

EPFL Rappel : ingrédients de base des algorithmes

Données

- Entrées
- Sorties
- Variables internes

Instructions

- Affectations
- Structures de contrôle
 - Branchements conditionnels (tests)
 - Itérations (boucles)
 - Boucles conditionnelles



Information, Calcul et Communication

Sous-algorithmes

Olivier Lévêque



Kathryn Greenhill [CC BY-SA 2.0 (<https://creativecommons.org/licenses/by-sa/2.0/>)]

Un problème récurrent :
comment préparer ses
valises en famille ?

Solution 1

Algorithme centralisé : une personne se charge de tout: pas idéal...

Solution 2

Chacun prépare sa propre valise: beaucoup mieux !

Et encore mieux: chaque personne prépare séparément :

- ses habits
- sa trousse de toilette
- ses livres
- sa tenue de plongée ...

Un exemple en pseudocode

algorithme principal

entrée : ...
sortie : ...

...
 $x \leftarrow \mathbf{algo1}(5)$
 $k \leftarrow 10$
 $y \leftarrow \mathbf{algo2}(k)$
 $z \leftarrow x + \mathbf{algo1}(y)$
...

algo1

entrée : nombre entier n
sortie : nombre entier $s1$

(instructions)

algo2

entrée : nombre entier n
sortie : nombre entier $s2$

(instructions)

Illustration du principe avec le tri d'une liste

Tri d'une liste de nombres

Comment trier une liste de nombres ? (ou un jeu de cartes, ou encore une liste de noms, ou ...)

Il existe de nombreuses façons de faire, plus ou moins efficaces. Nous allons en voir une: le **tri par insertion**, qui permet de bien illustrer le principe de l'utilisation de sous-algorithmes.

Tri par insertion: algorithme principal

tri par insertion

entrée : liste de nombres L , taille de la liste n
sortie : liste L triée dans l'ordre croissant

Pour i allant de 2 à n :
 Si $L(i) < L(i - 1)$, alors :
 $L \leftarrow \text{insérer}(L, i)$
Sortir : L

Tri par insertion: sous-algorithme 1

insérer

entrée : liste de nombres L , nombre entier positif i
sortie : liste L avec l'élément $L(i)$ bien placé

$j \leftarrow i$

Tant que $j > 1$ & $L(j) < L(j - 1)$:

$L \leftarrow \text{permuter}(L, j, j - 1)$

$j \leftarrow j - 1$

Sortir : L

Remarque importante :

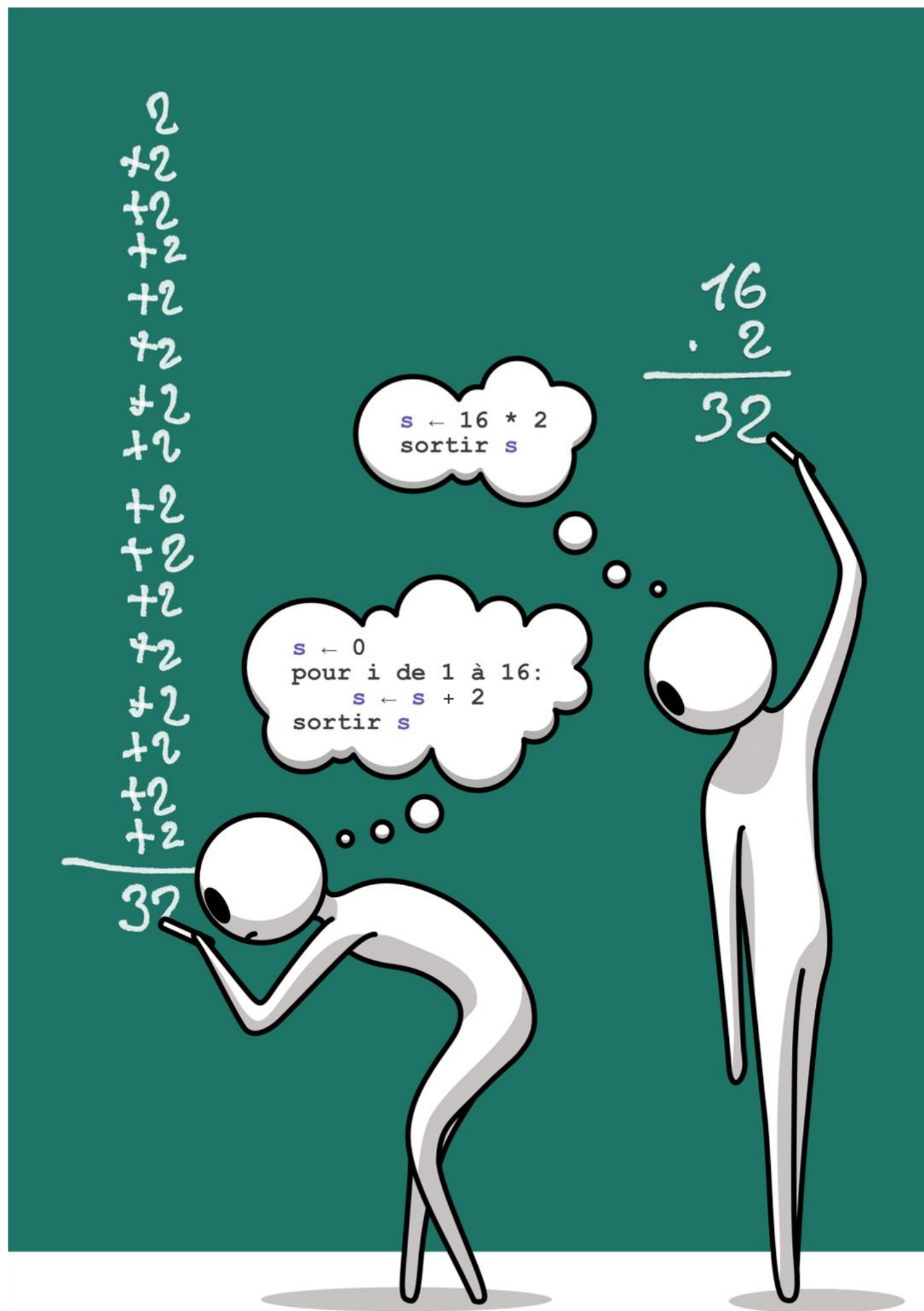
Les éléments $L(1) \dots L(i - 1)$ doivent être déjà triés pour que ce sous-algorithme fonctionne correctement. Heureusement, c'est le cas ici !

Tri par insertion: sous-algorithme 2

permuter

entrée : liste de nombres entiers L , nombres entiers positifs j, k
sortie : liste L avec les éléments $L(j)$ et $L(k)$ permutés

$temp \leftarrow L(j)$
 $L(j) \leftarrow L(k)$
 $L(k) \leftarrow temp$
Sortir : L



Information, Calcul et Communication

Algorithmes :
complexité temporelle

Olivier Lévêque

Complexité temporelle d'un algorithme

La complexité temporelle d'un algorithme est son temps d'exécution.

Définition plus précise :

La complexité temporelle d'un algorithme est le nombre **d'opérations élémentaires** effectuées au cours de son exécution, dans le **pire des cas**.

- **opération élémentaire** = addition, soustraction, multiplication ou comparaison de deux bits
- **pire des cas**: le temps d'exécution peut en effet dépendre des données d'entrée.

Approximation pour ce cours :

complexité temporelle = nombre d'**instructions** lues par l'algorithme au cours de son exécution (dans le pire des cas)

Algorithme 1

Tant que $1 > 0$:
Afficher "bonjour"

Algorithme 2

entrée : L liste de nombres, n taille de la liste
sortie : m moyenne des n nombres de la liste

$m \leftarrow 0$
Pour i allant de 1 à n :
 $m \leftarrow m + L(i)$
Sortir : m/n

Algorithme 3 (version légèrement modifiée de l'algorithme «Tous différents ?»)

entrée : L liste de nombres, n taille de la liste
sortie : *oui* ou *non*

Pour i allant de 1 à $n - 1$:
 Pour j allant de $i + 1$ à n :
 Si $L(i) = L(j)$, alors :
 Sortir : *non*
Sortir : *oui*

Notation $\Theta(\cdot)$: Introduction

- En général, on évalue la complexité temporelle d'un algorithme en fonction d'un paramètre lié à la **taille des données d'entrée** (le paramètre n dans les deux exemples précédents).
- Pourquoi tant s'intéresser à cette complexité temporelle ? Voici un exemple concret:

Supposons qu'un algorithme prenne une minute pour s'exécuter avec des données d'entrée de taille $n = 1'000$. On aimerait savoir en combien de temps (au pire) s'exécutera ce même algorithme avec des données d'entrée de taille $n = 10'000$.

- Si on peut caractériser le nombre d'opérations effectuées par l'algorithme en fonction de n (comme par exemple pour l'algorithme 3 qui effectue $\frac{n(n-1)}{2} + 1$ opérations lors de son exécution, dans le pire des cas), alors on peut répondre à la question ci-dessus.

Notation $\Theta(\cdot)$: définition

- Dans de nombreuses applications, on a affaire à des données d'entrée de **grande** taille.
- Dans ce cas, on aimerait obtenir des **ordres de grandeur** plutôt que de devoir faire des calculs détaillés.

Définition

Soient $f, g : \mathbb{N} \rightarrow \mathbb{R}_+$ deux fonctions non-négatives

On dit que " $f(n)$ est un **grand theta** de $g(n)$ " et on écrit " $f(n) = \Theta(g(n))$ "

s'il existe $0 < C_1 < C_2 < \infty$ et $N \geq 1$ tels que

$$C_1 g(n) \leq f(n) \leq C_2 g(n) \quad \text{pour tout } n \geq N$$

Deux exemples :

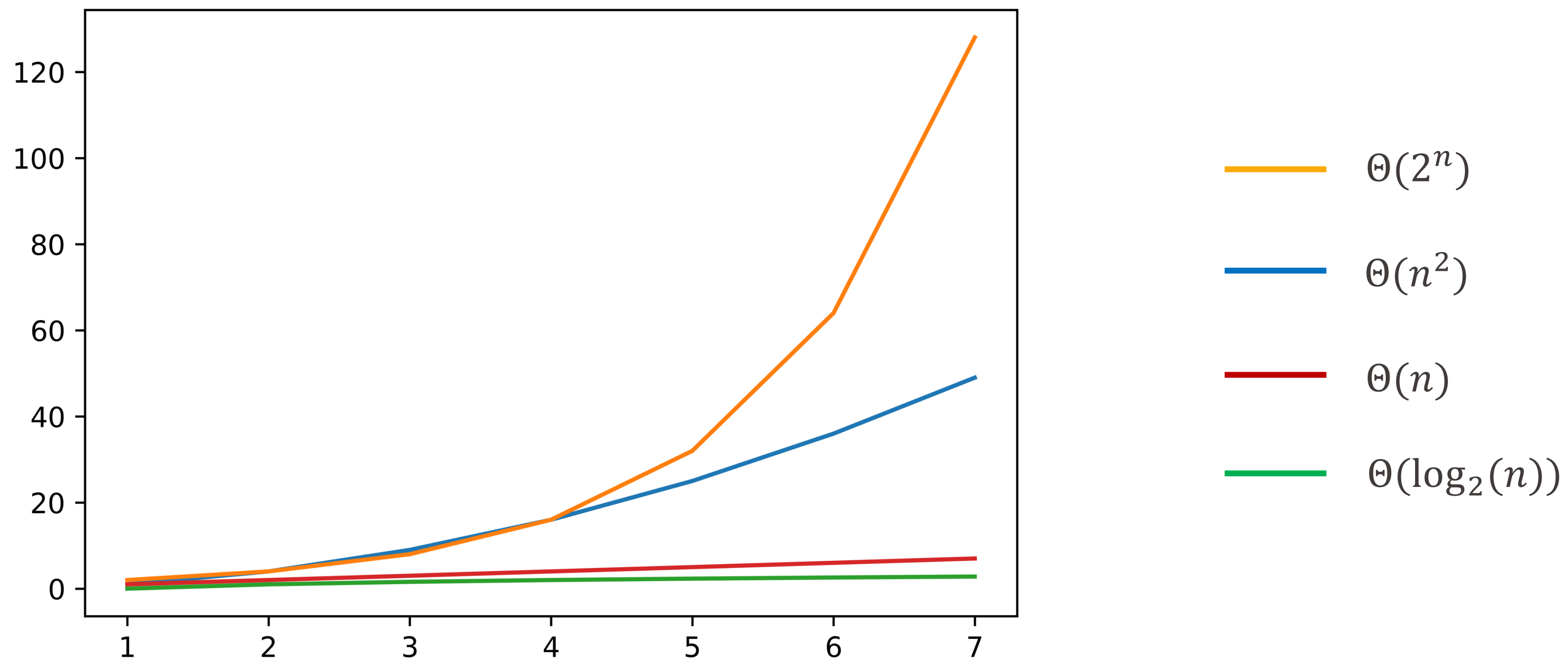
- Les fonctions $f(n) = n + 2$ et $f(n) = 3n + 3$ sont toutes deux des $\Theta(n)$ [cf. algorithme 2]
- La fonction $f(n) = \frac{n(n-1)}{2} + 1$ est un $\Theta(n^2)$ [cf. algorithme 3]

Notation $\Theta(\cdot)$: application

- Revenons à notre exemple :

Supposons qu'un algorithme prenne une minute pour s'exécuter avec des données d'entrée de taille $n = 1'000$. On aimerait savoir en combien de temps (au pire) s'exécutera ce même algorithme avec des données d'entrée de taille $n = 10'000$.

- Si la complexité temporelle de cet algorithme est un $\Theta(n)$, alors son temps d'exécution avec $n = 10'000$ en entrée vaudra (approximativement) 10 minutes.
- Si sa complexité temporelle est un $\Theta(n^2)$, alors son temps d'exécution avec $n = 10'000$ en entrée vaudra (approximativement) $10 \times 10 = 100$ minutes = 1 heure 40.



Calcul du nombre de paires d'éléments dans l'ensemble $\{1, \dots, n\}$

Pour calculer ce nombre, il existe plusieurs façons de faire :

- Utilisation de deux boucles imbriquées
→ complexité $\Theta(n^2)$
- Utilisation d'une seule boucle
→ complexité $\Theta(n)$
- Utilisation de la formule mathématique
→ complexité $\Theta(1)$

```
s ← 0
Pour i allant de 1 à n - 1 :
    Pour j allant de i + 1 à n :
        s ← s + 1
Sortir : s
```

```
s ← 0
Pour i allant de 1 à n - 1 :
    s ← s + n - i
Sortir : s
```

```
s ←  $\frac{n(n-1)}{2}$ 
Sortir : s
```

Calcul du nombre de paires d'éléments dans l'ensemble $\{1, \dots, n\}$

Pour calculer ce nombre, il existe plusieurs façons de faire :

- Utilisation de deux boucles imbriquées
→ complexité $\Theta(n^2)$
- Utilisation d'une seule boucle
→ complexité $\Theta(n)$
- Utilisation de la formule mathématique
→ complexité $\Theta(1)$

