Cours “Algorithmique et Structure des Données”

Licence L2 Informatique, semestre S3

Année 2019-2022

1. Analyse des algorithmes
   - Complexité
2. Algorithmes de tri
   - Principes
   - Tri insertion
   - Tri fusion
   - Tri rapide
   - Tri en tas
   - Tri par comptage
   - Comparaison des algorithmes de tri

Définitions

La complexité d'un algorithme est la mesure de son efficacité :

- Estimer les ressources (temps, mémoire, ...) nécessaires pour l'exécuter
- Comparer des algorithmes entre eux

Il s'agit d'une mesure théorique, indépendante :

- du langage de programmation et du compilateur
- du processeur de l'ordinateur

On distingue principalement deux types de complexité :

- en temps (nombre d'opérations élémentaires)
- en espace (taille de la mémoire)

Définitions

La complexité s'exprime en fonction de la taille des entrées. Elle peut dépendre également de la façon dont sont réparties les valeurs. Cela conduit à distinguer trois formes de complexité :

- la complexité dans le meilleur des cas
- la complexité dans le pire des cas
- la complexité en moyenne

Calcul de complexité

Pour simplifier le calcul de la complexité d'un algorithme, on regarde comment elle varie quand la taille des entrées tend vers l'infini (on parle de comportement asymptotique), ce qui implique que l'on négligera :

- les éléments constants : n sera équivalent à $n + C$
- les facteurs multiplicatifs : n sera équivalent à $2n$ ou $2048n$

Pour noter les complexités, on utilisera la notation O : c'est d'une certaine manière un ordre de grandeur. (en réalité les notions mathématiques derrière cette notation sont plus ... *complexes*, et dépassent la portée de ce cours).
Par exemple, si l'ordre de grandeur du nombre d'opérations élémentaires nécessaires pour exécuter un algorithme varie avec le carré de la taille (le nombre) des entrées, on notera sa complexité en temps : $O(n^2)$.

Différentes complexités

Temps exprimés en μs. Pour $n = 1$, le temps est de $1\mu s$.

Complexité	Type	$n = 5$	$n = 10$	$n = 10^6$
$O(1)$	constante	10	10	10
$O(\log(n))$	logarithmique	10	10	60
$O(n)$	linéaire	50	100	10.10^6
$O(n \log(n))$	linéarithmique	50	100	60.10^6
$O(n^2)$	quadratique	250	10^3	$2,8$ heures
$O(2^n)$	exponentielle	320	10^4	*beaucoup trop*
$O(n!)$	factorielle	$1,2.10^3$	36.10^6	*encore plus*

Tulité de la notion de complexité

C'est utile parce que une complexité linéarithmique est très inférieure à une complexité quadratique, par exemple.
Mais :

- cela ne permet pas de distinguer les algorithmes ayant une complexité équivalente
- c'est une estimation asymptotique
- même si attendre un calcul un an est long, l'attendre deux ans est deux fois plus long ...

Il est nécessaire aussi de prendre en compte les temps d'exécution !

Exemple

Recherche dans un tableau

```
1: function RECHERCHE(A, x)
2:     i ← 1, f ← FALSE
3:     while !f and i ≤ longueur(A) do
4:         f ← x = A[i]
5:         i ← i + 1
6:     end while
7:     if f then
8:         return i − 1
9:     else
10:        return −1
11:    end if
12: end function
```

- Complexité pire cas ?
- Complexité meilleur cas ?
- Complexité moyenne ?

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

Principes et définitions

On s'intéresse ici au tri sur tableaux (sur les listes les algorithmes sont parfois un différents : cf cours suivant). On dit qu'un tableau A d'indice $[1..n]$ est trié en ordre croissant si :

- $n = 0$ (tableau vide)
- $n = 1$ (tableau à un élément)
- pour $n > 1$, $\forall i \in [2..n] A[i-1] \leq A[i]$

Un *tri* est un algorithme qui transforme un tableau en un tableau trié. On dit qu'un tri est stable si l'ordre des éléments équivalents est préservé (le premier k du tableau d'origine est toujours le premier k du tableau résultat).

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

Algorithme

```
1:  procedure TRIINSERTION(A)                                   ▷ A tableau [1..n]
2:      for i ← 2 to longueur(A) do
3:          key ← A[j]
4:          i ← j − 1
5:          while i > 0 and A[i] > key do      ▷ Insérer A[j] dans la séquence triée A[1..j − 1]
6:              A[i + 1] ← A[i]
7:              i ← i − 1
8:          end while
9:          A[i + 1] ← key
10:     end for
11: end procedure                                   ▷ Le tableau A est trié [1..n]
```

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

Algorithme

```
1: procedure triFusion(A, p, r)
2:     if p < r then
3:         q ← ⌊(p + q)/2⌋
4:         triFusion(A, p, q)
5:         triFusion(A, q + 1, r)
6:         fusion(A, p, q, r)
7:     end if
8: end procedure
```

Algorithme

```
1: procedure triFusion(A, p, r)
2:     if p < r then
3:         q ← ⌊(p + q)/2⌋
4:         triFusion(A, p, q)
5:         triFusion(A, q + 1, r)
6:         fusion(A, p, q, r)
7:     end if
8: end procedure
```

```
1: procedure fusion(A, p, q, r)
2:     for i ← 1 to q − p do
3:         L[i] ← A[p + i]
4:     end for
5:     for j ← 1 to r − q do
6:         M[j] ← A[q + j]
7:     end for
8:     i ← 1, j ← 1, k ← 1
9:     while i ≤ q − p and j ≤ r − q do
10:        if L[i] ≤ M[j] then
11:            A[k] ← L[i]
12:            i ← i + 1
13:        else
14:            A[k] ← M[j]
15:            j ← j + 1
16:        end if
17:        k ← k + 1
18:    end while
19:    while i ≤ q − p do
20:        A[k] ← L[i]
21:        i ← i + 1
22:        k ← k + 1
23:    end while
24:    while j ≤ r − q do
25:        A[k] ← M[j]
26:        j ← j + 1
27:        k ← k + 1
28:    end while
29: end procedure
```

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

Algorithme

```
1: procedure triRapide(A, p, r)
2:     if p < r then
3:         q ← partition(A, p, r)
4:         triRapide(A, p, q)
5:         triRapide(A, q + 1, r)
6:     end if
7: end procedure
```

```
1: function partition(A, p, r)
2:     x ← A[p], i ← p − 1, j ← r + 1
3:     while True do
4:         repeat
5:             j ← j − 1
6:         until A[j] ≤ x
7:         repeat
8:             i ← i + 1
9:         until A[j] ≥ x
10:        if i < j then
11:            exchange(A[i], A[j])
12:        else
13:            return j
14:        end if
15:    end while
16: end function
```

1. Analyse des algorithmes
   - Complexité
2. Algorithmes de tri
   - Principes
   - Tri insertion
   - Tri fusion
   - Tri rapide
   - Tri en tas
   - Tri par comptage
   - Comparaison des algorithmes de tri

Algorithme

```
1: procedure TRITAS(A, p, r)
2:     CONSTRUIRETAS(A)
3:     for i ← length(A) downto 2 do
4:         exchange(A[1], A[j])
5:         TASSE(A, 1, i)
6:     end for
7: end procedure
```

```
1: procedure CONSTRUIRETAS(A)
2:     for i ← ⌊length(A)/2⌋ downto 1 do
3:         TASSE(A, i, length(A))
4:     end for
5: end procedure
```

```
1: function GAUCHE(i)
2:     return 2i
3: end function
```

```
1: function DROITE(i)
2:     return 2i + 1
3: end function
```

```
1: procedure TASSE(A, i, taille)
2:     l ← GAUCHE(i)
3:     r ← DROITE(i)
4:     if l ≤ taille and A[l] > A[i] then
5:         plusgrand ← l
6:     else
7:         plusgrand ← i
8:     end if
9:     if r ≤ taille and A[r] > A[plusgrand]
   then
10:        plusgrand ← r
11:    end if
12:    if plusgrand ≠ i then
13:        exchange(A[i], A[plusgrand]
14:        TASSE(A, plusgrand, length(A))
15:    end if
16: end procedure
```

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

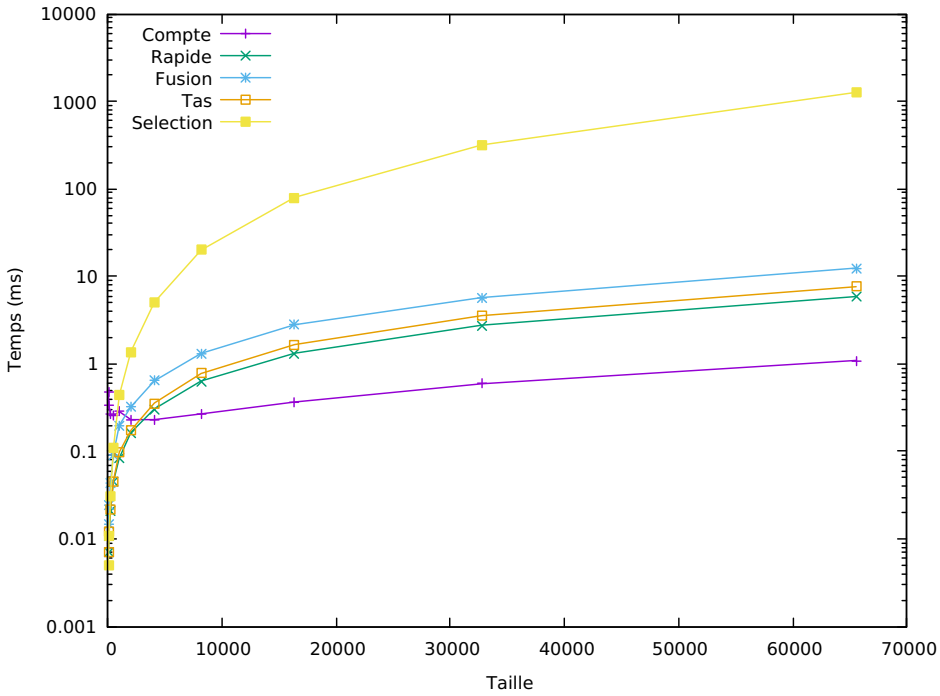
Algorithme

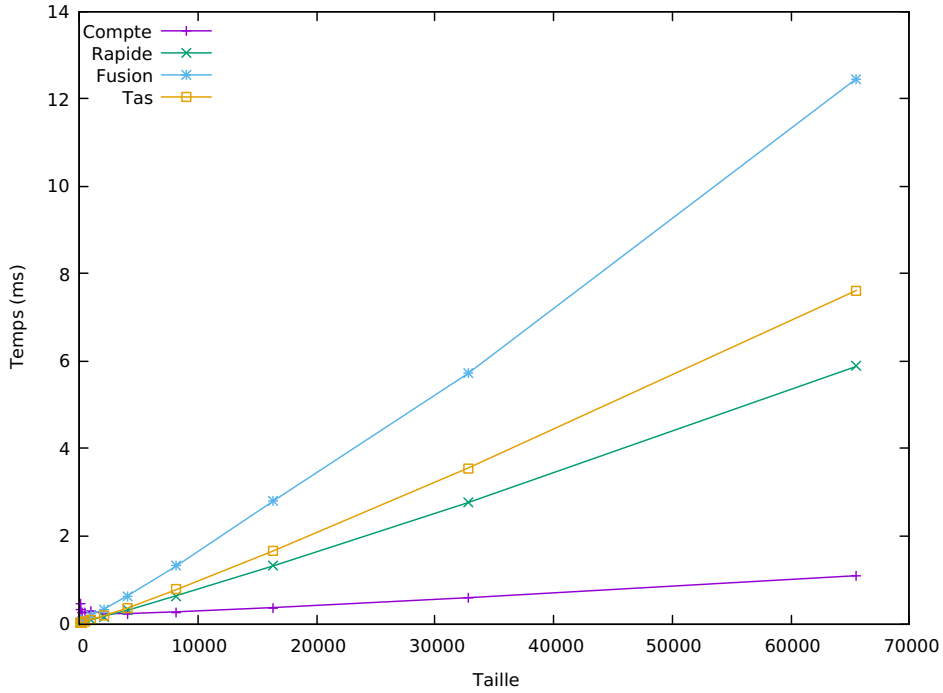
```
1:  procedure triCompte(A, B, k)
2:      for i ← 1 to k do
3:          C[i] ← 0
4:      end for
5:      for j ← 1 to length[A] do
6:          C[A[j]] ← C[A[j]] + 1
7:      end for
8:      for i ← 2 to k do
9:          C[i] ← C[i] + C[i − 1]
10:     end for
11:     for j ← length(A) downto 1 do
12:         B[C[A[j]]] ← A[j]
13:         C[A[j]]] ← C[A[j]]] − 1
14:     end for
15: end procedure
```

1 Analyse des algorithmes
- Complexité

2 Algorithmes de tri
- Principes
- Tri insertion
- Tri fusion
- Tri rapide
- Tri en tas
- Tri par comptage
- Comparaison des algorithmes de tri

	taille	Insertion	Fusion	Rapide	Tas	Compteurs
pire cas	n	$O(n^2)$	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(n+k)$
moyenne	n	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n+k)$
meilleur	n	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n+k)$
Aléatoire	2^6	0.033	0.065	0.037	0.050	4.169
Aléatoire	2^{11}	4.288	0.513	0.262	0.514	0.787
Aléatoire	2^{15}	1024.010	9.173	5.292	10.700	1.321
Aléatoire	2^{15}	316.394	5.986	2.818	3.601	0.585
Trié	2^6	0.006	0.050	0.042	0.045	4.195
Trié	2^{11}	0.010	0.334	5.419	0.457	0.857
Trié	2^{15}	0.156	5.988	1326.589	9.249	1.118
Inverse	2^6	0.067	0.067	0.054	0.046	4.559
Inverse	2^{11}	8.150	0.357	5.255	0.407	0.764
Inverse	2^{15}	2036.105	6.033	1317.498	8.615	1.101


Compte
Rapide
Fusion
Tas
Selection
Temps (ms)
Taille
10000
1000
100
10
1
0.1
0.01
0.001
0
10000
20000
30000
40000
50000
60000
70000


Compte
Rapide
Fusion
Tas
Temps (ms)
Taille
14
12
10
8
6
4
2
0
0
10000
20000
30000
40000
50000
60000
70000