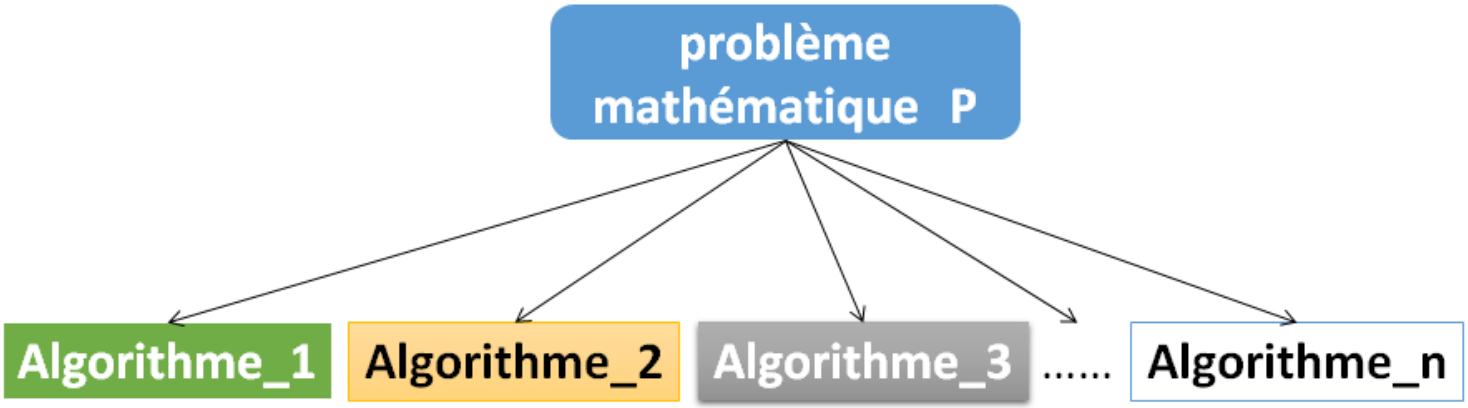
Plan...

- Problématique
- Notion de complexité
- Comment calculer la complexité d'un algorithme
- Etude des cas : Exemple de calculs de complexité

Problématique

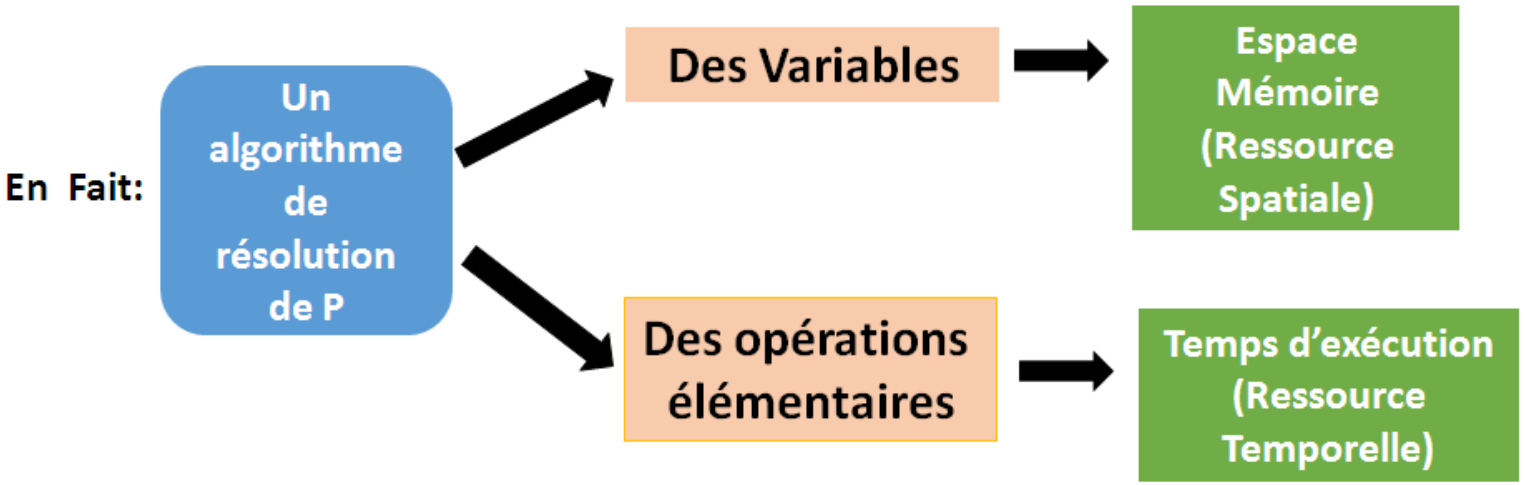


On suppose que ce **n** algorithmes sont

- **Justes** (Donnent le bon résultat correspondante au problème P),
- **Compréhensibles** (bien documentés).

→ **Lequel choisi-t-on ? Et Comment établir ce choix ?**

Problématique (2)



Un bon algorithme doit donc utiliser le minimum de ressources Spatiales et Temporelles

ALGORITHME OPTIMAL

Problématique (3)

Comment trouver l'algorithme optimal ?

↓

1. Calculer les complexités (ou coûts) des algorithmes résolvant le même problème
2. Choisir l'algorithme ayant le minimum de complexité (ou coût)

↓

Mais c'est quoi la complexité d'un algorithme ?

Notion de complexité (1)

Définition.

La complexité d'un algorithme est la mesure de la quantité Q de ressources nécessaires à son exécution. C'est donc une mesure intrinsèque de sa rapidité d'exécution.

Deux types de ressources:

- ✓ ***Spatiale:*** **Mémoire nécessaire** ➡ **Complexité Spatiale.**
- ✓ ***Temporelle:*** **Nombre d'opérations élémentaires** (affectation, +, *, -, comparaison) **exécutées par l'algorithme** ➡ **Complexité Temporelle.**

➡ **Dans ce cours on s'intéresse seulement à la complexité temporelle**

Notion de complexité (2): Exemple

Exemple1:

Echange des valeurs des deux variables entières x et y ➡ Deux Solutions possibles

Algorithme A1
Entrées: x, y: entier
Sorties: x, y :entier

```
Début
    Lire(x,y)
    temp ← x
    x ← y
    y ← temp
    Ecrire(x,y)
Fin
```

Ou

Algorithme A2
Entrées: x, y: entier
Sorties: x, y :entier

```
Début
    Lire(x,y)
    x ← x + y
    y ← x - y
    x ← x - y
    Ecrire(x,y)
Fin
```

➡

Algorithme	Complexité Spatiale CS	Complexité Temporelle CT
A1	3 Variables Entières	3 Affectations
A2	2 Variables Entières	3 Affectations + 3 additions

Conclusion.

CS(A2) < CS(A1) ➔ A2 est mieux que A1 en complexité Spatiale

CT(A2) > CT(A1) ➔ A1 est mieux que A2 en complexité Temporelle

A1 est l'algorithme Optimale : Choix de A1

Notion de complexité (2): Exemple

Exemple2: **Somme des n premiers entiers** ➡ **Deux Solutions possibles**

➢ ***Cas 1: Utilisation d'une boucle***

Algorithme A1
Entrées: n: entier
Sorties: S : entier

```
Début
    Lire(n)
    S ← 1
    i ← 1
    Tant que i <= n
    Faire
        S ← S + i
        i ← i + 1
    Ecrire(S)
Fin
```

Complexité Temporelle (Nombre d'opétions Elémentaires)	
Lire(n)	Une opération élémentaire qui se répète 1 fois
S ← 1	Une opération élémentaire qui se répète 1 fois
i ← 1	Une opération élémentaire qui se répète 1 fois
i <= n	Une opération élémentaire qui se répète n fois
S ← S + i	**Deux** opérations (←, +) élémentaires qui se répète n fois
i ← i + 1	**Deux** opérations (←, +) élémentaires qui se répète n fois

✓ **La complexité temporelle dépend de la variable entrée n:**
Elle est notée C(n)

✓ **La complexité temporelle de A1 est donc :**

$$C(n) = 1+1+1+n(1+1+1+1+1)$$

➡ $C(n) = 5n + 3$

Notion de complexité (2): Exemple

Exemple2: **Somme des n premiers entiers** ➡ **Deux Solutions possibles**

➢ ***Cas 2: Utilisation de la formule Mathématique: ∑ = n*(n-1)/2***

Algorithme A2
Entrées: n: entier
Sorties: S : entier

```
Début
    Lire(n)
    S ← n*(n-1)/2
    Ecrire(S)
Fin
```

➡

Complexité Temporelle (Nombre d'opétions Elémentaires)

Lire(n) — **Une** opération élémentaire qui se répète 1 fois
S ← n*(n-1)/2 — **Quatre** opérations élémentaires (affectation, addition, soustraction et division) qui se répètent 1 fois

⬇

✓ **La complexité temporelle de A1 est donc :**

$$C(n) = 1+(1+1+1+1) = 5$$

Pour n assez grand : complexité(A2) < Complexité(A1)

A2 est l'algorithme optimal à retenir.

CONCLUSION 1ère Partie

➡ La complexité C(n) d'un algorithme dépend d'une variable d'entrée n

➡ Les algorithmes s'exécutent souvent sur des données (entrées) de grandes tailles :

- Le calcule exacte de la complexité n'est trop intéressant
- Une approximation asymptotique de la complexité qui définit son Ordre de grandeur est suffisante : Elle est notée O(...)

➡
- ✓ Pire des cas (Nombre maximal d'opérations élémentaires effectuées)
- ✓ Moyenne des cas (Nécessite la connaissance de la distribution des données)
- ✓ Meilleure des cas (Nombre minimal d'opérations élémentaires effectuées)

Dans notre étude on s'intéresse au : **Pire des cas**

Comment calculer la complexité C(n) d'un algorithme ? (1)

Technique : INCREMENTER et COMPARER | **Comparer : C(n+1) et C(n)**

$C(n+1) \approx C(n)$	: Complexité Constante	: $O(1)$
$C(n+1) \approx C(n) + 1$	: Complexité Linéaire	: $O(n)$
$C(n+n) \approx C(2n) \approx C(n) + 1$	: Complexité Logarithmique	: $O(\log_2(n))$
$C(n+1) \approx C(n) + n$	: Complexité Polynomiale	: $O(n^2)$
$C(n+1) \approx 2*C(n)$	: Complexité Exponentielle	: $O(2^n)$

Remarque: Echelle de fonction de complexité.

$$\log_2(n) < n < n\log_2(n) < n^2 < n^3 \ldots\ldots < k^n < n! < n^n$$

Comment calculer la complexité C(n) d'un algorithme ? (2)

Règles générales à respecter

1. La complexité d'une opération élémentaire est considéré comme constant = 1,
2. La complexité d'une séquence d'instructions est la somme des complexités des instructions élémentaires qui la composent,
3. La complexité d'un branchement conditionnel est égal à la complexité du test plus le max des deux complexités correspondant aux deux alternatives (au Pire des cas).
4. La complexité d'une boucle est égal à la somme: complexité du test + complexité du corps de la boucle + complexité test de sortie de boucle.

Etude de cas: Complexité Constante

Algorithme :
Entrées: n: entier

```
Var n: entier
Début
    Lire(n)
    (1)
    Pour i de 1 à 10 Faire
    (10)
        Ecrire(i)
    FinPour
Fin
```

Lire(n)	: Une opération élémentaire qui se répète 1 fois
Incrémentation de i	: Une opération élémentaire qui se répète 10 fois

$C(n) = 1+1*10 = 11$
$C(n+1) = 1 + 1*10 = 11$

$\Rightarrow C(n+1) = C(n)$

Complexité indépendante de la valeur de n

➡ Complexité Constante : O(1)

Autres Exemples:

- ✓ Chercher le premier élément d'un tableau de taille n
- ✓ Vérifier la parité d'un nombre entier n

Etude de cas: Complexité Linéaire

```
Algorithme :
  Entrées: n: entier
Var n: entier
Début
    Lire(n)     (1)
    Pour  i de 1 à n Faire  (n)
       Ecrire(i)
    FinPour
Fin
```

Lire(n)	: Une opération élémentaire qui se répète 1 fois
Incrémentation de i	: Une opération élémentaire qui se répète n fois

$C(n) = 1+1*n = n+1$

$C(n+1) = 1 + 1*(n+1) = 1+n+1$ → $C(n+1) = C(n)+1$

Complexité dépend de la valeur de n

➡ **Complexité Linéaire : O(n)**

Autres Exemples:

- ✓ Chercher un élément dans tableau non ordonné de taille n
- ✓ Calcul de la longueur d'un tableau de taille n

Etude de cas: Complexité Polynômiale

Algorithme :
Entrées: n: entier

```
Var n: entier
Début
    Lire(n)                    (1)
    Pour i de 1 à n Faire      (n)
      Pour j de 1 à n Faire    (n*n)
        Ecrire(i)
      FinPour
    FinPour
Fin
```

$C(n) = 1+n+n*n = n^2+n+1$

$C(n+1) = 1 + (n+1)+(n+1)*(n+1)$
$= (n^2+n+1)+2(n+1)$

➡ $C(n+1) \approx C(n)+n$ quand n↑

➡ Complexité Polynomiale : $O(n^2)$

Lire(n)	: Une opération élémentaire qui se répète 1 fois
Incrémentation de i	: Une opération élémentaire qui se répète n fois
Incrémentation de j	: Une opération élémentaire qui se répète n*n fois

Autre Exemple: Chercher le maximum dans une matrice carrée de taille n*n

Etude des cas: Complexité Exponentielle :

Algorithme :
Entrées: n: entier

```
Var n: entier
Début
    Lire(n)                    (1)
    m ← n                      (1)                    (n)
    Pour i de 1 à n Faire      (2)
        j ← 1                  (1)
        Tant que j <=m Faire   (1)
            Ecrire(j)  (à ignorer)                    (m)
            j ← j+1            (1)
        FinTantque
        m←m*2                  (2)
    FinPour
Fin
```

n	i	m	Total Répétition Boucle Tantque
3	1, 2, 3	3, 6, 12	3 + 6 + 12 = 21 fois

$C(n) = (1+1)+(2+1+2)]*3 + 2*(3+6+12) = 59$

n	i	m	Total Répétition Boucle Tantque
4	1, 2, 3, 4	4, 8, 16, 32	4 + 8 + 16 + 32 = 60 fois

$C(n+1) = (1+1)+(2+1+2)]*4 + 2*(4+8+16+32) = 142 \approx 2*C(n)$

n	i	m	Total Répétition Boucle Tantque
5	1, 2, 3, 4, 5	5, 10, 20, 40, 80	5+10+20+40+80 =155 fois

$C(n+2) = (1+1)+(2+1+2)]*5 + 2*(5+10+20+40+80) = 337$
$\approx 2*C(n+1)$

$C(n+1) \approx 2*C(n)$ ➡ Complexité Exponentielle : $O(2^n)$

Etude des cas: Complexité Logarithmique :

Algorithme :
Entrées: n: entier

```
Var n: entier
Début
    Lire(n)                    (1)
    i ← 1                      (1)
    m ← n                      (1)        (?)
    Tant que i <=m Faire       (1)
        Ecrire(j)  (à ignorer)
        m← m div 2             (1)
        i ← i+1                (1)
    FinTantque
    FinFaire
Fin
```

n	i avant arrêt	m	Total Répétition Boucle Tant que
10	1, 2	10, 5, 2	2 fois

$C(n) = (1+1+1) + 2^* (1+1+1) = 9$

n	i	m	Total Répétition Boucle Tant que
20	1, 2, 3	20, 10, 5, 2	3 fois

$C(2n) = (1+1+1) + 3^*(1+1+1) = 12 \approx C(n) + 1$

n	i	m	Total Répétition Boucle Tant que
40	1, 2, 3, 4	40, 20, 10, 5, 2	4 fois

$C(4n) = (1+1+1) + 4^*(1+1+1) = 15 \approx C(2n)+1$

$C(2n) \approx C(n)+1$ ➡ Complexité logarithmique : $O(\log_2(n))$

Autre exemple: recherche d'un élément dans un tableau ordonné (Recherche Dichotomique)