# Études d'algorithmes classiques

## Objectifs

A la fin de la séquence d'enseignement l'élève doit pouvoir écrire et analyser les algorithmes suivants:

- recherche dans une liste
- recherche du maximum dans une liste de nombres
- calcul de la moyenne et de la variance
- recherche d'un mot dans une chaîne de caractères.

## Table des matières

# 1 Indice du maximum

## 1.1 Algorithme

**Algorithm 1** Indice du maximum

```
    entrée: T un tableau de n valeurs
    résultat: iMax l'indice du maximum du tableau
    indiceMax(T)
 1: n=taille(T)
 2: iMax=0
 3: max ← T[0]
 4: pour i entre 1 et n-1 faire
 5:     si T[i]> max alors
 6:         iMax ← i
 7:         max ← T[i]
 8:     fin si
 9: fin pour
10: renvoi: iMax
```

## 1.2 Invariant de boucle et correction

La variable `max` contient le maximum de `T[0:i]` et `iMax` l'indice de sa première apparition.

## 1.3 Terminaison

Absence de boucle conditionnelle *Tant que* . Pas de récursivité. Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

## 1.4 Complexité

- au mieux $n+1$ (`max` en première position)
- au pire $3.n$ (tableau strictement croissant).

Donc $C(n) = \Theta(n)$.

# 2 Recherche dans un tableau

## 2.1 Algorithme

**Algorithm 2** Recherche dans un tableau

```
    entrée: T un tableau de n valeurs et x un élément
    (pas forcément du tableau)
    résultat: -1 si x ∉ T, l'indice de la première occur-
    rence de x dans le tableau T
    recherche(T,x)
 1: i ← 0
 2: n← taille(T )
 3: tant que i<n et T[i]≠x faire
 4:     i ← i+1
 5: fin tant que
 6: si i=n alors
 7:     i ←-1
 8: fin si
 9: renvoi: i
```

## 2.2 Invariant de boucle et correction

A chaque itération, $x$ n'apparaît pas dans les $i+1$ premières cases du tableau.

## 2.3 Terminaison

La suite des valeurs prises par $i$, entier, est strictement croissante et majorée par $n$.

## 2.4 Complexité

- au mieux 2 (tableau vide) ou 3 ($x$ en première position)
- au pire $2.n+2$ ($x$ n'apparaît pas).

Donc $C(n) = O(n)$.

# 3 Moyenne et variance

## 3.1 Rappels

$$Moy(\mathrm{T}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{T}[i] \text{ et } Var(\mathrm{T}) = \left(\frac{1}{n}\sum_{i=1}^{n} \mathrm{T}[i]^2\right) - Moy(\mathrm{T})^2$$

## 3.2 Algorithme

**Algorithm 3** Moyenne et variance

**entrée:** T un tableau de $n$ valeurs
**résultat:** $Moy$, la moyenne de T
$Var$, la variance de T
MoyVar(T)

```
1: n← taille(T )
2: s← 0
3: sc ← 0.
4: pour i entre 0 et n-1 faire
5:     s← s+T[i]
6:     sc ← sc+T[i]^2
7: fin pour
8: moy ← s/n
9: renvoi: moy, sc/n-moy^2
```

## 3.3 Invariant de boucle et correction

A chaque itération $i$ : $s = \sum_{j=0}^{i-1} \mathrm{T}[j]$ et $sc = \sum_{j=0}^{i-1} \mathrm{T}[j]^2$

## 3.4 Terminaison

Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

## 3.5 Complexité

Dans la boucle `Pour`, il faut :

- ajouter `T[i]` à `s` : C+ = 1
- calculer le carré de `T[i]` : C+ = 1
- l'ajouter à `sc` : C+ = 1
- faire 3 affectations : C+ = 3

A chaque itération de la boucle, le coût C est donc de 6. Avec les différentes affectations en début et en fin de programme, on obtient un coût total $C(n)$ :

$$C(n) = 6.n + 7 = \Theta(n)$$

# 4 Recherche dichotomique dans un tableau trié

## 4.1 Algorithme

**Algorithm 4** Recherche dichotomique dans un tableau trié

**entrée:** T un tableau de $n$ valeurs et $x$ un élément (pas forcément du tableau)
**résultat:** -1 si $x \notin$ T, l'indice d'un $x$ dans le tableau T

```
1: DichoSearch(T,x)
2: n← taille(T )
3: min,max,mil← 0,n-1,(min+max)//2
4: tant que min<max et T[mil]≠x faire
5:     si T[mil]<x alors
6:         min← mil+1
7:     sinon
8:         max ← mil-1
9:     fin si
10:    mil← (min+max)//2
11: fin tant que
12: si T[mil]≠x alors
13:     rep ← -1
14: sinon
15:     rep ← mil
16: fin si
17: renvoi: rep
```

## 4.2 Invariant de boucle

**Proposition** $\mathcal{P}(k)$ **:** *à la fin de l'étape k (donc si x n'a pas été trouvé)* $max - min < \frac{n}{2^k}$ *et* ***si*** *x* ***est dans le tableau*** T *alors*

$$\mathrm{T}[min] \leq x \leq \mathrm{T}[max]$$

**Démonstration :** par récurrence :

- A la fin de l'étape 0, donc juste avant la boucle, si $x$ est dans le tableau (trié), on a bien

$$T[0] = T[min] \leq x \leq T[max] = T[n-1]$$

$$\text{et} \quad max - min = n - 1 < \frac{n}{2^0} = n$$

- Si $\mathcal{P}(k)$ est vraie à la fin de l'étape $k$, appelons $a$ la valeur de $min$, $b$ celle de $max$ tels que $b - a < \frac{n}{2^k}$. Alors $mil$ devient :

$$mil = \left\lfloor \frac{min + max}{2} \right\rfloor = \left\lfloor \frac{a+b}{2} \right\rfloor$$

Si $T[mil] \neq x$, on ne sort pas de la boucle, et

- soit $T[mil] < x$, alors $min$ prend la valeur $mil + 1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1$ et $max$ reste à $b$.

  **Si $x$ est dans le tableau (trié)**, on a alors bien $T[min] \leq x \leq T[max]$, et

$$max - min = b - \left\lfloor \frac{a+b}{2} \right\rfloor - 1 < b - \left\lfloor \frac{a+b}{2} \right\rfloor \leq \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

- soit $T[mil] > x$, alors $max$ prend la valeur $mil - 1 = \left\lfloor \frac{a+b}{2} \right\rfloor - 1$ et $min$ reste à $a$.

  **Si $x$ est dans le tableau (trié)**, on a alors bien $T[min] \leq x \leq T[max]$, et

$$max - min = \left\lfloor \frac{a+b}{2} \right\rfloor - 1 - a < \left\lfloor \frac{a+b}{2} \right\rfloor - a \leq \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

d'où la récurrence. CQFD

### 4.3 Terminaison

Comme la différence $max - min = Err$ est à valeur dans $\mathbb{N}$ ($Err \in \mathbb{N}$) et que $max - min < \frac{n}{2^k}$, si $k$ assez grand $max - min = 0$.

### 4.4 Correction

Soit on tombe sur $x$ à un moment, soit on arrive à $min = max$, et si $x$ est dans le tableau, $T[min] \leq x \leq T[max]$. Le résultat renvoyé est bien le bon.

### 4.5 Complexité

A chaque étape de la boucle `Tant que` , il se produit :

- deux tests : C+ = 2
- une affectation (cas **si** ou **sinon**) : C+ = 1
- une somme + une division+une affectation : C+ = 3

Or $x$ se loge dans un intervalle $max - min < \frac{n}{2^k}$ et on s'arrête au plus tard quand $min = max$. Ainsi, au plus, on a $k$ itérations telles que

$$\frac{n}{2^k} < 1 \Rightarrow 2^k > n \Rightarrow k.\ln(2) > \ln(n) \Rightarrow k > \log_2(n)$$

donc au plus $\lfloor \log_2(n) \rfloor + 1$ itérations. Ainsi, le coût pour la boucle est $C(n) = 6.(\log_2(n) + 1) = O(\ln(n))$.

**Remarque :** le cas de la recherche dichotomique dans un tableau trié de flottants ressemble beaucoup. Il suffit d'ajouter une précision $\epsilon > 0$ en argument et de remplacer $T[mil] = x$ par $|T[mil] - x| < \epsilon$.

# 5 Recherche d'un mot dans une chaîne de caractères

## 5.1 Principe

Pour écrire un algorithme *naïf* de recherche d'un mot dans une chaîne de caractères (`ChercheMot(mot,chaine)`), on utilise une fonction `TestMot(mot,chaine,i)` qui teste la coïncidence entre les lettres de `mot` à une position i de `chaîne` jusqu'à trouver une lettre différente dans la successions des lettres. Il renvoie alors `vrai` si toutes les lettres de `mot` se succèdent dans `chaîne` à la position i ; `faux` sinon.

La fonction `ChercheMot(mot,chaine)` balaye les lettres chaînes en appelant pour chaque lettre la fonction `TestMot` jusqu'à ce que cette dernière renvoie `vrai` ou que le nombre de lettres de `chaine` restant soit inférieur à la taille de `mot`.

## 5.2 Algorithmes

**Algorithm 5** Recherche d'un mot dans une chaîne de caractères.

**entrée:** *mot* et *chaine* deux chaînes de caractères
**résultat:** l'indice de la première occurrence de *mot* dans *chaine* ; sinon -1
ChercheMot(mot,chaine)

```
1: i,n,m,pos,pastrouv← 0,taille(chaine),taille(mot),0,vrai
2: tant que i≤n-m et pastrouv faire
3:     pastrouv← non(TestMot(mot ,chaine, i ))
4:     i←i+1
5: fin tant que
6: si pastrouv alors
7:     i ←-1
8: fin si
9: renvoi: i
```

**Algorithm 6** Teste la présence d'un mot à partir d'une position dans une chaîne de caractères.

**entrée:** *mot* et *chaine* deux chaînes de caractères, *i*, un indice
**résultat:** *vrai* si *mot* correspond aux premiers caractères de *chaine* à partir de l'indice *i* ; *faux* sinon

```
1: TestMot(mot,chaine,i)
2: j,m ← 0,taille(mot )
3: tant que  j<m et mot [j]=chaine[i+j] faire
4:     j← j+1
5: fin tant que
6: renvoi: j==m
```

## 5.3 Invariant de boucle et correction

- pour `Testmot` : les $j$ premières lettres de `mot` se suivent dans `chaine`
- pour `ChercheMot` : `mot` ne se trouve pas dans les $i$ premières positions de `chaine`

## 5.4 Complexité

En comparaison :

- pour `Testmot` : la boucle est parcourue au plus $m$ fois.
- pour `ChercheMot` : la boucle est parcourue au mieux 1 fois ; au pire, $n-m$ fois. Avec l'appel de `Testmot` $C(n) = m.(n-m+1) = O(m.(n-m))$

# 6 Complément : l'exponentiation

**Objectif :** calculer $x^n$ où $n$ est un entier positif et $x$ un réel.

## 6.1 Exponentiation « naïve »

### 6.1.1 Algorithme

**Algorithm 7** Exponentiation « naïve »

**entrée:** $n$ un entier positif et $x$ un nombre réel
**résultat:** un nombre réel $r = x^n$
Exponaive(x,n)

```
1: si n==0 alors
2:     renvoi: 1
3: sinon
4:     r← x
5:     pour i de 2 à n faire
6:         r← x
7:     fin pour
8: fin si
```

### 6.1.2 Principe

Multiplier $x$ $n$ fois par lui même.

### 6.1.3 Terminaison

Présence uniquement d'une boucle inconditionnelle *Pour*. Le nombre d'itérations est fini.

### 6.1.4 Invariant de boucle et correction

A chaque itération $i$, $r = x^i$.

### 6.1.5 Complexité

A chaque itération, il faut faire une multiplication $C(n) = \Theta(n)$.

## 6.2 Exponentiation rapide itérative

### 6.2.1 Algorithme

**Algorithm 8** Exponentiation rapide itérative

**entrée:** $n$ un entier positif et $x$ un nombre réel
**résultat:** un nombre réel $r = x^n$
ExpoRapide(x,n)

```
1: si n==0 alors
2:     renvoi: 1
3: sinon
4:     r← 1
5:     tant que n > 0 faire
6:         si n modulo 2==1 alors
7:             r←r.x
8:         fin si
9:         x←x.x
10:        n←n//2
11:    fin tant que
12:    renvoi: r
13: fin si
```

### 6.2.2 Principe

Se servir des résultats intermédiaires pour accélérer le processus de calcul de $x^n$ en décomposant $n$ en puissance de 2.

**Exemple :** calculer $x^5$. On calcule $x^2$ puis $x^4$ et finalement $x^5$.

### 6.2.3 Terminaison

La boucle conditionnelle s'arrête dès que $n$ cesse d'être strictement positif. Or $n$ est une suite d'entier (division euclidienne) décroissante puisque $n$ est divisé par 2 à chaque itération.

La suite est strictement décroissante et minorée ; l'algorithme termine.

### 6.2.4 Propriété

on appelle $r_i$, $x_i$ et $n_i$ les valeurs de $r$, $x$ et $n$ après l'itération $i$ dans la boucle `while`. Avec $i = 0$, on a $r_0 = 1$, $x_0 = x$ et $n_0 = n$.

On note P($k$), la propriété : **Proposition** $\mathcal{P}(i)$ **:** *à la fin de l'étape* $i$, $x^n = r_i.x_i^{n_i}$

**Démonstration :**

- au rang 0 : montrons $\mathcal{P}(0)$. Or $r_0.x_0^{n_0} = 1.x^n = x^n$la propriété est donc vraie au rang 0
- au rang $i$ : montre que $\mathcal{P}(i-1) \quad \Rightarrow \quad \mathcal{P}(i)$
  Supposons $\mathcal{P}(i-1)$ vraie : $x^n = r_{i-1}.x_{i-1}^{n_{i-1}}$, alors :
  - soit $n_{i-1} \pmod 2 == 1$ et $r_i = r_{i-1}.x$, $x_i = x_{i-1}^2$ et $n_{i-1} = 2.n_i+1$ d'où $r_i.x_i^{n_i} = r_{i-1}.x.x_{i-1}^{2.n_i} = \dfrac{x^n}{x_{i-1}^{n_{i-1}}}.x_{i-1}^{2.n_i+1} = x^n$
  - soit $n_{i-1} \pmod 2 == 0$ et $r_i = r_{i-1}$, $x_i = x_{i-1}^2$ et $n_{i-1} = 2.n_i$ d'où $r_i.x_i^{n_i} = r_{i-1}.x_{i-1}^{2.n_i} = \dfrac{x^n}{x_{i-1}^{n_{i-1}}}.x_{i-1}^{2.n_i} = x^n$

  ainsi $\mathcal{P}(i-1) \quad \Rightarrow \quad \mathcal{P}(i)$

### 6.2.5 Invariant de boucle et correction

A chaque itération $i$, $\mathcal{P}(i)$ est vraie. Or pour $k \in \mathbb{N}$ tel que $n_k = 0$, $\mathcal{P}(k)$ étant vraie $x^n = r_k.x_k^{n_k} = r_k$. L'algorithme retournant $r_k$ puisque $n_k = 0$, la valeur renvoyée est donc bien $x^n$

### 6.2.6 Complexité

A chaque itération $i$, $n_i = \left\lfloor \dfrac{n_{i-1}}{2} \right\rfloor \leq \dfrac{n_{i-1}}{2} \leq \dfrac{n}{2^i}$. Ainsi, si $m$ est le nombre d'itérations permettant de calculer $x^n$ alors :

$$1 = n_{m-1} = \frac{n}{2^{m-1}} \quad \Rightarrow \quad (m-1).\ln(2) \leq \ln(n) \quad \Rightarrow \quad C(m) = O(\ln(m))$$

## 6.3 Exponentiation rapide récursive

### 6.3.1 Principe

Se servir des résultats intermédiaires pour accélérer le processus de calcul de $x^n$ en décomposant $n$ en puissance de 2.

### 6.3.2 Algorithme

**Algorithm 9** Exponentiation rapide récursive

**entrée:** $n$ un entier positif et $x$ un nombre réel
**résultat:** un nombre réel $r = x^n$
ExpoRec(x,n)

```
si n==0 alors
    renvoi: 1
sinon
    si n modulo 2==0 alors
        renvoi: ExpoRec(x.x,n/2)
    sinon
        renvoi: x.ExpoRec(x.x,(n-1)/2)
    fin si
fin si
```

### 6.3.3 Terminaison

$n$ décrit une suite d'entiers strictement décroissante et minorée. L'algorithme se termine.

### 6.3.4 Complexité

- si $n = 1$, un test, un produit, une division $C(1) = 3$
- sinon, un test, un produit, une division : $C(n) = C(n/2) + 3$

On suppose l'algorithme logarithmique.

- on pose alors :$C(n) = a.\log_2(n) + c$

$$\begin{aligned} C(n/2) &= a.\log_2(n/2) + c = a.\log_2(n) - a.\log_2(2) + c \\ \bullet \quad &= a.\log_2(n) - a + c \\ C(n) &= a.\log_2(n) - a + c + 3 = a.\log_2(n) + c \end{aligned}$$

- on a donc $a = 3$
- or $3 = C(1) = a.\log_2(1) + c = c$ donc $c = 3$
- $C(n) = 3.\log_2(n) + 3 = \Theta(\ln(n))$

L'algorithme est donc bien de complexité logarithmique.