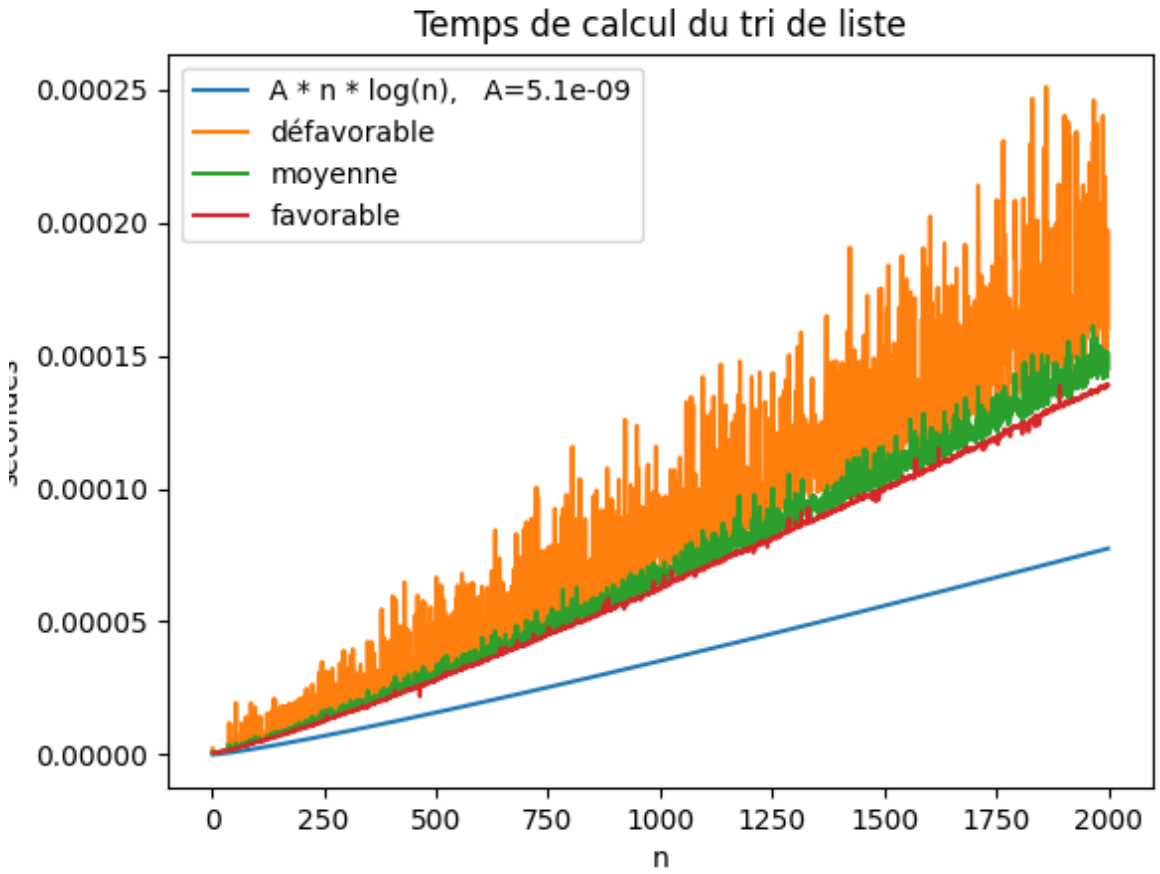
On note :

$$T = \Theta(h),$$

et on dit que T est un grand thêta de h.

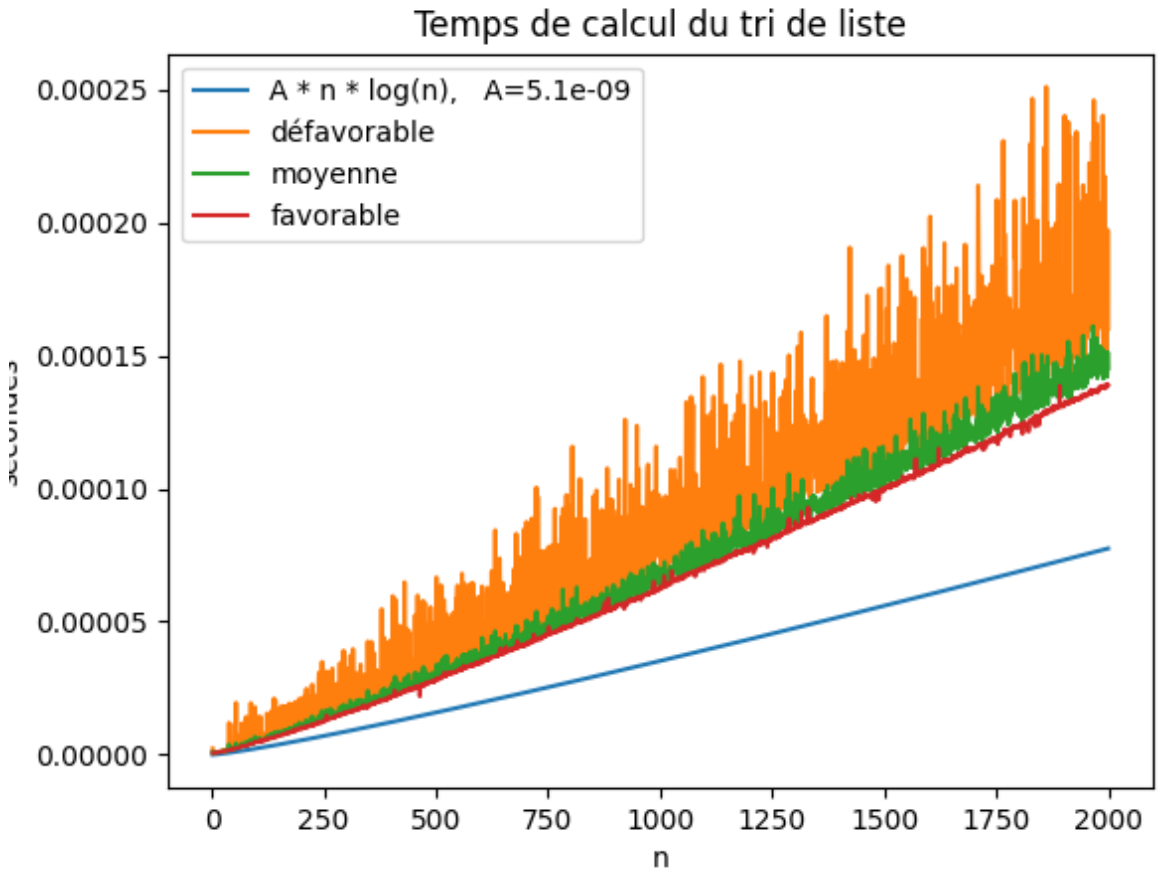
$$\exists A, B > 0, \quad \exists x_0 \in R \quad \forall x, \quad x \geq x_0 \Longrightarrow Ah(x) \leq |T(x)| \leq Bh(x)$$

complexité du tri rapide.



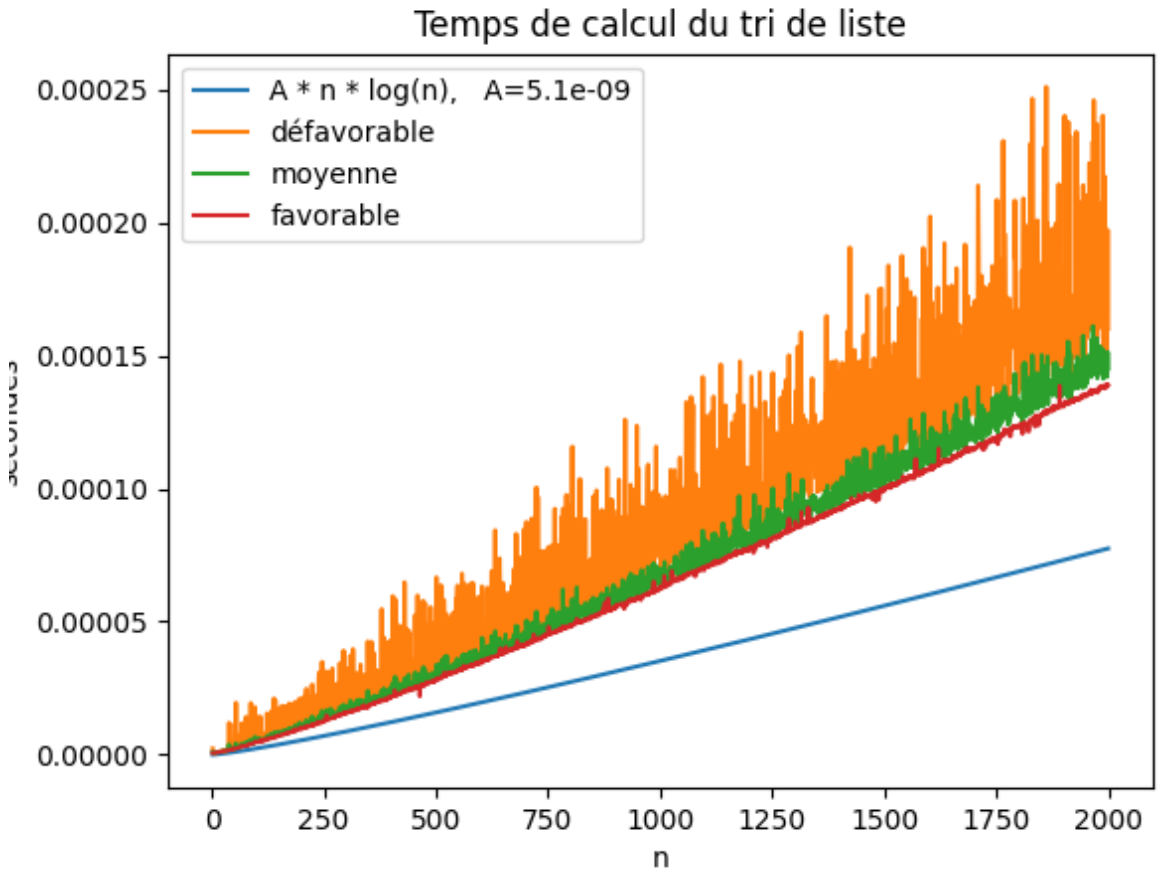
$$\check{T}(n) =$$

complexité du tri rapide.



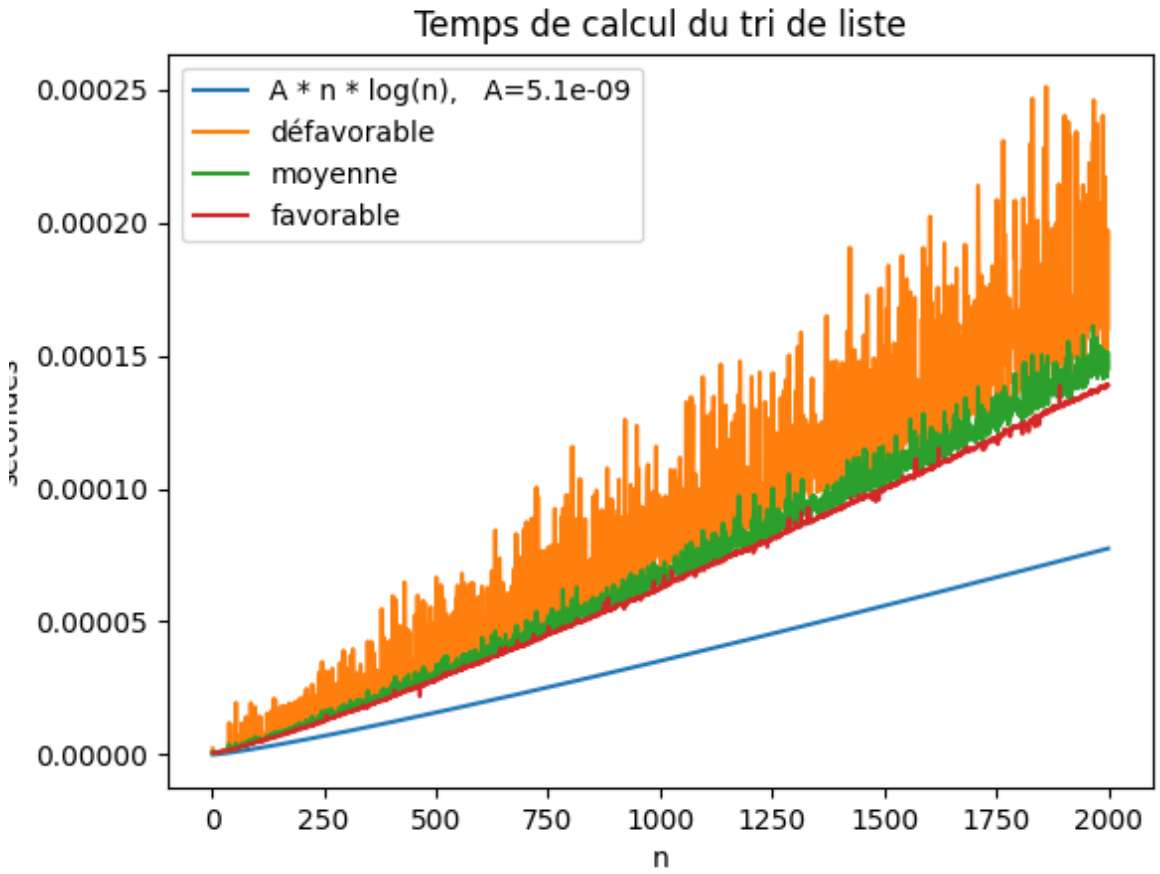
$\check{T}(n) = \Theta(n \log n) \quad \tilde{T}(n) =$

complexité du tri rapide.



$$\check{T}(n) = \Theta(n \log n) \quad \tilde{T}(n) = \Theta(n \log n) \quad \hat{T}(n) =$$

complexité du tri rapide.



$$\check{T}(n) = \Theta(n \log n) \quad \tilde{T}(n) = \Theta(n \log n) \quad \hat{T}(n) = \Omega(n \log n)$$

Tri par comparaison.

Théorème

Le nombre d'étapes d'un algorithme de tri fondé sur la comparaison est au moins quasilinéaire.

Démonstration.

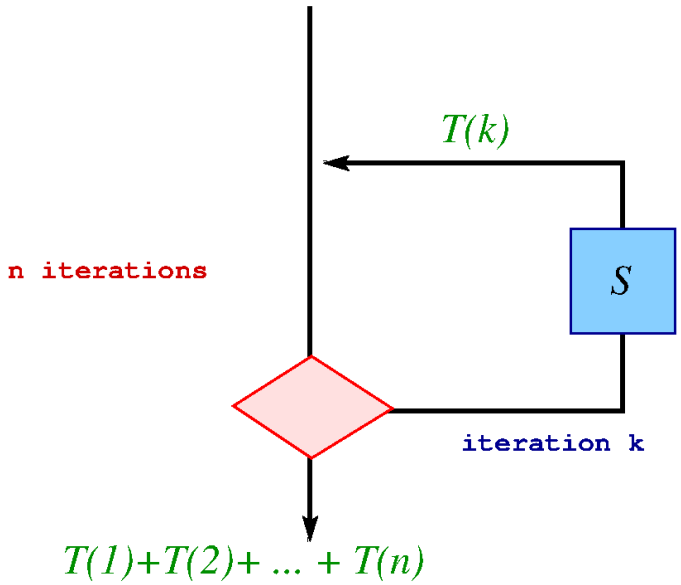
La preuve est au programme du cours de deuxième année. □

$$C(n) = \Omega(n \log n).$$

Conséquence, pour toute implantation d'un algorithme de tri, il existe une constante positive $A > 0$ tel que le temps de calcul vérifie :

$$An \log n \leq T(n).$$

Temps de calcul d'une boucle



Théorème (ordre de grandeur et somme)

La fonction T mesure un temps, elle est positive. Si $T(k) = \Theta\big(f(k)\big)$ alors le temps de la boucle est $T(1) + T(2) + \cdots + T(k) = \Theta(\sum_{k=1}^{n} f(k))$.

série arithmétique

Une suite de nombres complexes dont la différence entre deux termes consécutifs est constante est dite arithmétique.

Théorème

La somme de n termes consécutifs d'une suite arithmétique

$$\frac{u_{s+1} + u_{s+2} + \ldots + u_{s+n}}{n} =$$

série arithmétique

Une suite de nombres complexes dont la différence entre deux termes consécutifs est constante est dite arithmétique.

Théorème

La somme de n termes consécutifs d'une suite arithmétique

$$\frac{u_{s+1} + u_{s+2} + \ldots + u_{s+n}}{n} = \frac{u_{s+1} + u_{s+n}}{2}$$

Démonstration.

parcourir dans les deux sens. □

série géométrique

Une suite de nombres complexes dont le rapport entre deux termes consécutifs est constant est dite géométrique.

Théorème

Pour tout nombre complexe $q \neq 1$,

$$\sum_{k=0}^{n-1} q^k =$$

série géométrique

Une suite de nombres complexes dont le rapport entre deux termes consécutifs est constant est dite géométrique.

Théorème

Pour tout nombre complexe $q \neq 1$,

$$\sum_{k=0}^{n-1} q^k = \frac{q^n - 1}{q - 1}$$

Démonstration.

décalage !

somme de puissance

$$\sum_{k=0}^{n-1} k = 0 + 1 + 2 + \ldots + (n-1) = \frac{1}{2} n(n-1) = \Theta(n^2)$$

Théorème (série de puissances)

Soit s, un entier strictement positif.

$$\sum_{k=1}^{n} k^s = 1^s + 2^s + \ldots + n^s = \Theta(n^{s+1})$$

Il est clair que la somme est dominée par $\mathrm{O}(n^{s+1})$, la difficulté est plutôt de prouver la minoration en $\Omega(n^{s+1})$.

somme de logarithme

Théorème (série de logarithmes)

$$\sum_{k=1}^{n} \log k = \Theta(n \log n)$$

Il est clair que la somme est majorée par $\mathrm{O}(n \log n)$, la difficulté est encore de prouver la domination $\Omega(n \log n)$.

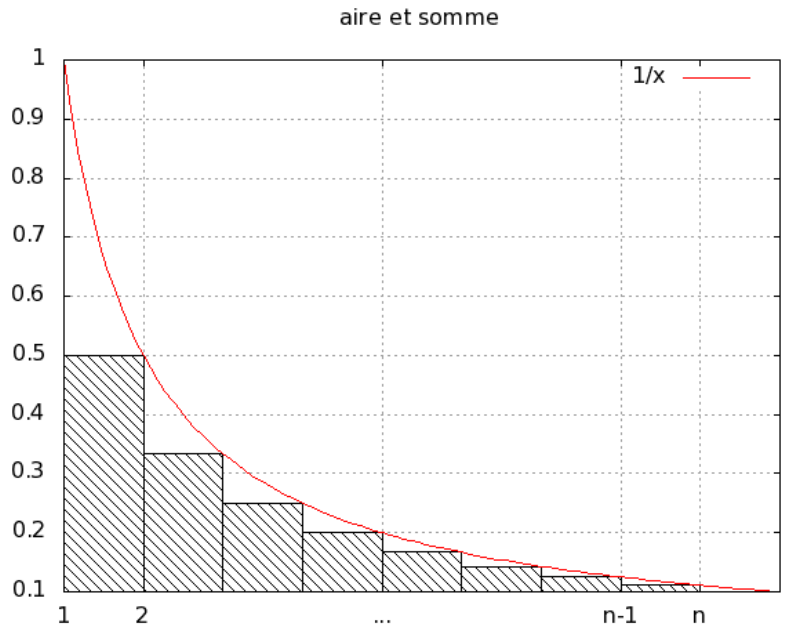
somme des inverses

Théorème (série Harmonique)

$$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$$

Pas d'évidence mais une stratégie bien rodée !

somme et intégrale



L'estimation de la série Harmonique se déduit de l'inégalité :

$$\int_1^n \frac{\mathrm{d}t}{t} + 1 \geq \sum_{k=1}^{n} \frac{1}{k} \geq \int_1^{n+1} \frac{\mathrm{d}t}{t}$$

Un principe analogue s'applique aux sommes des fonctions usuelles.

Principe général

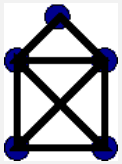
En notant, F une primitive d'une fonction croissante f ,

f	F
x^{s-1}	$\frac{1}{s}x^s$
$\log(x)$	$x\log(x)-x$
2^x	$\frac{1}{\log 2}2^x$

$$F(n)-F(1)=\int_0^n \frac{\mathrm{d}t}{t} \leq \sum_{k=1}^{n} f(k) \leq \int_1^{n+1} \frac{\mathrm{d}t}{t} = F(n+1)-F(1)$$

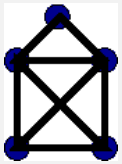
En particulier, si f n'est pas "trop" croissante :

$$F(n+1)=\Theta\big(F(n)\big) \quad \Longrightarrow \quad \sum_{k=1}^{n} f(k)=\Theta\big(F(n)\big).$$



Poids Binaire

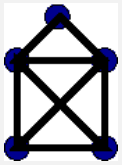
```
POIDSBINAIRE( x : entier de n bits )
variable  w : entier
debut
 w ← 0
 tantque x > 0 faire
  w ← w +  x mod 2
  x ← x div 2
 ftq
 retourner w
fin
```



Poids Binaire

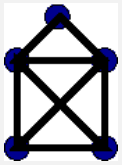
```
POIDSBINAIRE( x : entier de n bits )
variable  w : entier
debut
 w ← 0
 tantque x > 0 faire
  w ← w +  x mod 2
  x ← x div 2
 ftq
 retourner w
fin
```

$C(n) = \Theta(n)$, linéaire en la taille...  logarithmique (en x) !



Test de primalité

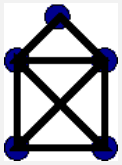
```
PREMIER( x : entier de n bits )
variable  t : entier
debut
 t ← 2
 tantque ( x mod t > 0 ) et ( t * t  <  x ) faire
  t ← t + 1
 ftq
 retourner x mod t  > 0
fin
```



Test de primalité

```
PREMIER( x : entier de n bits )
variable  t : entier
debut
 t ← 2
 tantque ( x mod t > 0 ) et ( t * t  <  x ) faire
  t ← t + 1
 ftq
 retourner x mod t  > 0
fin
```

$C(n) = \mathrm{O}(\sqrt{2}^n)$, exponentielle en la taille...  linéaire (en x) !



Crible d'Ératostène : quasilinéaire en n

```
CRIBLE(  n : entier )
variable  t : tableau de n booleens
   cpt = 0 : entier;        i = 2 , j : indice
debut
 t ← [VRAI, VRAI, ..., VRAI  ]
 tantque ( i < n )  faire
  si ( t[ i ] ) alors
     cpt++;   j ← 2 * i
     tant que (j < n) faire
                          t[ j ] = FAUX; j = j + i
     ftq
  fsi
  i = i + 1
 ftq
 retourner cpt
fin
```

Exercice 6 de la série 3.1

```
1  DEBUT
2  i ← 1
3  TQ  i ≤ n  FAIRE
4      j ← 1
5      TQ  j ≤ i  FAIRE
6          W( j )
7          j ← j + 1
8      FTQ
9  i ←   i + 1
10 FTQ
11 FIN
```

Listing 1 – $W(n)$ // job of complexity $\Theta(2^n)$

Le temps de cacul de la boucle interne (lignes 5-6-7-8) à l'étape i est encore $\Theta(2^i)$, c'est une propriété des sommes géométriques...

Exercice 2 de la série 3.1

```
DEBUT
i , j ← 1 , n
TANT QUE i ≤ j FAIRE
        W( i ) ; W' ( j )
        i ← i + 1
        j ← j − 1
FTQ
FIN
```

Listing 2 – $W(n)$ linéaire, $W'(n)$ logarithmique