INF1032

Algorithmique et programmation C 2

Enseignants

Arnaud Bannier, Nicolas Bodin

A

Complexité

Version 1.4

esiea

Algorithme – Ibijau (tab)
Entrée. Tab. de n entiers t
Sortie. Entier c

```
c ← 0
Pour i allant de 1 à n faire
    Pour j allant de 1 à n faire
        Si t[i]=t[j] alors
            Incrémenter(c)
Renvoyer c
```

Algorithme – Kakapo(tab)
Entrée. Tab. de n entiers t
Sortie. Entier c

```
c ← 0
Pour i allant de 1 à n faire
    Faire
        t[i] ← rand(0,9)
        Incrémenter(c)
    Tant que t[i]!=5
Renvoyer c
```

Algorithme – Ibijau (tab)
Entrée. Tab. de n entiers t
Sortie. Entier c

```
c ← 0
Pour i allant de 1 à n faire
    Pour j allant de 1 à n faire
        Si t[i]=t[j] alors
            Incrémenter(c)
Renvoyer c
```

Algorithme – Kakapo(tab)
Entrée. Tab. de n entiers t
Sortie. Entier c

```
c ← 0
Pour i allant de 1 à n faire
    Faire
        t[i] ← rand(0,9)
        Incrémenter(c)
    Tant que t[i]!=5
Renvoyer c
```

Quel algorithme se termine en premier ?
En combien de temps ?

```
Algorithme – Ibijau (tab)
Entrée.  Tab. de n entiers t
Sortie.  Entier c

c ← 0
Pour i allant de 1 à n faire
  Pour j allant de 1 à n faire
    Si t[i]=t[j] alors
      Incrémenter(c)
Renvoyer c
```

```
Algorithme – Kakapo(tab)
Entrée.  Tab. de n entiers t
Sortie.  Entier c

c ← 0
Pour i allant de 1 à n faire
  Faire
    t[i] ← rand(0,9)
    Incrémenter(c)
  Tant que t[i]!=5
Renvoyer c
```

Quel algorithme se termine en premier ? En combien de temps ?

Si la taille du tableau double, qu'en est-il du temps d'exécution ?

Premier exemple

Taille des entrées. Définie par la taille n du tableau.

Algorithme – `Ibijau (tab)`

Entrée. `Tab. de n entiers t`

Sortie. `Entier c`

```
c ← 0
Pour i allant de 1 à n faire
    Pour j allant de 1 à n faire
        Si t[i]=t[j] alors
            Incrémenter(c)
Renvoyer c
```

Nombre d'opérations. n^2 tours de boucle :

- Boucle sur i : n tours
- Boucle sur j : n tours

Un test dans chaque tour + un incrément si test validé

Total. kn^2 opérations, avec $k < 5$ un réel.

Comment comparer le temps d'exécution des algorithmes ?

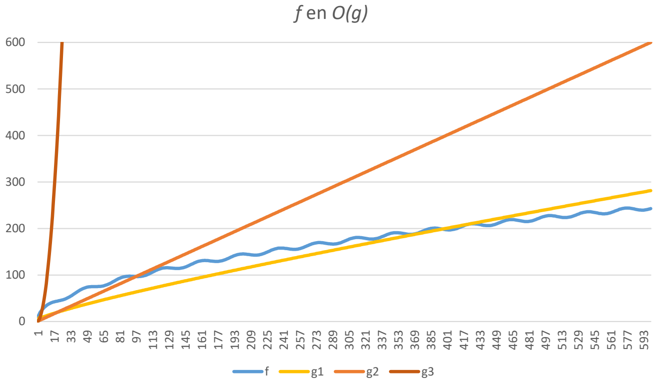
Définition (grand O). Soient f et g deux fonctions positives sur N. On dit que f est en grand O de g s'il existe deux entiers positifs c et n_0 tels que pour tout entier $n > n_0$, on a $f(n) < cg(n)$.

Autrement dit : à partir d'un certain rang n_0, f est dominée par g.

Notation (de Landau). Lorsque f est en grand O, on utilise la notation de Landau : $f = O(g)$

Exemple. Soit f la fonction de $\mathbb{R}^+$ dans $\mathbb{R}^+$ définie par $f(x) = 10\sqrt{x} + 4\sin\left(\frac{x}{6}\right)$

Exemple. La fonction f définie précédemment est en grand O de g_1, de g_2 et de g_3.



Règle. On recherche toujours la fonction de référence immédiatement dominante.

Notation	Appellation	Exemples d'algorithmes
$O(1)$	Constante	Temps d'accès à un élément d'un tableau
$O(\log N)$	Logarithmique	Jeu du juste prix (la vitrine)
$O(N)$	Linéaire	Parcours d'un tableau
$O(N \log N)$	Quasi-linéaire	Tri rapide (à découvrir au S2)
$O(N^2)$	Quadratique	Tris à bulles, insertion et sélection
$O(N^c)$	Polynomiale	$c > 2$, c parcours imbriqués
$O(c^N)$	Exponentielle	$c > 1$ énumération complète
$O(N!)$	Factorielle	Génération des permutations de N éléments

Méthode (Nombre d'opérations). On compte d'abord le nombre total d'opérations :

- identification des boucles : nombre de tours, imbrication ;
- nombre d'opérations par boucle : vérifier qu'il n'y a pas de surprise ;
- appels de fonction

Méthode (Déterminer la complexité finale). On se place souvent dans le pire des cas ou on calcule la complexité moyenne.

- logique de l'algo : itératif, récursif, dichotomie ?
- élimination des constantes ;
- identification de la fonction de référence la plus proche.

Taille des entrées. Définie par la taille N du tableau.

Algorithme – `Ibijau (tab)`
Entrée. `Tab. de n entiers T`
Sortie. `Entier res`

```
res ← 0
Pour i allant de 1 à n faire
    res ← res + i
    Pour j allant de 1 à i faire
        res ← res + T[i] + j
Renvoyer res
```

Nombre d'opérations.

- Boucle sur i : n tours, 1 affectation.
- Boucle sur j : i tours, 1 affectation.

Boucle sur i : n affectations.
Boucle sur j :

- quand $i = 1$: une affectation,
- quand $i = 2$: deux affectations ...

Total op. boucle j : $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

Total. Nombre d'opérations : $n + \frac{n(n+1)}{2}$ affectations. Complexité $= O(n^2)$.