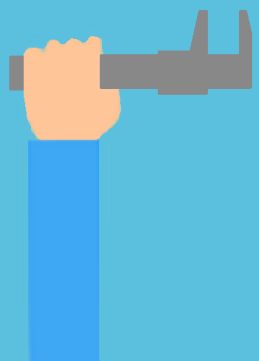
Complexité d'algorithmes

Renaud Costadoat
Lycée Dorian

DORIAN



Définition

Il existe souvent plusieurs façons de programmer un algorithme. Ainsi, lorsque le nombre d'opérations et la taille des données d'entrée deviennent importants, deux paramètres deviennent déterminants :

- le temps d'exécution,
- l'occupation mémoire.

- La **complexité en temps** donne le nombre d'opérations ou d'instructions effectuées lors de l'exécution d'un programme. On appelle C_o le coût en temps d'une opération o,
- La **complexité en mémoire** donne le nombre d'emplacements mémoires occupés lors de l'exécution d'un programme.

On distingue la complexité dans le pire des cas, la complexité dans le meilleure des cas, ou la complexité en moyenne.

- En effet, pour un même algorithme, suivant les données à manipuler, le résultat sera déterminé plus ou moins rapidement,
- Généralement, on s'intéresse au cas le plus défavorable à savoir, la complexité dans le pire des cas.

Exemple

Déterminer la position d'une valeur dans une liste.

Recherche naïve

```
def number_list(nb,tab):
    for i in range(len(tab)):
        if tab[i]==nb:
            return i
    return None
```

Recherche par dichotomie

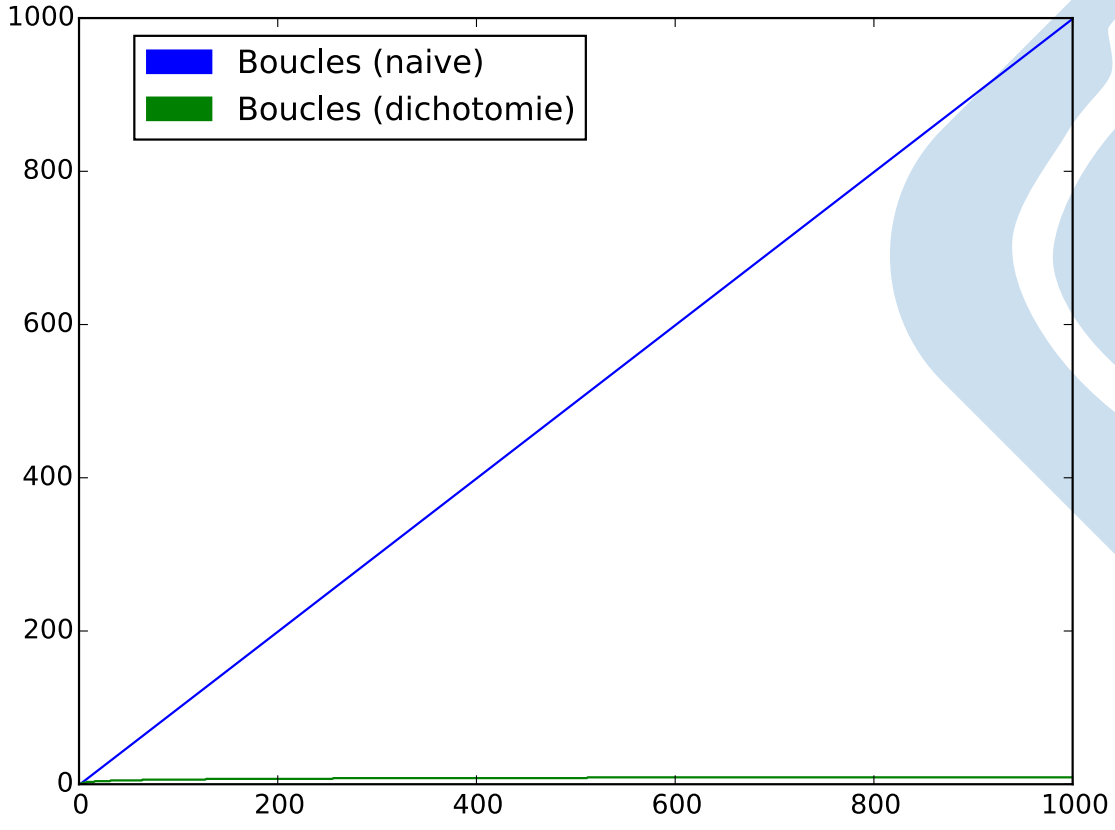
```
def number_list_dicho(nb,tab):
    g, d = 0, len(tab)-1
    while g <= d:
        m = (g + d) // 2
        if tab[m] == nb:
            return a
        if tab[m] < nb:
            g = m+1
        else:d = m-1
    return None
```

Résultats:

```
>>> nb=25
>>> tab=range(1,100,2)
>>> number_list(nb,tab)
12
>>> number_list_dicho(nb,tab)
12
```

Exemple

Le graphe présente le nombre maximal de boucles pour trouver un nombre en fonction de la taille de la liste. La complexité en temps n'est pas la même dans les deux cas.



Coût temporel

On considère que le coût élémentaire C_e correspond au coût d'une affectation, d'une comparaison ou de l'évaluation d'une opération arithmétique.

```
>>> a=20 #(1 Ce)
>>> a<=100 #(1 Ce)
>>> a=a+1 #(2 Ce)
```

Pour une séquence de deux instructions de coûts respectifs C_1 et C_2, le coût total est de la séquence est de $C_1 + C_2$. Le coût d'un affichage est plus important qu'un coût élémentaire.

```
>>> a=20 #(1 Ce)
>>> print(a)  #(1 Ca)
```

Le coût d'un test *if test* : $inst_1$ *else* : $inst_2$ est inférieur ou égal au maximum du coût de l'instruction 1 et du coût de l'instruction 2 additionné au coût du test (coût élémentaire).

```
>>> if x<0 :
        x=x+1
        x=x+2
    else :
        x=x+1  #(5 Ce)
```

Calcul de coût

Soit le programme suivant qui permet de calculer la factorielle d'un nombre.

```
def factorielle(n) :
    if  n==0:
        return 1
    else:
        i, res =1, 1
        while i<=n:
            res=res*i
            i=i+1
        return res
```

Déterminer:

- la complexité en mémoire,
- la complexité en temps.

Complexité en mémoire : n, res, i.

Complexité en temps:

- $n == 0$ $(1.C_e)$,
- $i, res = 1, 1$ $(2.C_e)$,
- n fois $i <= n$, $res = res * i$, $i = i + 1$ $(5.n.C_e)$,
- *return res* $(1.C_r)$.

En conséquence, la complexité en temps s'élève à : $C_T(n) = C_e + 2 * Ce + n * 5 * Ce + Cr$, ainsi, lorsque le nombre de boucles devient important: $C_T \underset{n \to \infty}{\sim} n * 5 * Ce$.

Il s'agit donc d'une complexité algorithmique **linéaire**, notée O(n).

Calcul de coût

Soit le programme suivant qui permet de classer une liste.

```
for i in range(0,len(tab)-1):
    min=i
    for j in range(i+1,len(tab)):
        if tab[j] < tab[min]:
            min=j
    tmp=tab[i]
    tab[i]=tab[min]
    tab[min]=tmp
print tab
```

Déterminer:

- la complexité en mémoire,
- la complexité en temps.

Ici les bornes de la boucle imbriquée dépendent de l'indice i. Ainsi :

- au rang 1, $C_1 = C_e + (n-1)(2.Ce) + 3.Ce$,
- au rang 2, $C_2 = C_e + (n-2)(2.Ce) + 3.Ce$,
- au rang i, $C_i = C_e + (n-i)(2.Ce) + 3.Ce$.

Le coût temporel peut donc s'exprimer ainsi :

$$C_T(n) = \sum_{i=0}^{n-1} C_e + (n-i)(2.C_e) + 3.C_e = C_e.\sum_{i=0}^{n-1} 4 + 2.n - 2.i = Ce.\left(4.n + 2.n^2 - 2.\frac{n.(n-1)}{2}\right) =$$

$C_e.(3.n + n^2)$. Dans ce cas, $C_T \underset{n\to\infty}{\sim} C_e.n^2$.

Les types de complexités

Par ordre de complexité croissante on a donc :

- $O(1)$: algorithme s'exécutant en temps constant, quelle que soit la taille des données,
- $O(log(n))$: algorithme rapide (complexité logarithmique), (Exemple : recherche par dichotomie dans un tableau trié),
- $O(n)$: algorithme linéaire,
- $O(n.log(n))$: complexité n.log(n),
- $O(n^2)$: complexité quadratique,
- $O(2^n)$: complexité exponentielle.