

DUT SRC – IUT de Marne-la-Vallée  
18/01/2012  
INF220 - Algorithmique

# ***Cours 1***

## ***Réversivité et tris***

# Organisation pratique

- **Contact**

- Courriel : philippe.gambette@gmail.com  
(INF220 doit apparaître dans le sujet du courriel).
- Avant ou après le cours.
- Possibilité de poser des questions, de demander des exercices supplémentaires d'entraînement.

- **Notes et devoirs**

Chez soi :

- Projet Monopoly : exercices à trous à remplir au plus tard la veille des cours/TP/TD pour vous inciter à travailler **régulièrement** et corriger vos erreurs. **Annonce sur IRIS 6 jours avant.**

Pendant les cours :

- Note globale de TP à la fin du semestre, prenant en compte votre avancée à chaque séance.
- Devoir final le 18 juin.

# Sources

---

- Cours de Jean-François Berdjugin à l'IUT de Grenoble  
<http://berdjugin.com/enseignements/inf/inf220/>
- Cours de Xavier Heurtebise à l'IUT de Provence  
<http://x.heurtebise.free.fr>
- *Le livre de Java premier langage*, d'A. Tasso
- <http://xkcd.com>, <http://xkcd.free.fr>

# Plan des cours du semestre

---

- Récursivité
- Tris
- Notions de complexité
- Listes
- Arbres

# Plan des cours du semestre

- Récursivité
  - Tris
  - Notions de complexité
  - Listes
  - Arbres
- 
- cours 1
- cours 2

# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

# Récurtivité

---

2 façons de le voir :

- Une mise en abyme
- Une baguette magique algorithmique

# Récurtivité

- Une mise en abyme

## La "minute xkcd" - Jeu de rôle sur table



<http://xkcd.com/244>

<http://xkcd.free.fr?id=244>

# Récurtivité

- Une mise en abyme



Photo Ethan Clements  
<http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html>



<http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme>

# Récurtivité

- Une mise en abyme



Photo Ethan Clements  
<http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html>



<http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme>

Comment dessiner ces images ?

# Récurtivité

- Une mise en abyme



Photo Ethan Clements  
<http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html>



<http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme>

Comment dessiner ces images ?

L'image contient une plus petite version d'elle-même.

# Récurtivité

- Une mise en abyme



Photo Ethan Clements  
<http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html>



<http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme>

Comment dessiner ces images ?

Pour dessiner la grande image, j'utilise le dessin de l'image en plus petit.

# Récurtivité

---

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

$$\text{factorielle}(1) = 1$$

$$\text{factorielle}(2) = 1 \times 2 = 2$$

$$\text{factorielle}(3) = 1 \times 2 \times 3 = 6$$

$$\text{factorielle}(4) = 1 \times 2 \times 3 \times 4 = 24$$

...

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

$$\text{factorielle}(1) = 1$$

$$\text{factorielle}(2) = 1 \times 2 = 2$$

$$\text{factorielle}(3) = 1 \times 2 \times 3 = 6$$

$$\text{factorielle}(4) = 1 \times 2 \times 3 \times 4 = 24$$

...

**Initialisation :**

**Formule d'hérédité :**

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

$$\text{factorielle}(1) = 1$$

$$\text{factorielle}(2) = 1 \times 2 = 2$$

$$\text{factorielle}(3) = 1 \times 2 \times 3 = 6$$

$$\text{factorielle}(4) = 1 \times 2 \times 3 \times 4 = 24$$

...

**Initialisation :**

$$\text{factorielle}(1) = 1$$

**Formule d'hérédité :**

$$\text{factorielle}(n) = \dots \text{factorielle}(n-1) \dots$$

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

$$\text{factorielle}(1) = 1$$

$$\text{factorielle}(2) = 1 \times 2 = 2$$

$$\text{factorielle}(3) = 1 \times 2 \times 3 = 6$$

$$\text{factorielle}(4) = 1 \times 2 \times 3 \times 4 = 24$$

...

**Initialisation :**

$$\text{factorielle}(1) = 1$$

**Formule d'hérédité :**

$$\text{factorielle}(n) = \text{factorielle}(n-1) \times n$$

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :

$$\text{factorielle}(1) = 1$$

x2

$$\text{factorielle}(2) = 1 \times 2 = 2$$

$$\text{factorielle}(3) = 1 \times 2 \times 3 = 6$$

$$\text{factorielle}(4) = 1 \times 2 \times 3 \times 4 = 24$$

...

**Initialisation :**

$$\text{factorielle}(1) = 1$$

**Formule d'hérédité :**

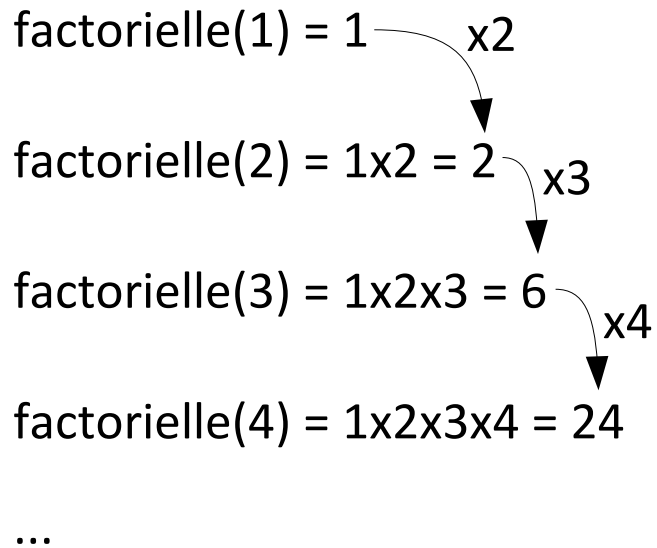
$$\text{factorielle}(n) = \text{factorielle}(n-1) \times n$$

# Récurtivité

- Une baguette magique algorithmique

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Exemple des factorielles :



**Initialisation :**

$$factorielle(1) = 1$$

**Formule d'hérédité :**

$$factorielle(n) = factorielle(n-1) \times n$$

# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je **multiplie par  $n$**

Algorithme **factorielle**

Entrées :

Type de sortie :

Variable :

Début

Fin

# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je multiplie par  $n$

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

    si  $n=1$  alors :

*resultat*  $\leftarrow 1$

    sinon :

*resultat*  $\leftarrow n \times$  **factorielle**( $n-1$ )

    renvoyer *resultat*

Fin

# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je multiplie par  $n$

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

si  $n=1$  alors :

*resultat*  $\leftarrow 1$

sinon :

*resultat*  $\leftarrow n \times$   $\underbrace{\text{factorielle}(n-1)}_{\text{entier}}$

renvoyer *resultat*

entier

Fin

# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1

- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je multiplie par  $n$

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

si  $n=1$  alors :

*resultat*  $\leftarrow 1$

sinon :

*resultat*  $\leftarrow n \times$  **factorielle**( $n-1$ )

renvoyer *resultat*

Fin

*version non réursive :*

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

*resultat*  $\leftarrow 1$

Pour  $i$  de 2 à  $n$  faire :

*resultat*  $\leftarrow i \times$  *resultat*

FinPour

renvoyer *resultat*

Fin

# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1
- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je **multiplie par  $n$**

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

si  $n=1$  alors :

$resultat \leftarrow 1$

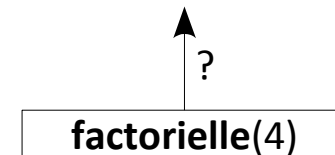
sinon :

$resultat \leftarrow n \times \mathbf{factorielle}(n-1)$

renvoyer *resultat*

Fin

Trace de **factorielle(4)** :



# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1
- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je **multiplie par  $n$**

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

    si  $n=1$  alors :

*resultat*  $\leftarrow$  1

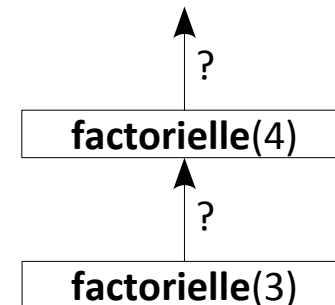
    sinon :

*resultat*  $\leftarrow$   $n \times$  **factorielle**( $n-1$ )

    renvoyer *resultat*

Fin

Trace de **factorielle**(4) :



# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1
- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je **multiplie par  $n$**

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

    si  $n=1$  alors :

*resultat*  $\leftarrow$  1

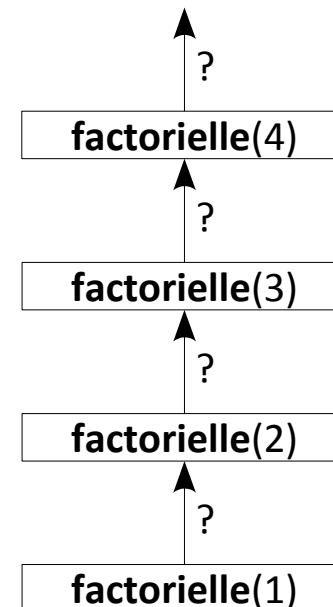
    sinon :

*resultat*  $\leftarrow$   $n \times$  **factorielle**( $n-1$ )

    renvoyer *resultat*

Fin

Trace de **factorielle**(4) :



# Récurtivité

- Une baguette magique algorithmique

Exemple des factorielles :

- la première factorielle est 1
- pour passer de factorielle( $n-1$ ) à factorielle( $n$ ), je **multiplie par  $n$**

Algorithme **factorielle**

Entrées : entier  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

si  $n=1$  alors :

$resultat \leftarrow 1$

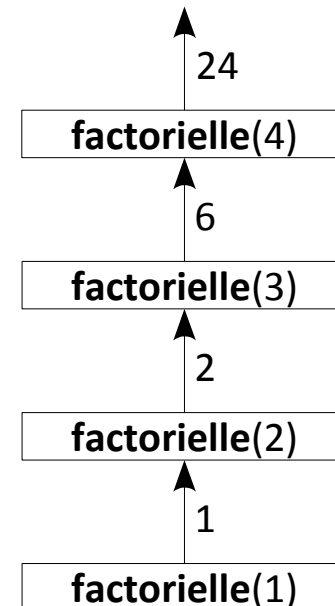
sinon :

$resultat \leftarrow n \times \mathbf{factorielle}(n-1)$

renvoyer *resultat*

Fin

Trace de **factorielle(4)** :



# Récurtivité

Même concept que la **preuve par récurrence**

*La “minute mathématique”*

**Théorème** : Je sais monter une échelle

# Récurtivité

Même concept que la **preuve par récurrence**

*La “minute mathématique”*

**Théorème** : Je sais monter une échelle

**Démonstration** :

**Initialisation** : je sais monter depuis le sol jusqu'au premier barreau.

**Hérédité** : si je sais monter jusqu'au  $(n-1)$ -ième barreau, je saurai monter jusqu'au  $n$ -ième barreau.

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

Initialisation :

Formule d'hérédité :

Exemple :

$$a \times 0 = 0$$

$$a \times 1 = a$$

$$a \times 2 = 2a$$

$$a \times 3 = 3a$$

?

?

?

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

Initialisation :

$$a \times 0 =$$

Formule d'hérédité :

$$a \times n = \dots a \times (n-1) \dots$$

Exemple :

$$a \times 0 = 0$$

$$a \times 1 = a$$

$$a \times 2 = 2a$$

$$a \times 3 = 3a$$

?

?

?

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

Initialisation :

$a \times 0 =$

Formule d'hérédité :

$a \times n = \dots a \times (n-1) \dots$

Exemple :

$$\begin{array}{l} 3 \times 0 = 0 \\ 3 \times 1 = 3 \\ 3 \times 2 = 6 \\ 3 \times 3 = 9 \\ 3 \times 4 = 12 \end{array} \begin{array}{l} \curvearrowright +3 \\ \curvearrowright +3 \\ \curvearrowright +3 \end{array}$$

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

Initialisation :

$$a \times 0 = 0$$

Formule d'hérédité :

$$a \times n = a \times (n-1) + a$$

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

**Initialisation :**

$$a \times 0 = 0$$

**Formule d'hérédité :**

$$a \times n = a \times (n-1) + a$$

Algorithme **produit**

Entrées :

Type de sortie :

Variable :

Début

Fin

# Un autre algorithme récursif

## La multiplication

Si je sais construire le  $n$ -ième objet à partir du  $(n-1)$ -ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

Pour calculer  $a \times n$  :

**Initialisation :**

$$a \times 0 = 0$$

**Formule d'hérédité :**

$$a \times n = a \times (n-1) + a$$

Algorithme **produit**

Entrées : entiers  $a$  et  $n$

Type de sortie : entier

Variable : entier *resultat*

Début

si  $n=0$  alors :

*resultat*  $\leftarrow 0$

sinon :

*resultat*  $\leftarrow$  **produit**( $a, n-1$ ) +  $a$

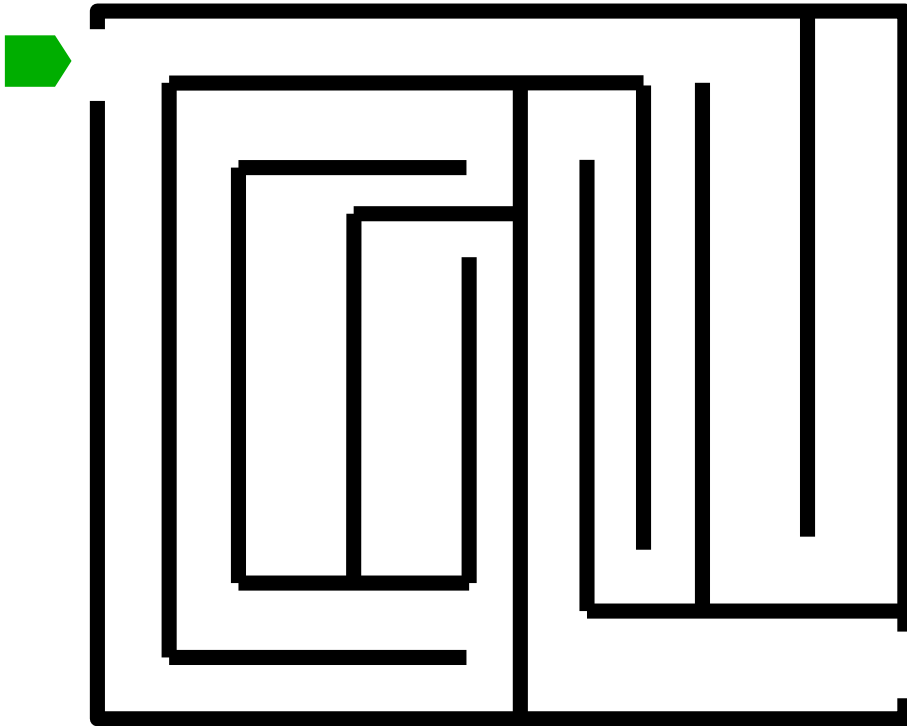
renvoyer *resultat*

Fin

# Récurtivité

Le labyrinthe et les robots tueurs :

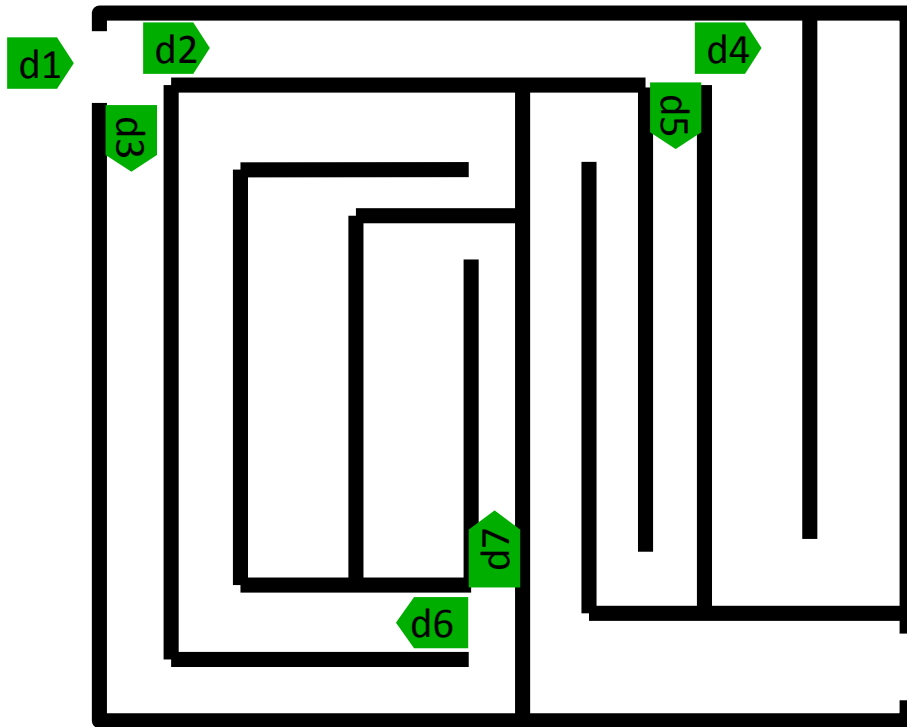
10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.



# Récurtivité

Le labyrinthe et les robots tueurs :

10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.



Algorithme **TrouveBouton**

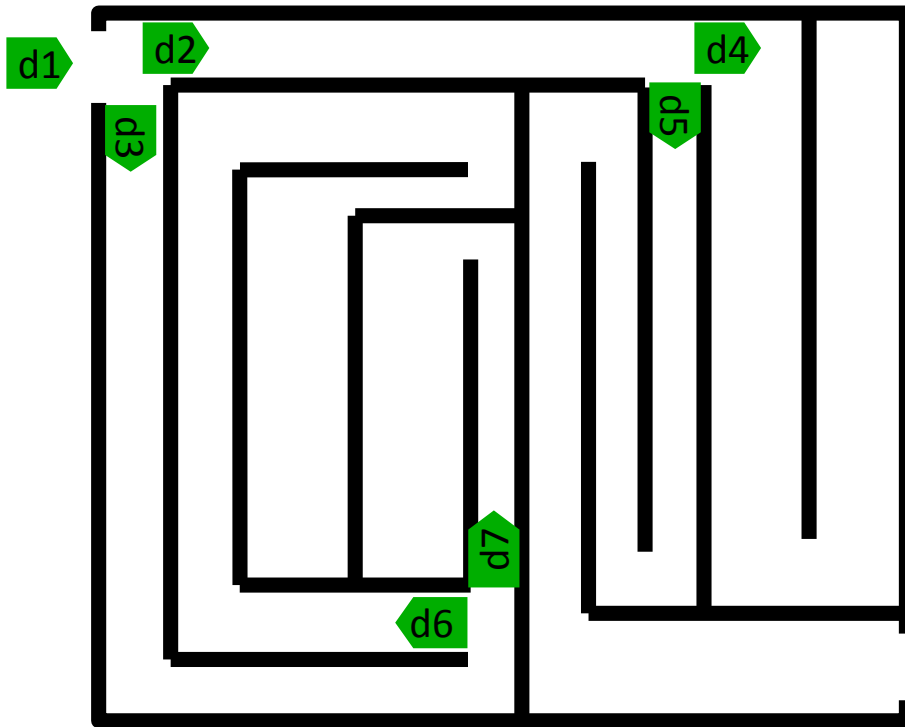
Entrées : entier *nombrePersonnes*,  
flottant *tempsRestant*,  
chaîne de caractères *direction*  
Type de sortie : booléen



# Récurtivité

Le labyrinthe et les robots tueurs :

10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.



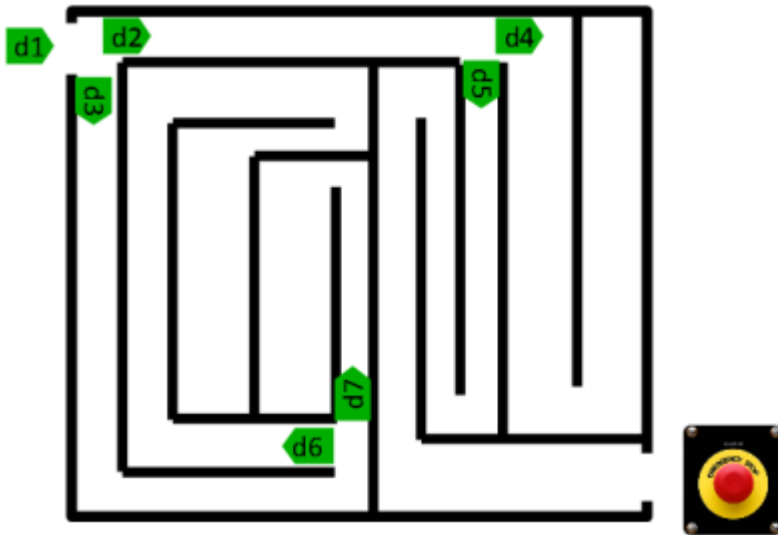
Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*  
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.



# Récurtivité



Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*

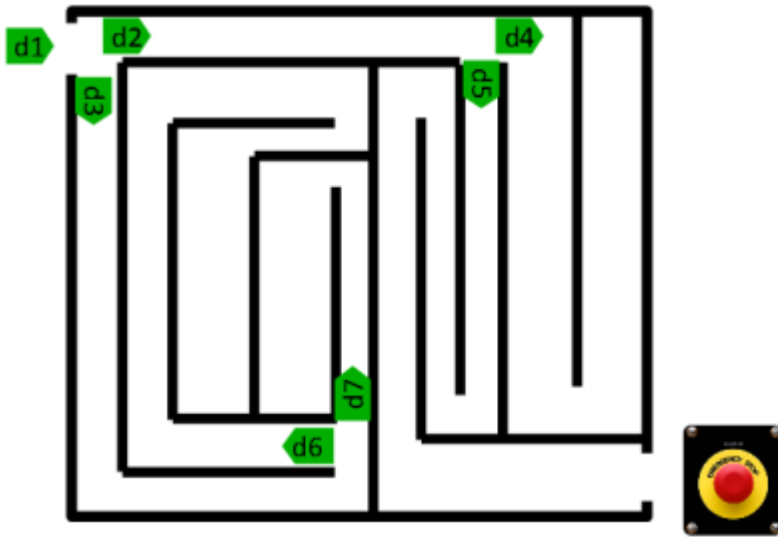
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?

Il faut faire la trace de **trouveBouton**(10,5,"d1").

# Récurivité



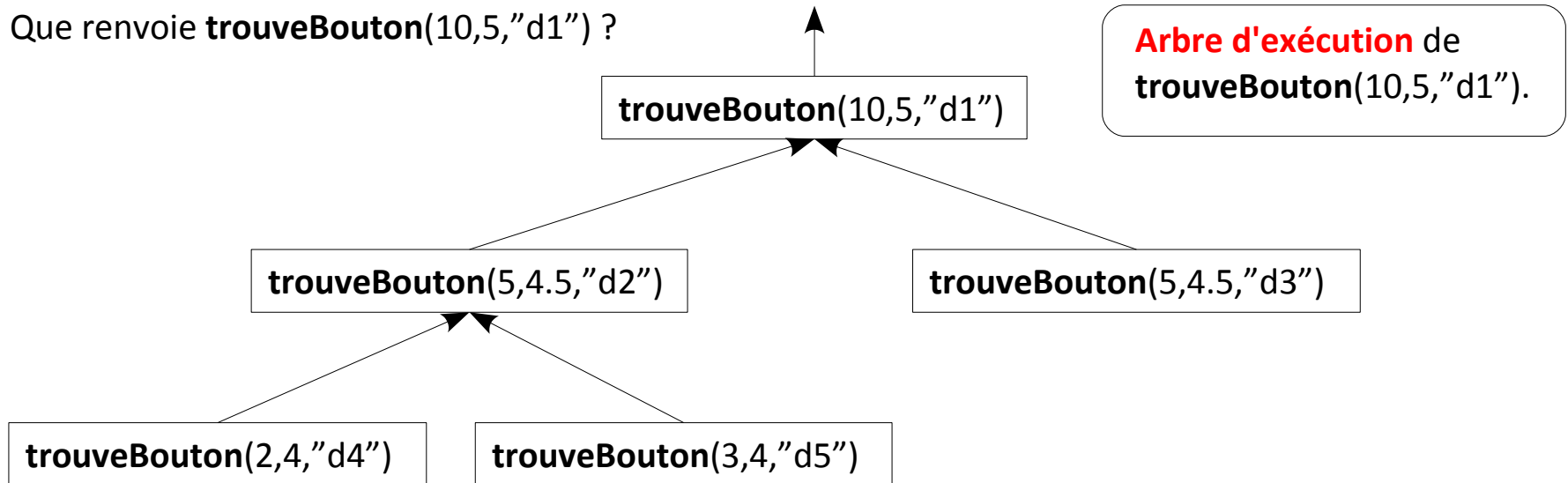
Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*

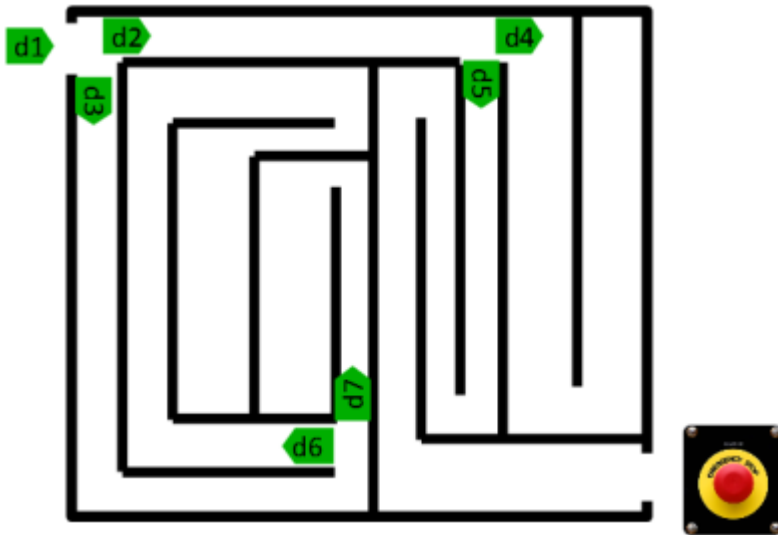
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



# Récurivité



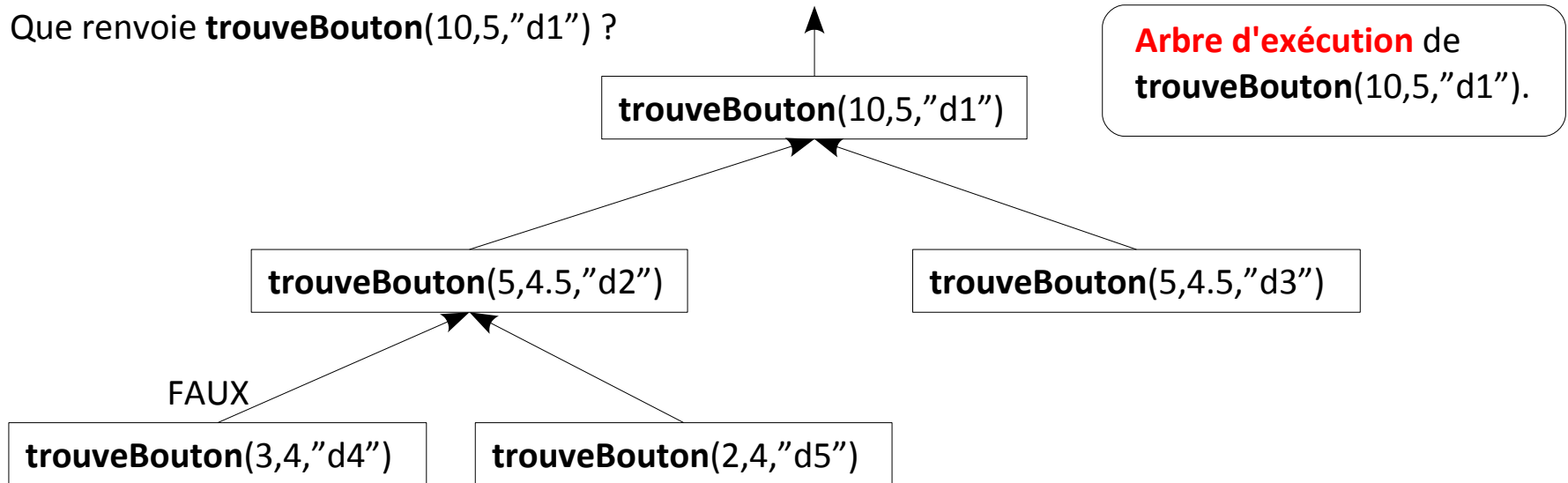
Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*

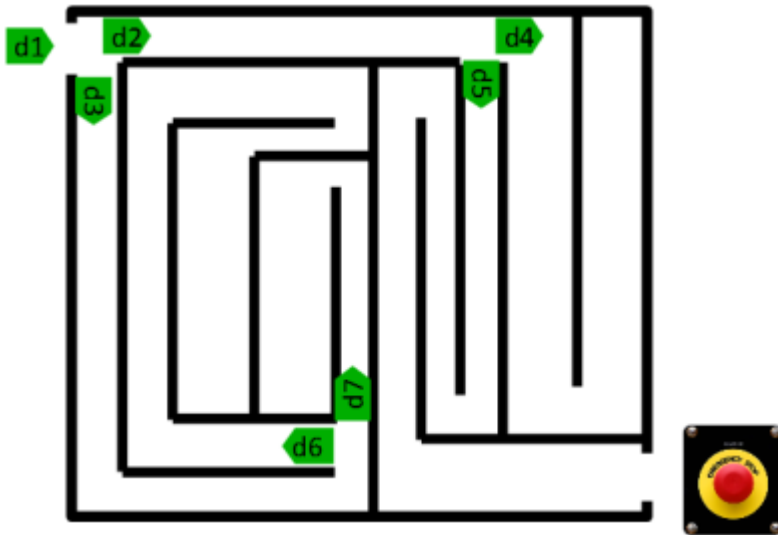
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



# Récurivité



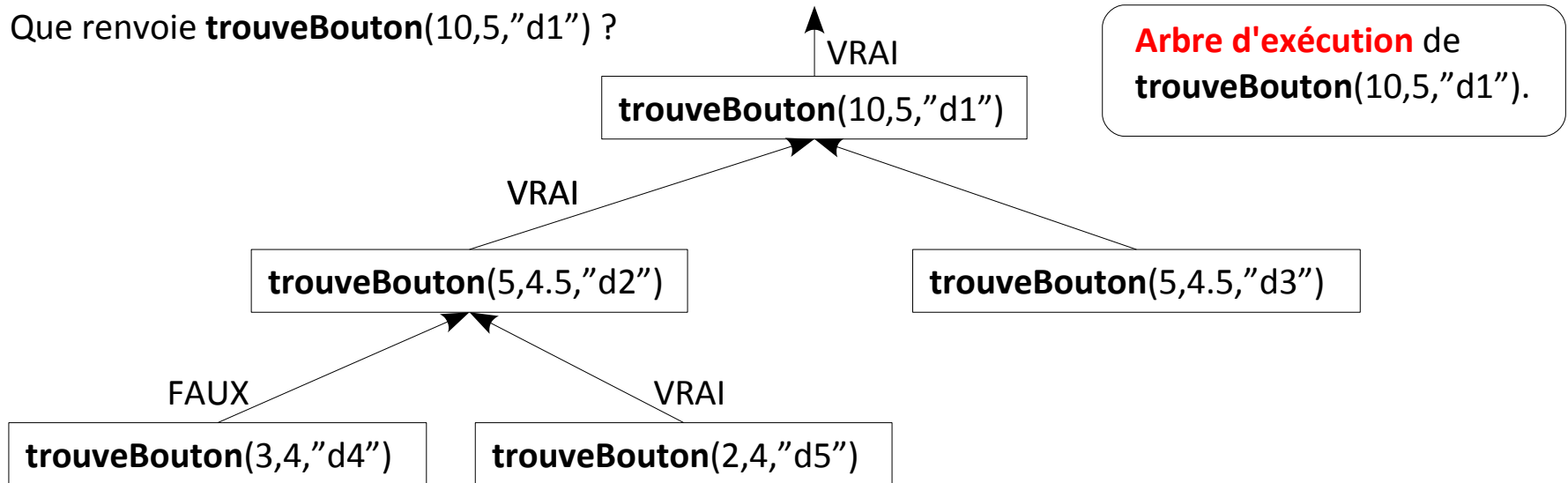
Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*

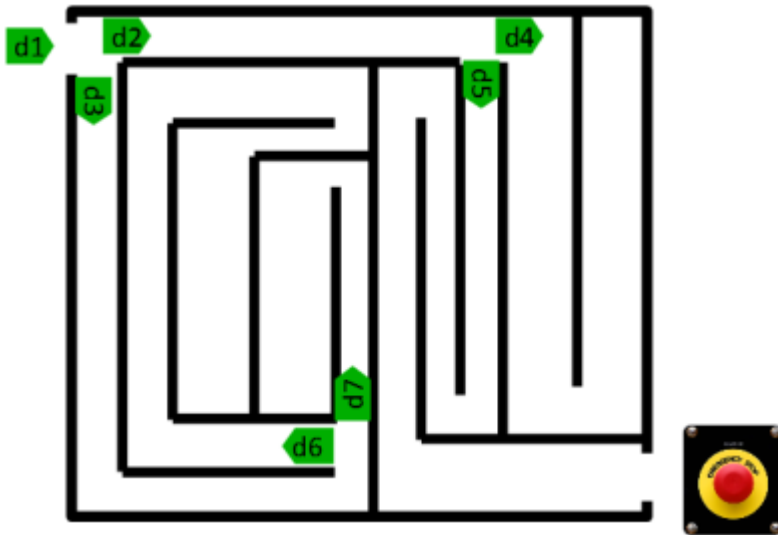
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



# Récurivité



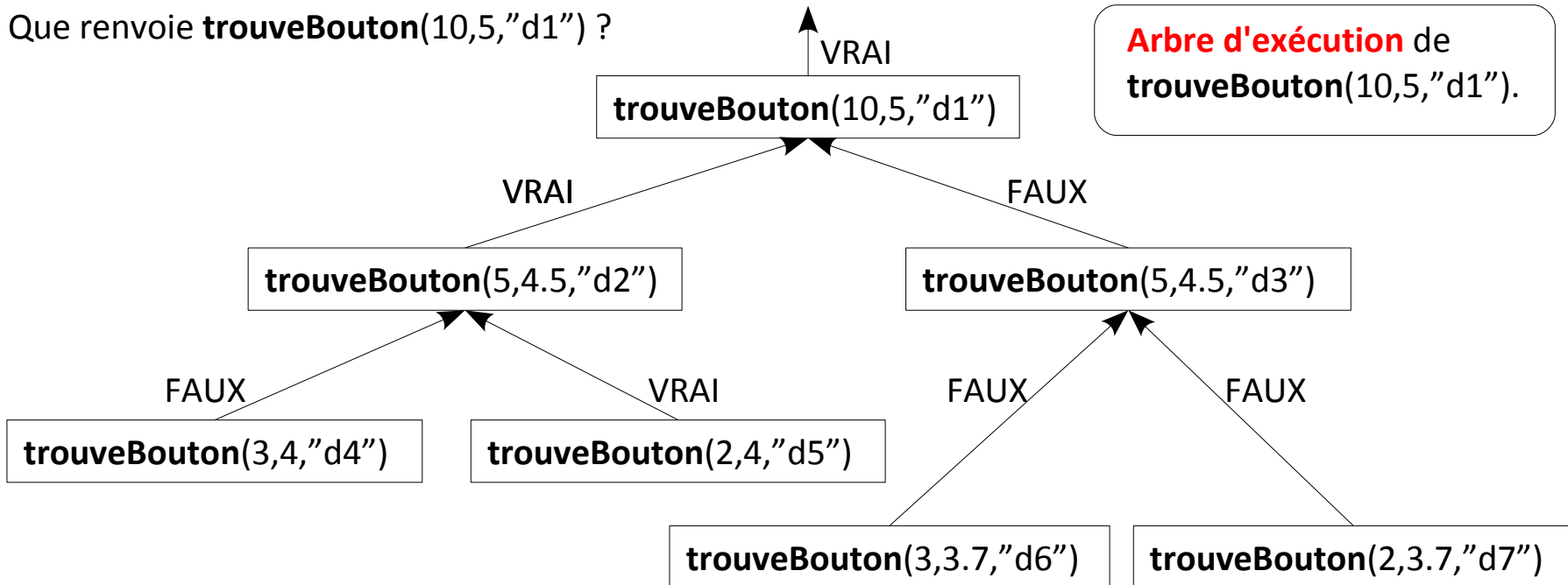
Algorithme **trouveBouton**

Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*

Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?



# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- **Les tris**
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

# Principe d'un algorithme de tri

---

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments : quel type ?

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments : quel type ?

Type pour stocker plusieurs éléments ?

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments : quel type ?

Type pour stocker plusieurs éléments ?

→ un tableau !

→ autres structures de données au prochain cours.

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments	Ordre	Tous comparables deux à deux ?
tableau d'entiers	plus petit	
tableau de chaînes de caractères	lexicographique (du dictionnaire)	
tableau de cartes à jouer	$2 \leq 3 \leq \dots \leq 10 \leq$ $V \leq D \leq R \leq 1$	
tableau de cartes à jouer	$\clubsuit \leq \diamondsuit \leq \heartsuit \leq \spadesuit$ puis $2 \leq 3 \leq \dots \leq 10 \leq$ $V \leq D \leq R \leq 1$	

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments	Ordre	Tous comparables deux à deux ?
tableau d'entiers	plus petit	oui : $1 \leq 2, 2 \leq 3, 1 \leq 3...$
tableau de chaînes de caractères	lexicographique (du dictionnaire)	oui : $a \leq b, \text{char} \leq \text{charles}, \text{char} \leq \text{chat}...$
tableau de cartes à jouer	$2 \leq 3 \leq \dots \leq 10 \leq V \leq D \leq R \leq 1$	non : $5 \clubsuit$ incomparable avec $5 \spadesuit$
tableau de cartes à jouer	$\clubsuit \leq \diamondsuit \leq \heartsuit \leq \spadesuit$ puis $2 \leq 3 \leq \dots \leq 10 \leq V \leq D \leq R \leq 1$	oui : $A \clubsuit \leq 5 \spadesuit, 5 \spadesuit \leq 6 \spadesuit, \dots$

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Exemple avec un tableau d'entiers :

5	3	1	8	5	2	9
---	---	---	---	---	---	---



1	2	3	5	5	8	9
---	---	---	---	---	---	---

En anglais : trié = sorted  
trier = to sort

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Exemple avec un tableau d'entiers :

5	3	1	8	5	2	9
---	---	---	---	---	---	---



1	2	3	5	5	8	9
---	---	---	---	---	---	---

En anglais : trié = sorted  
trier = to sort

Algorithme **inferieurOuEgal**

Entrées : ?

Type de sortie : ?

*Exemple* : **inferieurOuEgal(5,8)** renvoie ...

# Principe d'un algorithme de tri

**Entrée :**

**Ensemble** d'éléments **tous comparables deux à deux** par un ordre  $\leq$

**Sortie :**

Éléments **triés selon cet ordre**, du plus petit au plus grand

Exemple avec un tableau d'entiers :

5	3	1	8	5	2	9
---	---	---	---	---	---	---



1	2	3	5	5	8	9
---	---	---	---	---	---	---

En anglais : trié = sorted  
trier = to sort

Algorithme **inferieurOuEgal**

Entrées : deux entiers  $a$  et  $b$

Type de sortie : booléen

*Exemple* : **inferieurOuEgal**(5,8) renvoie VRAI

# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- **Le tri par sélection**
- Le tri à bulles
- Complexité des tris

# Tri par sélection (ou tri par extraction)

---

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

# Tri par sélection (ou tri par extraction)

*La "minute culturelle"*

Select-sort with Gypsy folk dance

AlgoRythmics  S'abonner 6 vidéos

facebook.com/AlgoRythmics Intercultural Computer Science Education



0:12 / 7:07 360p

 J'aime  + Ajouter à  Partager  98 228 

Ajoutée par AlgoRythmics le 2 avril 2011

Created at Sapientia University, Tirgu Mures (Marosvásárhely), Romania. 443 aiment, 9 n'aiment pas

<http://www.youtube.com/watch?v=Ns4TPTC8whw>

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :

5	3	<b>1</b>	8	5	2	9
---	---	----------	---	---	---	---

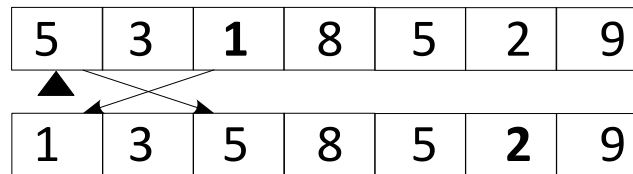


$i$  vaut 1

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

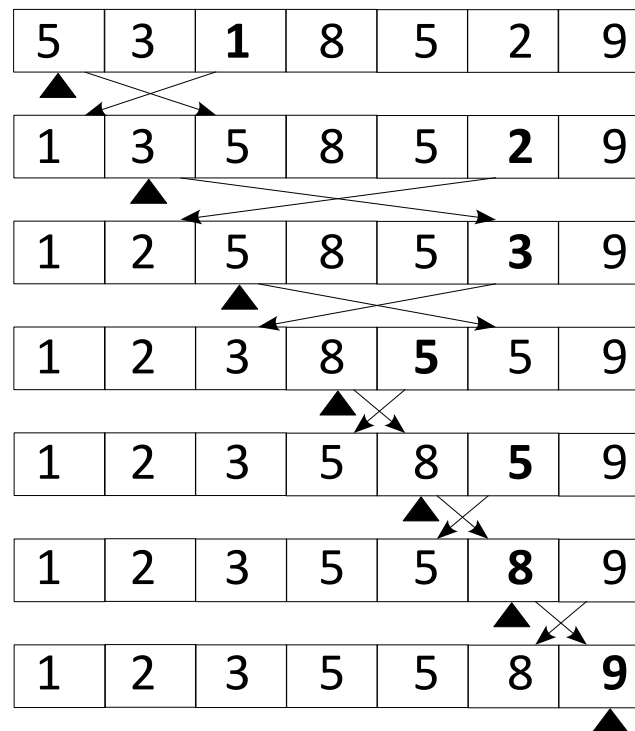
Exemple avec un tableau d'entiers :



# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :



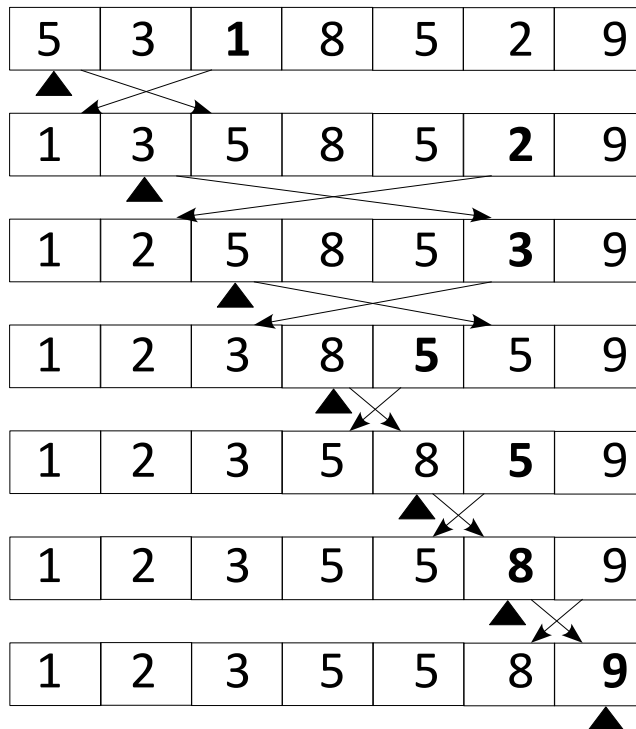
**Tri en place** :

on arrive à trier le tableau sans avoir besoin de créer un nouveau tableau.

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :



Algorithme **positionMinimum**

Entrée : tableau d'entiers *tab*, entier *debut*

Type de sortie : entier

Variables : entiers *position*, *i* et *min*

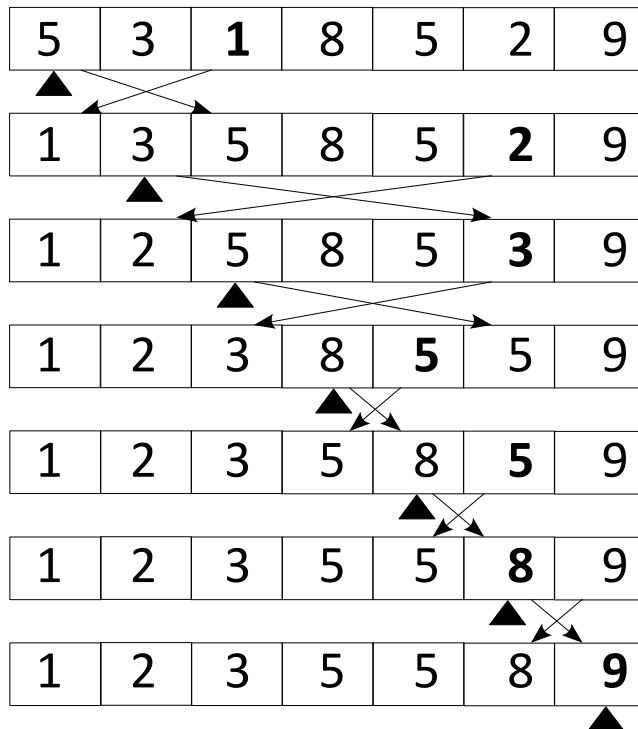
Début

Fin

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :



Algorithme **positionMinimum**

Entrée : tableau d'entiers  $tab$ , entier  $debut$

Type de sortie : entier

Variables : entiers  $position$ ,  $i$  et  $min$

Début

$position \leftarrow debut$

$min \leftarrow \mathbf{Case}(tab,debut)$

$i \leftarrow debut$

Tant que  $i \leq \mathbf{Longueur}(tab)$  faire :

si  $\mathbf{Case}(tab,i) < min$  alors :

$position \leftarrow i$

$min \leftarrow \mathbf{Case}(tab,i)$

$i \leftarrow i+1$

Fin Tant que

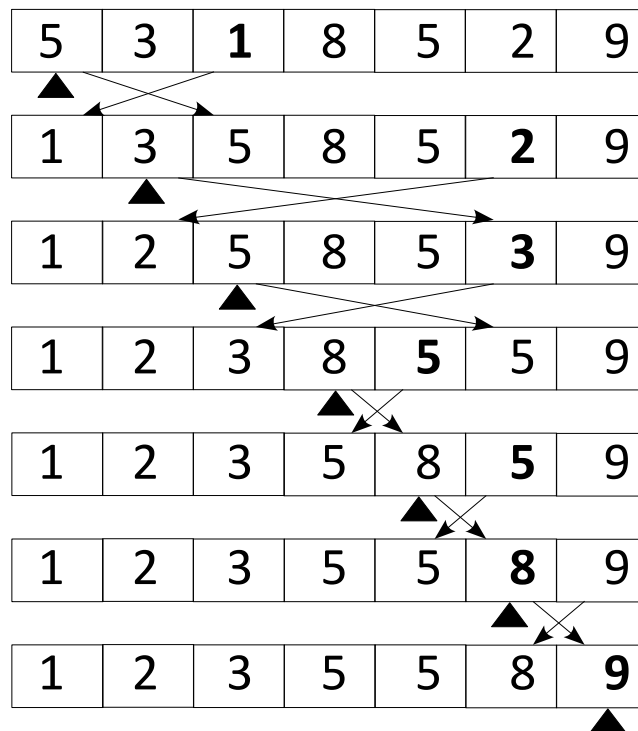
renvoyer  $position$

Fin

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :



Algorithme **triSelection**

Entrée : tableau d'entiers *tab*

Type de sortie : tableau d'entiers

Variables : entiers *i*, *temp*, *posMin*

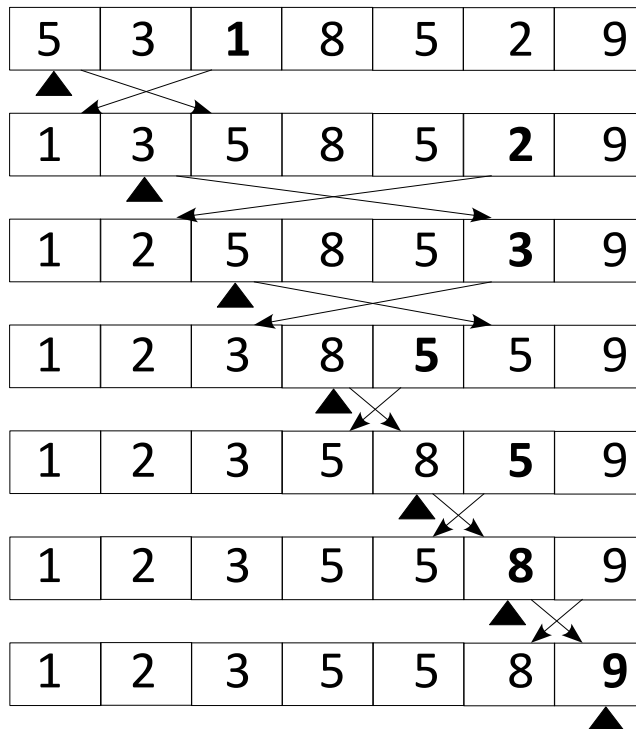
Début

Fin

# Tri par sélection (ou tri par extraction)

**Idée** : à la  $i$ -ième étape, on prend le plus petit élément à partir de la  $i$ -ième case (comprise) et on l'échange avec l'élément de la  $i$ -ième case.

Exemple avec un tableau d'entiers :



Algorithme **triSelection**

Entrée : tableau d'entiers  $tab$

Type de sortie : tableau d'entiers

Variables : entiers  $i$ ,  $temp$ ,  $posMin$

Début

$i \leftarrow 1$

Tant que  $i < \text{Longueur}(tab)$  faire :

$posMin \leftarrow \text{positionMinimum}(tab, i)$

//  $posMin$  peut être égal à  $i$

$temp \leftarrow \text{Case}(tab, i)$

$\text{Case}(tab, i) \leftarrow \text{Case}(tab, posMin)$

$\text{Case}(tab, posMin) \leftarrow temp$

$i \leftarrow i + 1$

Fin Tant que

renvoyer  $tab$

Fin

# Plan du cours 1 – Récursivité et tris

---


- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- **Le tri à bulles**
- Complexité des tris

# Tri à bulles

**Idée** : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la  $i$ -ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :

5	3	1	8	5	2	9
---	---	---	---	---	---	---



# Tri à bulles

**Idée** : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la  $i$ -ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :

*Étape 1 :*

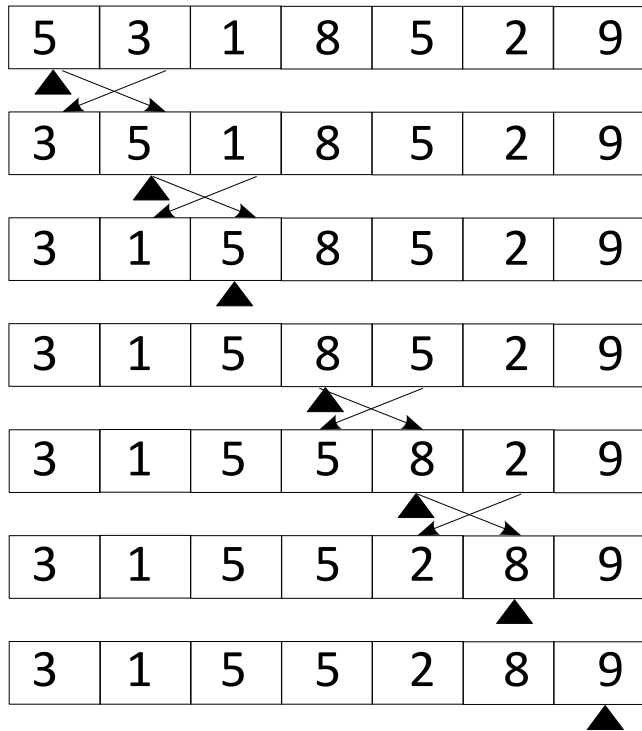


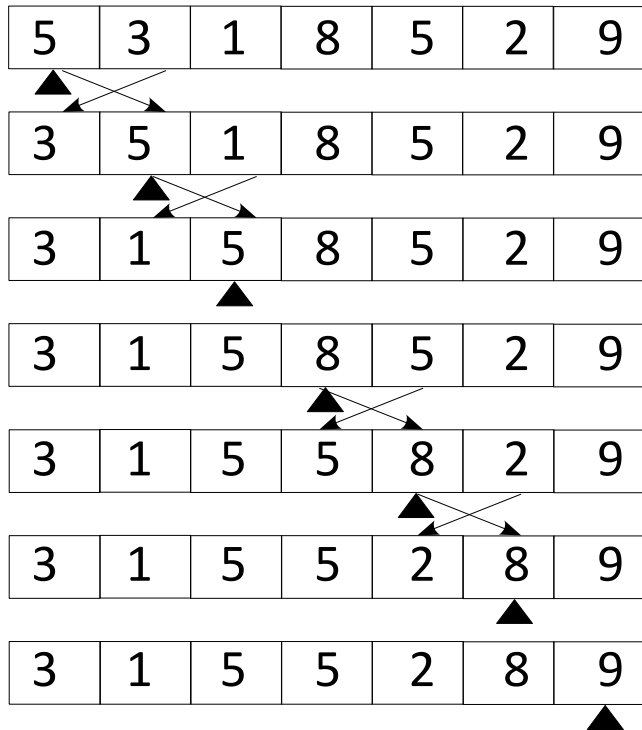
Tableau trié ?

# Tri à bulles

**Idée** : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la  $i$ -ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :

*Étape 1 :*



*Étape 2 :*

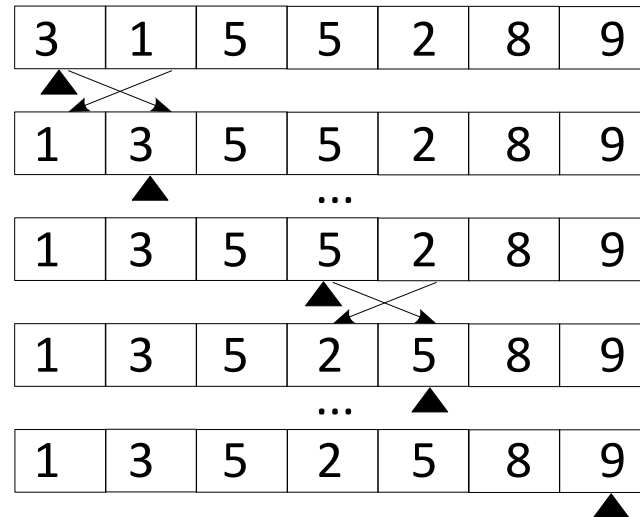


Tableau trié ?

# Tri à bulles

**Idée** : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la  $i$ -ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :

*Étape 2 :*

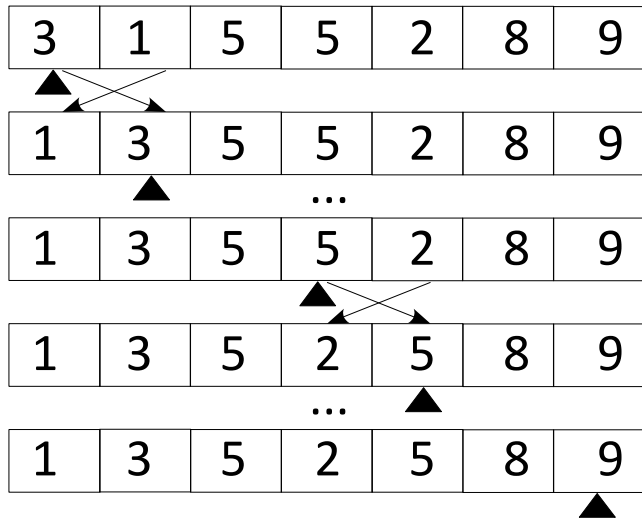
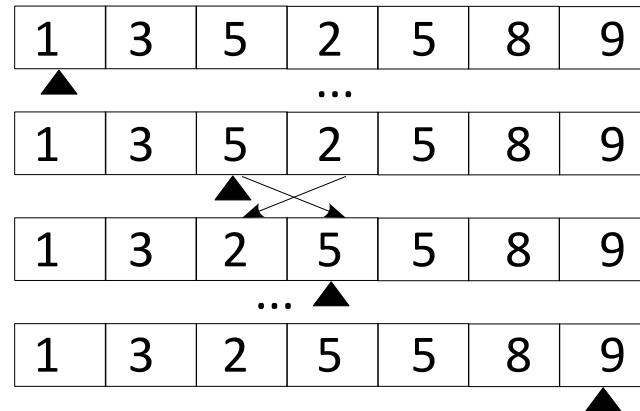


Tableau trié ?

*Étape 3 :*



# Tri à bulles

**Idée** : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la  $i$ -ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :

*Étape 3 :*

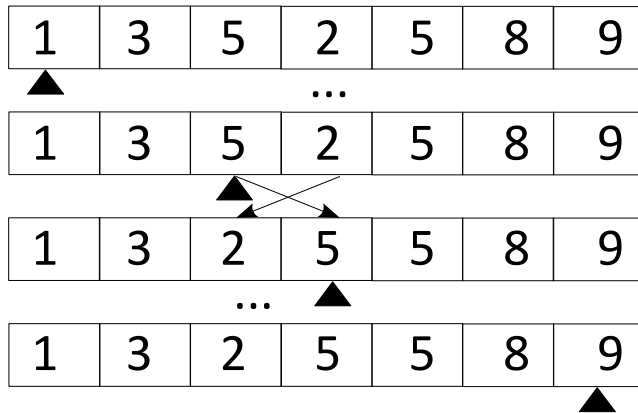


Tableau trié ?

*Étape 4 :*

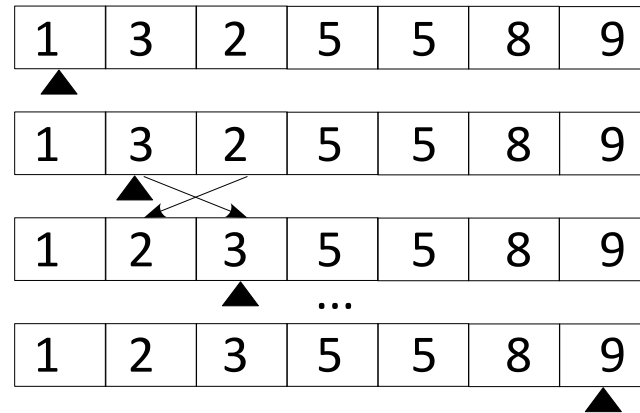


Tableau trié ?

Oui.

# Plan du cours 1 – Récursivité et tris

---

- Introduction à la récursivité
- Traces d'exécution de fonctions récursives
- Les tris
- Le tri par sélection
- Le tri à bulles
- Complexité des tris

# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans tous les cas :  $\leq n$  étapes,  $n-1$  comparaisons par étape  
donc  $\leq n \times (n-1)$  comparaisons au total

## Tri par sélection

Pour  $n$  éléments :  $n$  étapes,  $n-i$  comparaisons par étape pour trouver le min

# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans tous les cas :  $\leq n$  étapes,  $n-1$  comparaisons par étape  
donc  $\leq n \times (n-1)$  comparaisons au total

## Tri par sélection

Pour  $n$  éléments :  $n$  étapes,  $n-i$  comparaisons par étape pour trouver le min

étape 1 :  $n-1$  comparaisons

étape 2 :  $n-2$  comparaisons

...

étape  $n-2$  : 2 comparaisons

étape  $n-1$  : 1 comparaison

étape  $n$  : 0 comparaison

# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

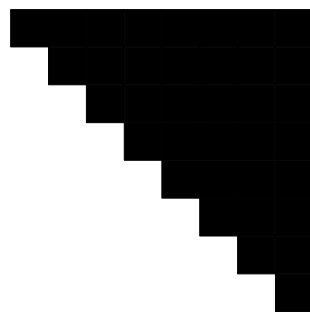
Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans tous les cas :  $\leq n$  étapes,  $n-1$  comparaisons par étape  
donc  $\leq n \times (n-1)$  comparaisons au total

## Tri par sélection

Pour  $n$  éléments :  $n$  étapes,  $n-i$  comparaisons par étape pour trouver le min



étape 1 :  $n-1$  comparaisons

étape 2 :  $n-2$  comparaisons

...

étape  $n-2$  : 2 comparaisons

étape  $n-1$  : 1 comparaison

étape  $n$  : 0 comparaison

nombre total de comparaisons =  
nombre de carrés du triangle noir

# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

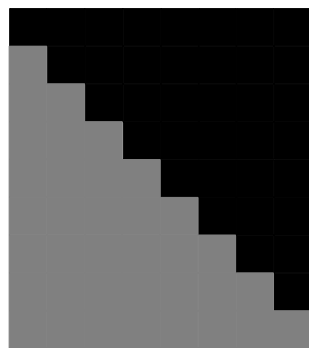
Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans tous les cas :  $\leq n$  étapes,  $n-1$  comparaisons par étape  
donc  $\leq n \times (n-1)$  comparaisons au total

## Tri par sélection

Pour  $n$  éléments :  $n$  étapes,  $n-i$  comparaisons par étape pour trouver le min



étape 1 :  $n-1$  comparaisons

étape 2 :  $n-2$  comparaisons

...

étape  $n-2$  : 2 comparaisons

étape  $n-1$  : 1 comparaison

étape  $n$  : 0 comparaison

nombre total de comparaisons =  
nombre de carrés du triangle noir =  
nombre de carrés du rectangle divisé par 2 =

# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

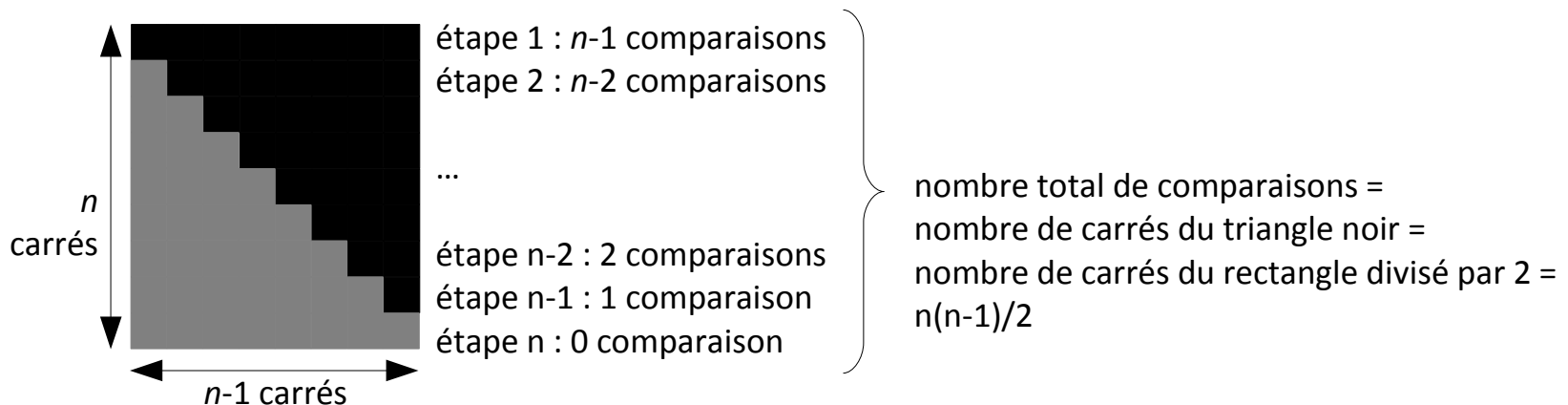
Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans tous les cas :  $\leq n$  étapes,  $n-1$  comparaisons par étape  
donc  $\leq n \times (n-1)$  comparaisons au total

## Tri par sélection

Pour  $n$  éléments :  $n$  étapes,  $n-i$  comparaisons par étape pour trouver le min



# Complexité des tris

Combien de comparaisons ? (lié au **temps d'exécution** de l'algorithme)

Sur un exemple ou dans le pire des cas, pour  $n$  éléments.

## Tri à bulles

Dans le pire des cas (tableau trié dans le sens inverse) :  $n \times (n-1)$  comparaisons

Dans le meilleur cas (tableau trié) :  $n-1$  comparaisons

## Tri par sélection

Dans tous les cas :  $n \times (n-1)/2$  comparaisons