

IIM - A2DW

---

# ALGORITHMIQUE : PENSÉE LOGIQUE

# OBJECTIFS DU COURS & MODALITÉS D'ÉVALUATIONS

- ▶ Que vous soyez à l'aise avec les concepts de variables, conditions, boucles et tableaux.
- ▶ Que vous soyez capables d'écrire vos propres algorithmes avec ces outils.
- ▶ Que vous connaissiez un ou deux algorithmes de tri.
- ▶ Évaluation : un partiel à la fin du module. Pas de question de cours, que de la pratique. Durée et contenu à déterminer suivant l'avancement.

# QU'EST-CE QU'UN ALGORITHME ?

- ▶ Une algorithmme, c'est simplement une **suite d'instructions**. Une fois exécutée correctement, elle conduit à un résultat donné.
- ▶ **Vous avez tous déjà exécuté des algorithmes**  
Ex : Jouer à un jeu de société, suivre une recette
- ▶ **Vous avez tous déjà conçu des algorithmes**  
Ex : Donner des direction, expliquer comment faire quelque chose
- ▶ **Pas besoin de maths !**

# L'ALGORITHMIQUE DANS LE WEB : TRAITER DES DONNÉES

- ▶ **Google** : Vérifier qu'un texte comprend certains mots
- ▶ **Netflix** : Optimiser le stockage et l'envoi de fichiers
- ▶ **Amazon** : Afficher une liste de produits, calculer un panier utilisateur, exécuter un paiement
- ▶ **Facebook** : Proposer de la publicité ciblée en fonction d'un profil utilisateur
- ▶ **Waze** : Calculer la distance entre deux points

# 4 INSTRUCTIONS À L'ORIGINE DE TOUT

- ▶ Attribuer une variable
- ▶ Demander ou afficher des informations à l'utilisateur
- ▶ Tester une valeur
- ▶ Faire des boucles

# LES VARIABLES

### QU'EST-CE QU'UNE VARIABLE ?

- ▶ Une variable est une **boîte** permettant de stocker une valeur.
- ▶ Ce n'est pas juste un concept. Une variable a une véritable existence dans la mémoire de la machine.
- ▶ C'est la seule utilité d'une variable : conserver une valeur pour la réutiliser plus tard.

### QUE METTRE DANS UNE VARIABLE ?

- ▶ Des nombres (1, -1, 1.0976, ...) : **int, float, ...**
- ▶ Les chaînes de caractères ("Whatever you want" ) : **string**
- ▶ Des booléens (True / False) : **boolean**
- ▶ Des dates (2020-09-23) : **date, datetime, timestamp, ...**
- ▶ ... rien du tout : **null, void...**
- ▶ Selon les langages, vous pourrez attribuer à une variable la valeur de votre choix, ou uniquement un type précis.



# UTILISER UNE VARIABLE POUR GARDER UNE VALEUR

`var A = 1`

`var A = 1 + 2`

`var A = B`

# UTILISER UNE VARIABLE POUR GARDER UNE VALEUR

var A = 1

Nom de la variable

Valeur

var A = 1 + 2

Nom de la variable

Expression

var A = B

Nom de la variable

Autre variable



**PRACTICE TIME !**

# OUTILS ALGORITHMIQUES SIMPLES

# LES OPÉRATEURS NUMÉRIQUES

- ▶ `+` : addition OU concaténation
- ▶ `-` : soustraction
- ▶ `*` : multiplication
- ▶ `/` : division
- ▶ `%` : modulo
- ▶ `Math.pow(3, 2)` : exposant
- ▶ `( )` : parenthèses classiques.

# LES OPÉRATEURS NUMÉRIQUES

- ▶ + : addition OU concaténation "toto" + "tata" → "tototata"
- ▶ - : soustraction
- ▶ \* : multiplication
- ▶ / : division
- ▶ % : modulo  $2\%2 \rightarrow 0$ ;  $18\%2 \rightarrow 0$ ;  $1\%2 \rightarrow 1$ ;  $231\%2 \rightarrow 1$ ;
- ▶ ^ : exposant  $2^2 \rightarrow 4$ ;  $3^2 \rightarrow 9$ ; .....
- ▶ ( ) : parenthèses classiques.

# INTERACTIONS UTILISATEUR : LECTURE ET ÉCRITURE

- ▶ **prompt()** : demander une valeur à l'utilisateur
- ▶ **alert()** : afficher une valeur à l'écran.

```
var titre = prompt("Quel film recherchez vous ?");
```

```
alert("Le film " + titre + " n'est pas disponible.");
```



**PRACTICE TIME !**



# LES STRUCTURES CONDITIONNELLES

# QU'EST-CE QU'UNE STRUCTURE CONDITIONNELLE ?

- Structure permettant de faire varier le comportement de votre algorithme, en fonction d'une **condition**.

```
if (condition) {  
    // instructions cas 1;  
} else {  
    // instructions cas 2;  
}
```

```
var A = prompt('saisir un chiffre');  
if (A > 0) {  
    alert('A est positif');  
} else {  
    alert('A est négatif');  
}
```

# SUCCESION DE CONDITIONS

```
if (condition1) {  
    // instructions1  
} else if (condition2) {  
    // instructions2  
} else if (condition3) {  
    // instructions3  
} else {  
    // instructions par défaut  
}
```

# SUCCESION DE CONDITIONS

```
var A = prompt("Saisir un chiffre : ");
var B = prompt("Saisir un chiffre : ");
if (A > 0) {
    alert("A est positif");
} else if (A > B) {
    alert("A est négatif");
    alert("A est supérieur à B");
} else if (B > 0) {
    alert("A est négatif");
    alert("A est inférieur à B");
    alert("B est positif");
} else { ...
```

```
} else {
    alert("A est négatif");
    alert("A est inférieur à B");
    alert("B est négatif");
}
```

# LES OPÉRATEURS DE COMPARAISON (JAVASCRIPT)

A est égal à B :  $A == B$  Ex: "3" == 3

A est égal, et de même type que B :  $A === B$  Ex: "3" === 3

A est différent de B :  $A != B$

A est strictement inférieur à B :  $A < B$

A est strictement supérieur à B :  $A > B$

A est inférieur ou égal à B :  $A <= B$

A est supérieur ou égal à B :  $A >= B$

### ! ATTENTION !

- ▶ Ne pas confondre `=` et `==`
- ▶ Ne pas confondre `Faux`, et `Invalide`.
- ▶ Comparer ce qui est comparable :
  - `3 > 1` est `Vrai`.
  - `3 < 1` est `Faux`.
  - `"3" > 2` est `Faux`. (et ce n'est pas nécessairement ce qu'on veut)
- ▶ On ne peut comparer que deux valeurs à la fois :
  - `1 < x < 3` est `Invalide`.



**PRACTICE TIME !**

# LES OPÉRATEURS BOOLÉENS



# COMBINER LES COMPARAISONS


- ▶ Les opérateurs de comparaison permettent de faire pas mal de chose, mais comment exprimer :
- ▶ **SI** ( $A < B$  est vrai) **ET SI** ( $A > 0$  est vrai)  
**Alors .....**
- ▶ **SI** ( $A > B$  est vrai) **OU** ( $A$  positif est vrai),  
**Alors .....** (ou que les deux sont vrai)
- ▶ **SI** ( $A > B$  est vrai) **OU** ( $A$  positif est vrai),  
**Alors .....** (mais pas les deux à la fois)

# LES OPÉRATEURS BOOLÉENS

- ▶ Et :  $(A < B) \ \&\& \ (A > 0)$
- ▶ Ou :  $(A < B) \ || \ (A > 0)$
- ▶ Ou exclusif :  $(A == B) \ \text{XOR} \ (A == 3)$


# LES OPÉRATEURS BOOLÉENS

► Et :  $(A < B) \ \&\& \ (A > 0)$




Condition1      Condition2

► Ou :  $(A < B) \ || \ (A > 0)$



Condition1      Condition2

► Ou exclusif :  $(A == B) \ \text{XOR} \ (A == 3)$



Condition1      Condition2

# LES OPÉRATEURS BOOLÉENS

► Et :  $(A < B) \ \&\& \ (A > 0) \longrightarrow \text{FAUX}$

$\underbrace{(A < B)}_{\text{FAUX}} \quad \underbrace{(A > 0)}_{\text{FAUX}}$

► Ou :  $(A < B) \ || \ (A > 0) \longrightarrow \text{FAUX}$

$\underbrace{(A < B)}_{\text{FAUX}} \quad \underbrace{(A > 0)}_{\text{FAUX}}$

► Ou exclusif :  $(A == B) \ \text{XOR} \ (A == 3) \longrightarrow \text{FAUX}$

$\underbrace{(A == B)}_{\text{FAUX}} \quad \underbrace{(A == 3)}_{\text{FAUX}}$

# LES OPÉRATEURS BOOLÉENS

► Et :  $(A < B) \ \&\& \ (A > 0) \longrightarrow \text{VRAI}$

$\underbrace{(A < B)}_{\text{VRAI}} \quad \underbrace{(A > 0)}_{\text{VRAI}}$

► Ou :  $(A < B) \ || \ (A > 0) \longrightarrow \text{VRAI}$

$\underbrace{(A < B)}_{\text{VRAI}} \quad \underbrace{(A > 0)}_{\text{VRAI}}$

► Ou exclusif :  $(A == B) \ \text{XOR} \ (A == 3) \longrightarrow \text{FAUX}$

$\underbrace{(A == B)}_{\text{VRAI}} \quad \underbrace{(A == 3)}_{\text{VRAI}}$

# LES OPÉRATEURS BOOLÉENS

► Et :  $(A < B) \ \&\& \ (A > 0) \longrightarrow \text{FAUX}$

$\underbrace{(A < B)}_{\text{FAUX}} \quad \underbrace{(A > 0)}_{\text{VRAI}}$

► Ou :  $(A < B) \ || \ (A > 0) \longrightarrow \text{VRAI}$


$\underbrace{(A < B)}_{\text{FAUX}} \quad \underbrace{(A > 0)}_{\text{VRAI}}$

► Ou exclusif :  $(A == B) \ \text{XOR} \ (A == 3) \longrightarrow \text{VRAI}$

$\underbrace{(A == B)}_{\text{FAUX}} \quad \underbrace{(A == 3)}_{\text{VRAI}}$

TABLES DE VÉRITÉ

Inverse la valeur de A

A	B	 ! A	A && B	A    B	A XOR B
VRAI	VRAI	FAUX	VRAI	VRAI	FAUX
FAUX	VRAI	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	VRAI	FAUX	FAUX	FAUX



**PRACTICE TIME !**



# LES BOUCLES

# BOUCLE WHILE

- ▶ *Tant que* la condition est vraie, la boucle continue.
- ▶ Utilisé quand **on ne sait pas** combien de fois on va devoir répéter les instructions.

```
while (condition1) {  
    instructions...  
    instructions...  
}
```

### BOUCLE WHILE

- ▶ *Tant que* la condition est vraie, la boucle continue.
- ▶ Utilisé quand **on ne sait pas** combien de fois on va devoir répéter les instructions.

```
var kmRestants = prompt('quelle distance ? (en km) ');  
var vitesse = prompt('vitesse ? (en km)');  
while (kmRestants > 0) {  
    kmRestants = kmRestants - vitesse;  
}  
Alert('vous êtes arrivé !');
```

### BOUCLE FOR

- ▶ *Pour* une condition donnée, la boucle continue de tourner. A chaque nouvelle boucle, le compteur est incrémenté.
- ▶ Utilisé quand **on sait** précisément combien de boucles on souhaite effectuer

```
For (var compteur = X; condition; increment) {  
    instructions ...  
    instructions ...  
}
```

### BOUCLE FOR

- ▶ *Pour* une condition donnée, la boucle continue de tourner. A chaque nouvelle boucle, le compteur est incrémenté.
- ▶ Utilisé quand **on sait** précisément combien de boucles on souhaite effectuer

```
var lettre_a_deviner = 'z';  
For (var i = 0; i < 10; i++) {  
    var lettre = prompt('saisissez une lettre : ');  
    if (lettre === lettre_a_deviner) {  
        alert('félicitation ! Vous avez trouvé');  
    }  
    alert (9 - i + " tentatives restantes." )  
}
```

### QUELQUES SPÉCIFICITÉS

- ▶ On peut imbriquer des boucles, dans des boucles.
- ▶ Attention aux boucles infinies !
- ▶ Attention aux boucles jamais exécutées !



**PRACTICE TIME !**

# LES TABLEAUX



# QU'EST-CE QU'UN TABLEAU ?

- ▶ C'est une structure de données permettant de conserver plusieurs valeurs.

```
Var A = 3;
```

```
Var monTableau = [  
    'Du texte', A, 5/3, true, prompt('saisissez un chiffre');  
];
```

```
alert(monTableau[0]); // Du texte  
alert(monTableau[1]); // 3  
alert(monTableau[2]); // 5  
alert(monTableau[3]); // true  
alert(monTableau[4]); // undefined
```

# PARCOURIR UN TABLEAU

```
var tableau = ['toto', 'tata', 'tutu'];  
  
// tableau.length = 3;  
  
for (var i = 0; i < tableau.length; i++) {  
    // boucle 1 : 'toto'  
    // boucle 2 : 'tata'  
    // boucle 3 : 'tutu'  
    // boucle 4 : undefined  
    alert(tableau[i]);  
}
```

# QUELQUES PARTICULARITÉS DES TABLEAUX

- ▶ L'indice d'un tableau comment toujours à 0.
- ▶ Il est impossible d'accéder à un indice qui n'existe pas. Cela donnera différents résultats suivant le langage utilisé, mais jamais rien de bon. (en javascript, on aura une valeur **undefined**).
- ▶ Les chaînes de caractères sont en fait des tableaux.

# TABLEAUX MULTIDIMENSIONNELS

- On peut placer des tableaux dans des tableaux :

```
var tableau_multidimension = [  
    ['toto', 'tata', 'tutu'],  
    [1, 2, 3, 4],  
    [true, false]  
];  
alert(tableau_multidimension[0][1]); // 'tata'
```



**PRACTICE TIME !**

# ALGORITHMES DE TRI

# LE TRI PAR SÉLECTION

- ▶ On trie le tableau, valeur par valeur :
- ▶ on fait une première passe et on met la plus petite valeur en position 0.
- ▶ Sur la seconde passe, on commence à l'indice 1, et on cherche la plus petite valeur, qu'on va mettre à l'indice 1.
- ▶ Sur la troisième passe, on commence à l'indice 2, et on cherche la plus petite valeur, qu'on va mettre à l'indice 2.
- ▶ Etc...

# LE TRI PAR SÉLECTION : STEP-BY-STEP

↓  
▶ [10, 3, 7, 4, 2, 9] // plus petit = 10 (indice 0)



# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [10, 3, 7, 4, 2, 9] // plus petit = 3 (indice 1)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  10,  3, 7, 4, 2, 9 // plus petit = 3 (indice 1)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [10, 3, 7,  4, 2, 9] // plus petit = 3 (indice 1)


# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [10, 3, 7, 4,  2, 9] // plus petit = 2 (indice 4)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [10, 3, 7, 4, 2, 9]  // plus petit = 2 (indice 4)

### LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [10, 3, 7, 4, 2, 9] // plus petit = 2 (indice 4)
- ▶ On inverse la plus petite valeur avec la valeur à l'indice 0.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [2, 3, 7, 4, 10, 9]

- ▶ La première valeur est maintenant correcte. On recommence à partir de l'indice 1.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

↓  
▶ [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)



# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶  [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)



# LE TRI PAR SÉLECTION : STEP-BY-STEP


▶ [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)




# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)
- ▶ On place notre plus petite valeur à l'indice 1.

### LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [2, 3, 7, 4, 10, 9] // Plus petit = 3 (indice 1)
- ▶ La deuxième valeur du tableau est maintenant correcte.  
On recommence à partir de l'indice 2.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

↓  
▶ [2, 3, 7, 4, 10, 9] // Plus petit = 7 (indice 2)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 7, 4, 10, 9]    // Plus petit = 4 (indice 3)





# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 7, 4, 10, 9]    // Plus petit = 4 (indice 3)




# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 7, 4, 10, 9]    // Plus petit = 4 (indice 3)



### LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [2, 3, 7, 4, 10, 9] // Plus petit = 4 (indice 3)
- ▶ On place notre plus petite valeur à l'indice 2.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 4, 7, 10, 9]



- ▶ La valeur à l'indice 2 est maintenant correcte, on recommence à partir de l'indice 3.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

↓  
▶ [2, 3, 4, 7, 10, 9] // Plus petit = 7 (indice 4)

# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 4, 7, 10, 9]    // Plus petit = 7 (indice 4)




# LE TRI PAR SÉLECTION : STEP-BY-STEP

▶ [2, 3, 4, 7, 10, 9]    // Plus petit = 7 (indice 4)



# LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶ [2, 3, 4, 7, 10, 9]    // Plus petit = 7 (indice 4)  

- ▶ On place notre plus petite valeur à l'indice 3



# LE TRI PAR SÉLECTION : STEP-BY-STEP


- ↓
- ▶ [2, 3, 4, 7, 10, 9] // Plus petit = 10 (indice 4)
  - ▶ On recommence à l'indice 4.

# LE TRI PAR SÉLECTION : STEP-BY-STEP


▶ [2, 3, 4, 7, 10, 9]    // Plus petit = 9 (indice 5)



### LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [2, 3, 4, 7, 10, 9] // Plus petit = 9 (indice 5)
- ▶ On place notre plus petite valeur à l'indice 4.

# LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶  [2, 3, 4, 7, 9, 10] // on recommence au début
- ▶ On est maintenant arrivé à la fin du tableau, c'est que tout le tableau est trié !

# LE TRI PAR SÉLECTION : STEP-BY-STEP

- ▶ [2, 3, 4, 7, 9, 10]
- ▶ On est maintenant arrivé à la fin du tableau, c'est que tout le tableau est trié !



**PRACTICE TIME !**

# LE TRI À BULLES

- ▶ On part d'un constat : dans un tableau trié par ordre croissant, chaque valeur est plus petite que la valeur qui la suit.
- ▶ Principe du tri à bulle : on parcourt le tableau. On compare chaque valeur à la valeur qui la suit.
- ▶ Si les deux valeurs ne sont pas dans le bon ordre, on les permute. Puis on passe à la valeur suivante.
- ▶ Petit à petit, les valeurs les plus grandes "remontent" vers le bout du tableau (comme des bulles).
- ▶ Quand on arrive à la fin du tableau, on recommence le processus du début, et ainsi de suite jusqu'à ce que tout le tableau soit trié.

# LE TRI À BULLES : STEP-BY-STEP




► [10, 3, 7, 4, 2, 9]



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [**10**, **3**, 7, 4, 2, 9] // 10 > 3 ?

# LE TRI À BULLES : STEP-BY-STEP

▶  [10, 3, 7, 4, 2, 9] // 10 > 3 ? Oui → on permute !

# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [**3**, **10**, 7, 4, 2, 9] //  $10 > 3$  ? Oui → on permute !

# LE TRI À BULLES : STEP-BY-STEP



► [3, 10, 7, 4, 2, 9] // On passe à l'indice suivant


# LE TRI À BULLES : STEP-BY-STEP



▶ [3, **10**, **7**, 4, 2, 9] // 10 > 7 ?

# LE TRI À BULLES : STEP-BY-STEP

▶ [3, **10**, **7**, 4, 2, 9]    // 10 > 7 ? Oui → On permute !



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, **7**, **10**, 4, 2, 9] // 10 > 7 ? Oui → On permute !

# LE TRI À BULLES : STEP-BY-STEP



► [3, 7, 10, 4, 2, 9] // On passe à l'indice suivant



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, **10**, **4**, 2, 9]    // 10 > 4 ?

# LE TRI À BULLES : STEP-BY-STEP

▶ [3, 7, **10**, **4**, 2, 9]    // 10 > 4 ? Oui → on permute !



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, **4**, **10**, 2, 9]    // 10 > 4 ? Oui → on permute !

# LE TRI À BULLES : STEP-BY-STEP

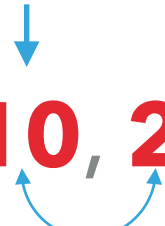
↓  
▶ [3, 7, 4, 10, 2, 9] // On passe à l'indice suivant

# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, 4, **10**, **2**, 9]    //  $10 > 2$  ?

# LE TRI À BULLES : STEP-BY-STEP

▶ [3, 7, 4, **10**, **2**, 9]    // 10 > 2 ? Oui → on permute !



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, 4, **2**, **10**, 9]    // 10 > 2 ? Oui → on permute !

# LE TRI À BULLES : STEP-BY-STEP



► [3, 7, 4, 2, 10, 9] // On passe à l'indice suivant




# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, 4, 2, **10**, **9**] // 10 > 9 ?

# LE TRI À BULLES : STEP-BY-STEP


▶ [3, 7, 4, 2, **10**, **9**]    // 10 > 9 ? Oui → on permute !



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 7, 4, 2, **9**, **10**] // 10 > 9 ? Oui → on permute !

# LE TRI À BULLES : STEP-BY-STEP

- ▶  [3, 7, 4, 2, 9, 10] // on passe à l'indice suivant
- ▶ On arrive au bout du tableau.
- ▶ Une fois arrivé au bout la première fois, la plus grande valeur sera "remontée" à droite du tableau.
- ▶ C'est l'inverse du tri par sélection : plutôt que de commencer à ranger les plus petites valeurs à gauche du tableau, on commence par ranger les plus grandes à droite.

# LE TRI À BULLES : STEP-BY-STEP



- ▶ [3, 7, 4, 2, 9, **10**] // On revient au début du tableau
- ▶ On est arrivé au bout une première fois, mais notre tableau n'est toujours pas trié.
- ▶ Une fois la première boucle terminée, on recommence du début du tableau
- ▶ On va recommencer le processus, encore et encore, jusqu'à faire un passage complet sans permutation : moment où tout le tableau sera trié.

# LE TRI À BULLES : STEP-BY-STEP



► [3, 7, 4, 2, 9, **10**] // 3 > 7 ? Non → On ne touche rien.

# LE TRI À BULLES : STEP-BY-STEP



► [3, 7, 4, 2, 9, **10**] // On passe à l'indice suivant

# LE TRI À BULLES : STEP-BY-STEP



► [3, 7, 4, 2, 9, 10] // 7 > 4 ? Oui → On permute !



# LE TRI À BULLES : STEP-BY-STEP

▶  [3, 4, 7, 2, 9, 10] // 7 > 4 ? Oui → On permute !

# LE TRI À BULLES : STEP-BY-STEP



► [3, 4, 7, 2, 9, **10**] // On passe à l'indice suivant


# LE TRI À BULLES : STEP-BY-STEP



► [3, 4, 7, 2, 9, 10] // 7 > 2 ? Oui → On permute

# LE TRI À BULLES : STEP-BY-STEP

▶ [3, 4, 2, 7, 9, 10]    // 7 > 2 ? Oui → On permute



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 4, 2, 7, 9, **10**] // Indice suivant

# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [3, 4, 2, 7, 9, 10] // 7 > 9 ? Non. Au suivant.

# LE TRI À BULLES : STEP-BY-STEP



▶ [3, 4, 2, 7, 9, **10**]

▶ On arrive à l'avant-dernier indice.

On sait qu'on a déjà fait un passage sur le tableau.

La valeur suivante est donc forcément plus grande, on peut s'arrêter ici.

# LE TRI À BULLES : STEP-BY-STEP



► [3, 4, 2, 7, 9, 10] // On recommence du début




# LE TRI À BULLES : STEP-BY-STEP



► [3, 4, 2, 7, 9, 10]

# LE TRI À BULLES : STEP-BY-STEP

▶ [3, 2, 4, 7, 9, 10]



# LE TRI À BULLES : STEP-BY-STEP



► [3, 2, 4, 7, 9, 10]

# LE TRI À BULLES : STEP-BY-STEP



► [3, 2, 4, 7, 9, 10]

# LE TRI À BULLES : STEP-BY-STEP



► [3, 2, 4, 7, 9, 10] // On recommence du début

# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [2, 3, 4, 7, 9, 10] // On recommence du début



# LE TRI À BULLES : STEP-BY-STEP



► [2, 3, 4, 7, 9, 10] // On recommence du début

# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [2, 3, 4, 7, 9, 10]



# LE TRI À BULLES : STEP-BY-STEP

↓  
▶ [2, 3, 4, 7, 9, 10]    // On recommence du début

# LE TRI À BULLES : STEP-BY-STEP



▶ [2, 3, 4, 7, 9, 10]

- ▶ On est arrivé au bout sans faire de permutation :  
c'est que tout le tableau est trié !

# LE TRI À BULLES : STEP-BY-STEP



▶ [2, 3, 4, 7, 9, 10]

- ▶ On est arrivé au bout sans faire de permutation :  
c'est que tout le tableau est trié !

# RECHERCHE DICHOTOMIQUE

### QU'EST-CE QUE C'EST ?

- ▶ C'est une méthode pour effectuer une recherche dans un tableau **TRIÉ**, sans forcément parcourir tout ce tableau.
- ▶ Permet de gagner beaucoup de temps. Plus le jeu de données est grand, plus le gain de temps sera important.

### EXEMPLE : RECHERCHE D'UN MOT DANS LE DICTIONNAIRE

- ▶ Prenons le cas d'un dictionnaire comprenant 40.000 mots. C'est un dictionnaire, donc les mots sont triés.
- ▶ Si on parcourt notre dictionnaire en lisant les mots un par un, et que le mot qu'on cherche est tout à la fin (ou pire : s'il n'existe pas), on aura dû lire 40.000 mots avant de trouver celui qu'on cherche.
- ▶ Avec une recherche dichotomique, nous aurons lu au maximum... 16 mots.

## STEP-BY-STEP

- indiceMin                      indiceMax  
↓                                      ↓
- ▶ [a, b, c, f, l, o, s, u, **v**, z]
  - ▶ Prenons un exemple avec un tableau de 10 entrées.
  - ▶ Disons que la valeur recherché est « v ».
  - ▶ La première étape va consister à se placer exactement au milieu de notre tableau.
  - ▶ On calcul le milieu de la manière suivante :  
$$\text{milieu} = (\text{indiceMin} + \text{indiceMax}) / 2$$

## STEP-BY-STEP

- indiceMin                      indiceMax  
↓                                      ↓
- ▶ [a, b, c, f, l, o, s, u, **v**, z]
  - ▶ IndiceMin = 0. IndiceMax = 9.  
Milieu = 4,5.
  - ▶ On peut prendre 4 ou 5, cela n'a pas d'importance. Il faut juste faire attention si on a arrondi au supérieur, de continuer à arrondi au supérieur pour la suite, et si on a arrondi à l'inférieur, de continuer à arrondir à l'inférieur.



## STEP-BY-STEP

indiceMin  
↓

milieu  
↓

indiceMax  
↓

► [a, b, c, f, l, **o**, s, u, **v**, z]      // milieu = 5

► Milieu == 5, on se place donc à l'indice 5.

## STEP-BY-STEP

indiceMin



milieu



indiceMax



► [a, b, c, f, l, **o**, s, u, **v**, z]      // 'o' == 'v' ? Non.

► On compare la valeur au milieu, à la valeur recherchée

## STEP-BY-STEP

indiceMin



milieu



indiceMax



► [a, b, c, f, l, o, s, u, v, z]      // 'o' < 'v' ? Oui.

- On regarde si on se trouve avant ou après la valeur recherchée

### STEP-BY-STEP

indiceMin                      milieu                      indiceMax  
↓                                      ↓                                      ↓

► [a, b, c, f, l, **o**, s, u, **v**, z]                      // 6 < 9 ? Oui.

- La valeur du milieu est plus petite que la valeur recherchée, on place donc notre indiceMin au milieu.
- Si la valeur du milieu était plus grande, on aurait déplacé indiceMax

### STEP-BY-STEP

indiceMin      indiceMax  
↓                    ↓

- ▶ [a, b, c, f, l, o, s, u, **v**, z]      // indiceMin = milieu
- ▶ La valeur du milieu est plus petite que la valeur recherchée, on place donc notre indiceMin au milieu.
- ▶ Si la valeur du milieu était plus grande, on aurait déplacé indiceMax.

## STEP-BY-STEP

milieu  
indiceMin      indiceMax  
↓                ↓                ↓

► [a, b, c, f, l, o, s, **u**, **v**, z]      // milieu = 7

► On recalcule notre milieu :  $(9 + 5)/2 = 7$

### STEP-BY-STEP

indiceMin   milieu   indiceMax  
↓   ↓   ↓

► [a, b, c, f, l, o, s, **u**, **v**, z]      // 'u' == 'v' ? Non.

► La valeur du milieu est-elle la valeur recherchée ?

### STEP-BY-STEP

milieu  
indiceMin      indiceMax  
↓                ↓                ↓

► [a, b, c, f, l, o, s, **u**, **v**, z]      // 'u' < 'v' ? Oui.

► La valeur du milieu est-elle plus petite que la valeur recherchée ?

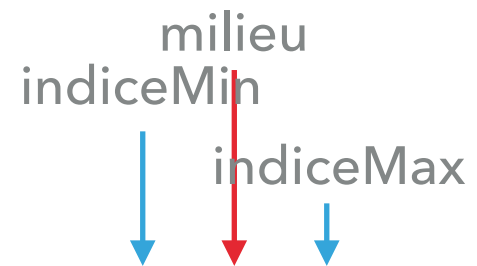


### STEP-BY-STEP

indiceMin  
↓  
indiceMax  
↓

- ▶ [a, b, c, f, l, o, s, u, **v**, z] // indiceMin = milieu
- ▶ C'est le cas, donc on déplace notre indiceMin au milieu.

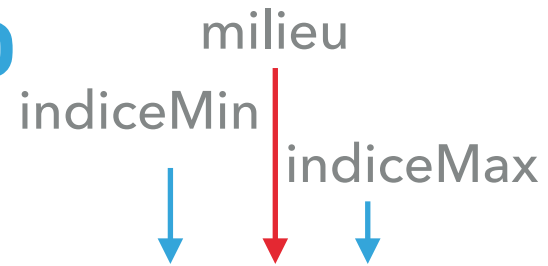
## STEP-BY-STEP



► [a, b, c, f, l, o, s, u, **v**, z] // milieu = 8

► On recalcule notre milieu :  $(7+9)/2 = 8$

### STEP-BY-STEP



► [a, b, c, f, l, o, s, u, **v**, z]      // 'v' === 'v' ? Oui.

► La valeur du milieu est-elle la valeur recherchée ? Oui. On a trouvé notre valeur !

# RÉSUMÉ DE L'ALGORITHME DE RECHERCHE DICHOTOMIQUE

1. Calculer le milieu en fonction d'indiceMin et indiceMax.
2. Vérifier si la valeur du milieu est la valeur recherchée
3. Si ce n'est pas le cas, vérifier si la valeur du milieu est plus petite ou plus grande que la valeur recherchée.
4. Si elle est plus petite, on déplace indiceMin au milieu.
5. Si elle est plus grande, on déplace indiceMax au milieu.
6. Recommencer en 1.



**PRACTICE TIME !**

# COMPLEXITÉ ALGORITHMIQUE

# EVALUER LA PERFORMANCE D'UN ALGORITHME

- ▶ Certains algorithmes vont être plus rapides que d'autres pour effectuer les mêmes tâches.
- ▶ Les algorithmes peuvent prendre plus de temps, ou de mémoire, suivant le nombre données à traiter, et donc être plus ou moins performants.
- ▶ Cette performance, c'est ce qu'on appelle la **complexité algorithmique**.

# DEUX TYPES DE COMPLEXITÉ : SPATIALE, ET TEMPORELLE

- ▶ La complexité **temporelle** va évaluer le temps que votre algorithme va prendre pour terminer son exécution.
- ▶ La complexité **spatiale** correspond à l'espace que va prendre l'algorithme dans la mémoire de votre ordinateur. Combien de variables supplémentaires va-t-il créer pendant son exécution ?



### 4 TYPES DE COMPLEXITÉ :

- ▶  $O(1)$  : complexité en temps constant
- ▶  $O(n)$  : complexité linéaire
- ▶  $O(10^n)$  : complexité exponentielle
- ▶  $O(n \log n)$  : complexité logarithmique

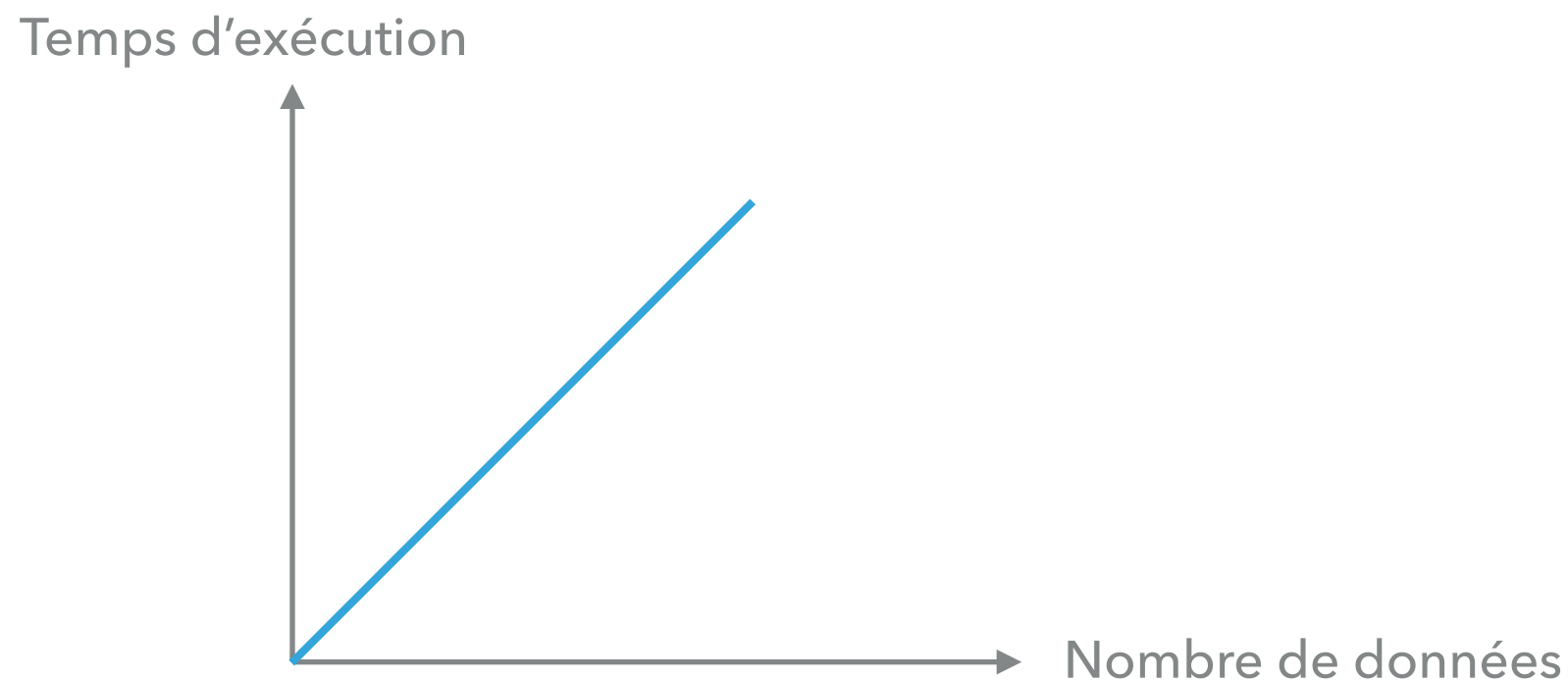
# COMPLEXITÉ EN TEMPS CONSTANT

- ▶ Peu importe combien de données on a, l'algorithme prend toujours autant de temps.
- ▶ Exemple : afficher directement une valeur précise, dont on connaît l'index.



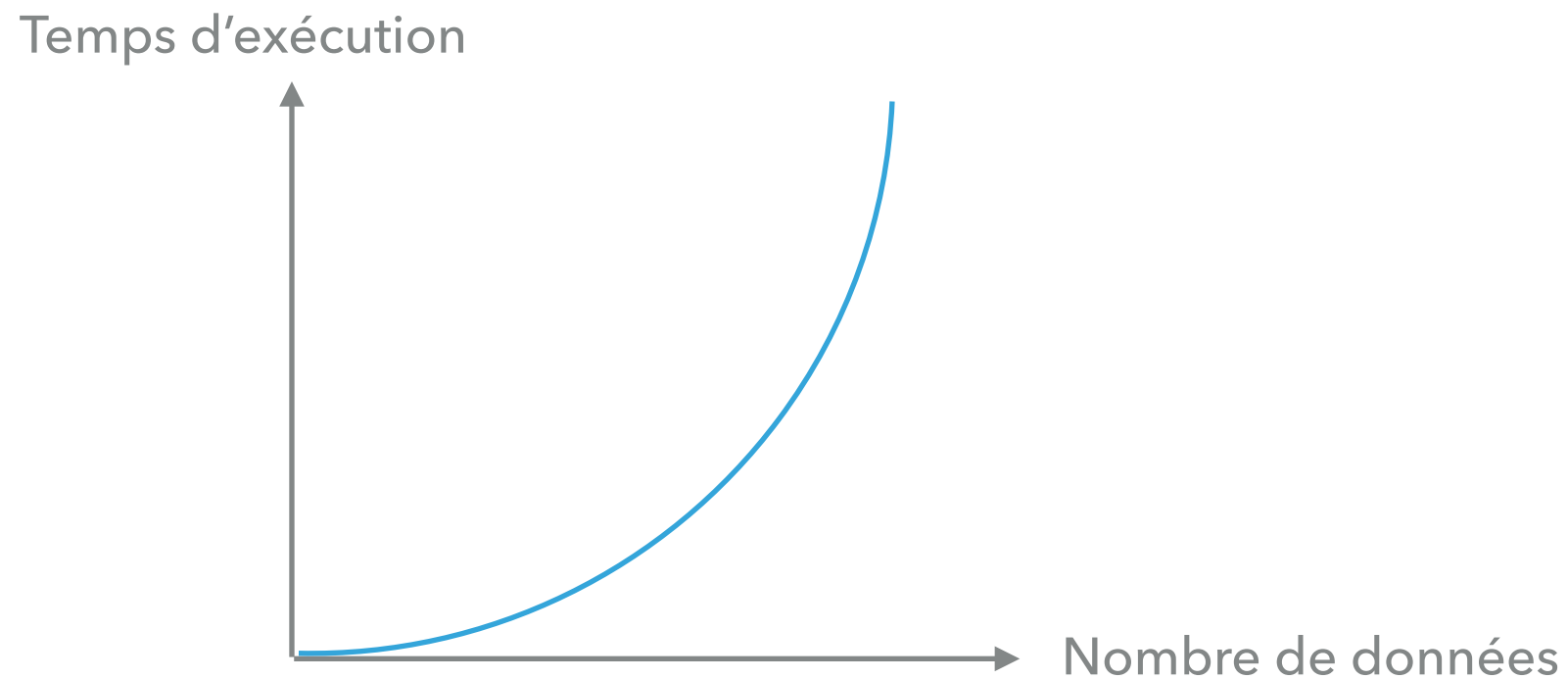
# COMPLEXITÉ LINÉAIRE

- ▶ La complexité augmente proportionnellement au nombre de données.
- ▶ Si l'algo prend 1sec pour traiter 1 donnée, il prendra 2sec pour 2 données, 3 sec pour 3 données, etc....



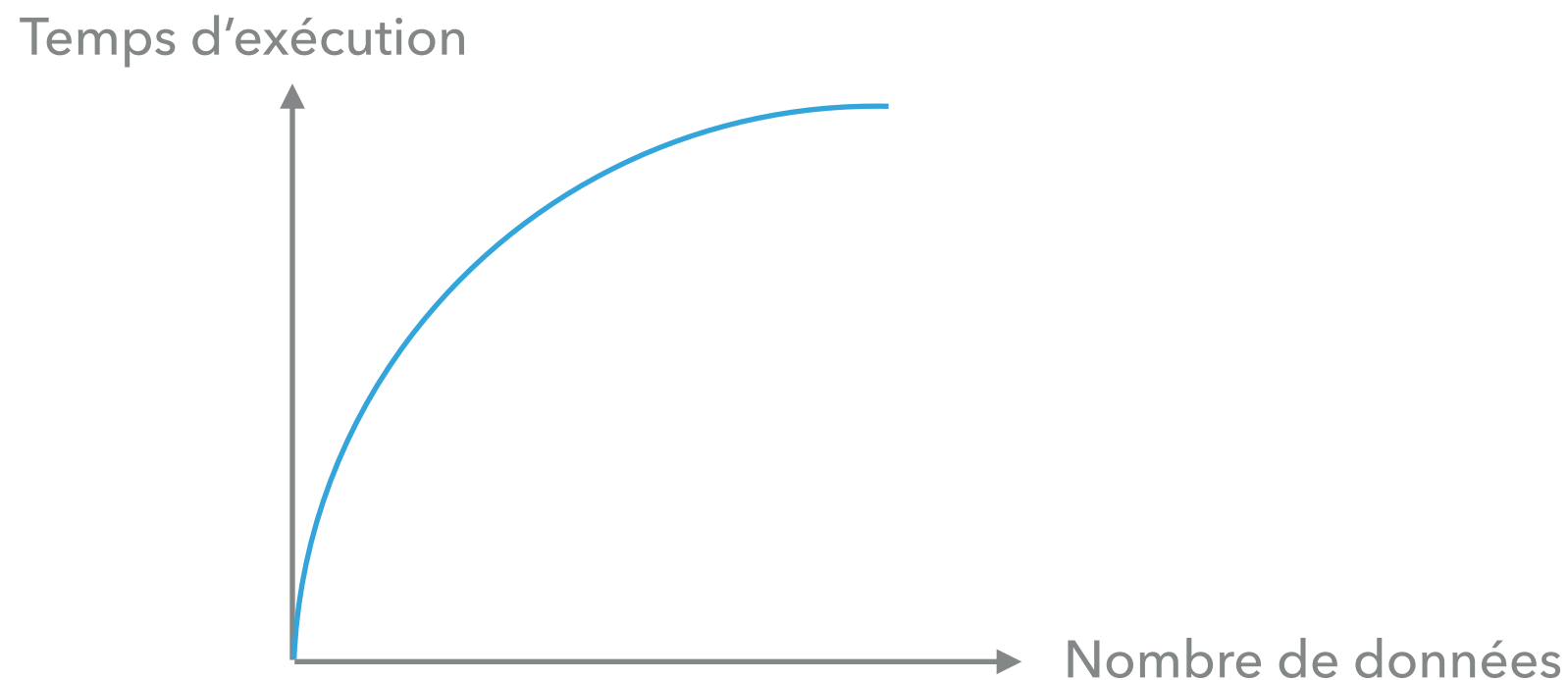
# COMPLEXITÉ EXPONENTIELLE

- ▶ La complexité augmente exponentiellement.
- ▶ Ex : tri par sélection.



# COMPLEXITÉ LOGARITHMIQUE

- ▶ La complexité augmente de manière logarithmique.
- ▶ Ex : recherche dichotomique.



**LEETCODE.COM**



- ▶ <https://linkedin.com/in/clement-trumpff/>
- ▶ Feedback → <https://Bit.ly/2F8z1Ri>
- ▶ Merci à vous et bon courage pour la suite !