Proprieté :

Soit T un tableau d'entiers trié entre les indices i et j (i<=j).

Soit e un entier quelconque, alors on a l'une des propriétés suivantes :

- e≤T[i]
- il existe un unique entier k dans [i..j-1] tel que T[k]<e≤T[k+1]
- e>T[j]

On déduit de cette propriété deux algorithmes permettant de trier un tableau.

Proprieté :

0	1	2	3	4	5	6	7	8	9
1	3	4	8	10	7	12	9	5	6

Idées ?

Proprieté :

0	1	2	3	4	5	6	7	8	9
1	3	4	8	10	7	12	9	5	6

Idées ?

1. Ajouter 8 parmi les éléments déjà triés entre eux repésentés sur fond vert ici… ->tri insertion
2. Par échange de la droite vers la gauche amener 5 à sa place à la droite du 4…. ->tri à bulle

Tri Insertion

C'est le tri des joueurs de cartes : insérer une nouvelle carte parmi celles déjà triées.

Hypothèse :

- Les éléments du tableau entre les indices 0 et i-1sont triés entre eux.

  Attention ce ne sont pas nécessairement les i plus petits éléments du tableau ce sont les i premiers d'origine triés entre eux.

- Les éléments de t d'indice supérieur ou égal à i n'ont jamais été observés.

On doit mettre t[i] à sa place.

Tri insertion : légende

0	1	2	3	4	5	6	7	8	9
12	5	1	8	10	7	4	9	3	6

Sur les diapositives à suivre 2 tableaux sont dessinés il s'agit **du même tableau** avant/pendant/après le déroulement d'une étape de de l'algorithme. Le tri par insertion n'utilise pas de tableau annexe tout se fait dans le tableau d'origine.

Légende :

Vert : éléments déjà triés entre eux, ils ne sont peut-être pas à leur place définitive

Orange : élément que l'ont veut ajouter à la zone verte pour l'étendre

e est une variable nécessaire pour le déroulement de l'algorithme

Tri insertion : étape 1

0	1	2	3	4	5	6	7	8	9
12	5	1	8	10	7	4	9	3	6

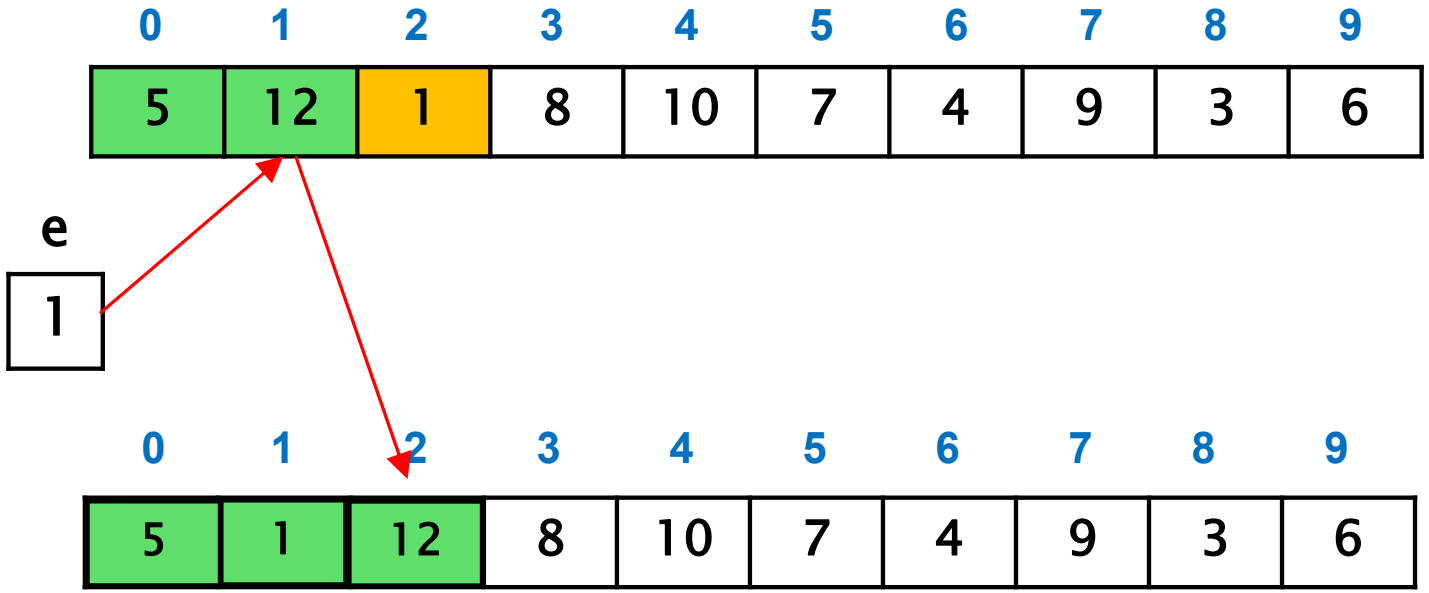
e

5

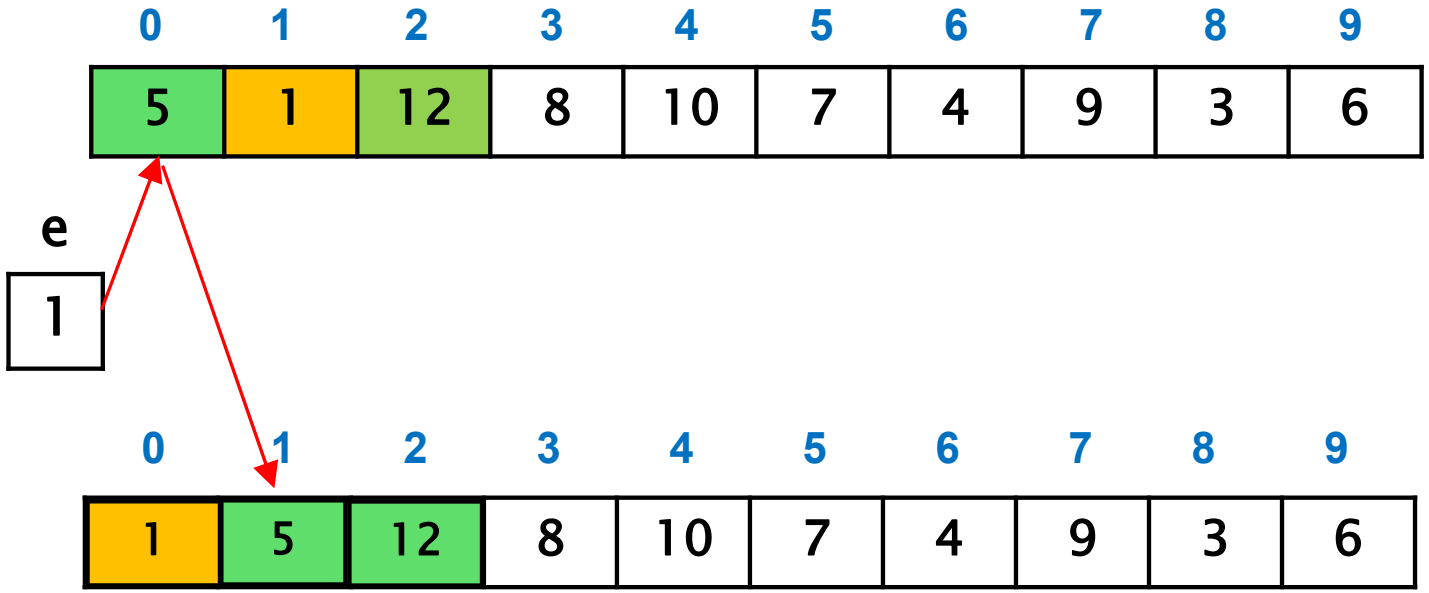
12>5 ? Oui donc décalage de 12 vers la droite
Placement de 5 en première place

0	1	2	3	4	5	6	7	8	9
5	12	1	8	10	7	4	9	3	6

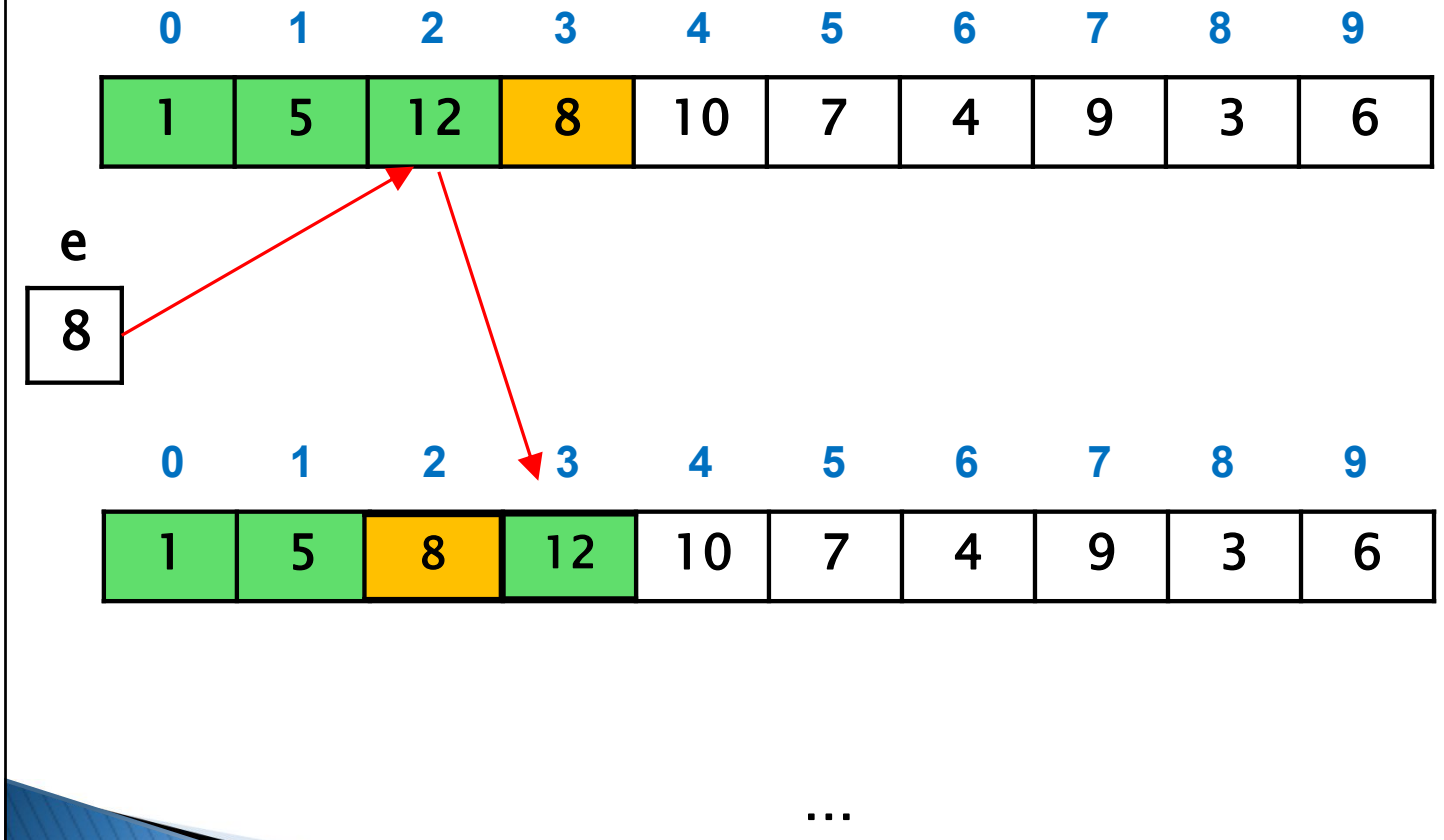
Tri insertion : étape 2



Tri insertion : étape 2

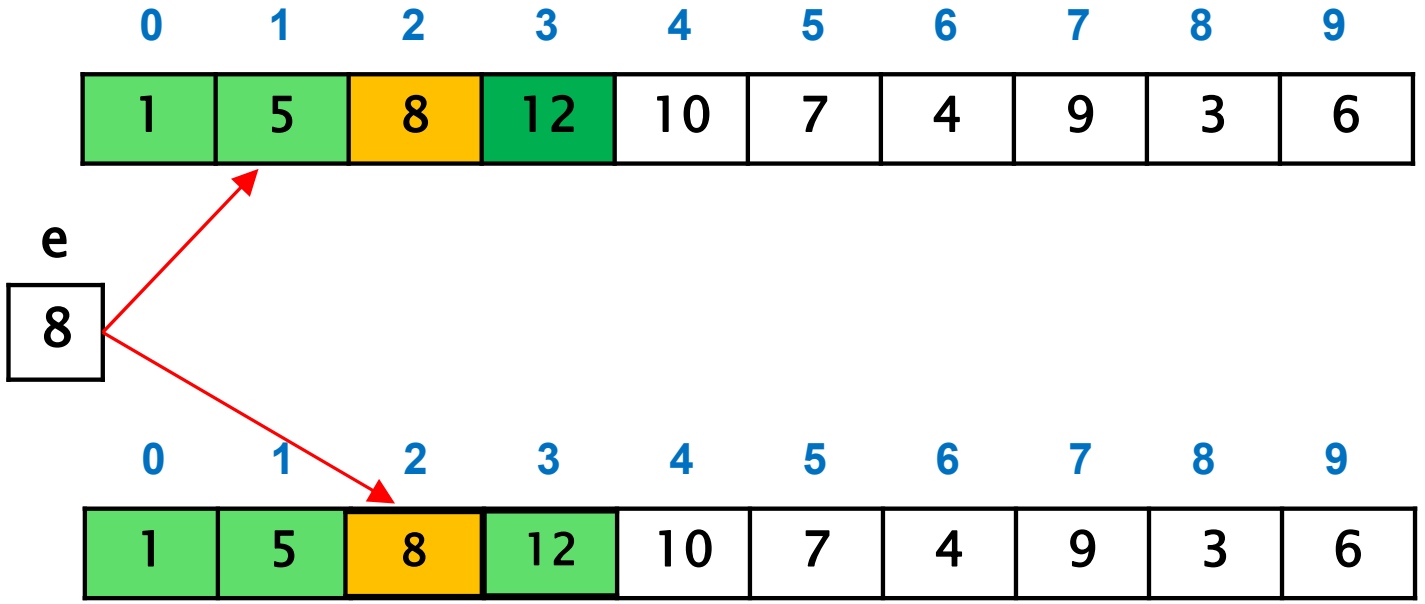


Tri insertion : : étape 3



...

Tri insertion : : étape 3



...

Tri Insertion

```
def triInsertion(t, n):
 for i in range(1,n):
  e=t[i]
  j=i-1
  while (j>=0 and t[j]>e):
    t[j+1]=t[j]
    j=j-1
  t[j+1]=e
```

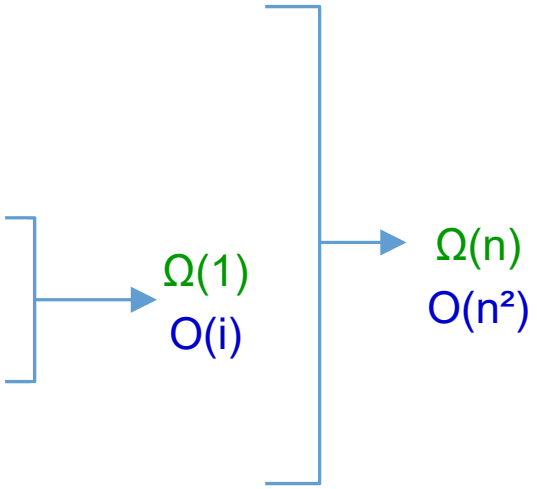
Complexité en temps au mieux? Au pire ?

Complexité en mémoire ?

Stabilité ?

Tri Insertion

```
def triInsertion(t, n):
 for i in range(1,n):
  e=t[i]
  j=i-1
  while (j>=0 and t[j]>e):
    t[j+1]=t[j]
    j=j-1
  t[j+1]=e
```



Complexité en temps au mieux? Au pire ? Ω(n) O(n²)

Complexité en mémoire ? Θ(1)

Stabilité ? Oui car test strict

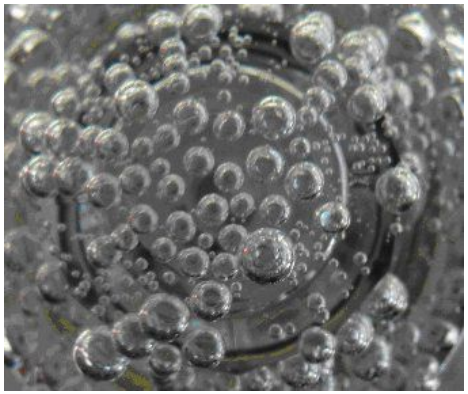
Tri insertion

Exemple : InsertSort

https://www.youtube.com/user/AlgoRythmics/videos

Tri à bulles

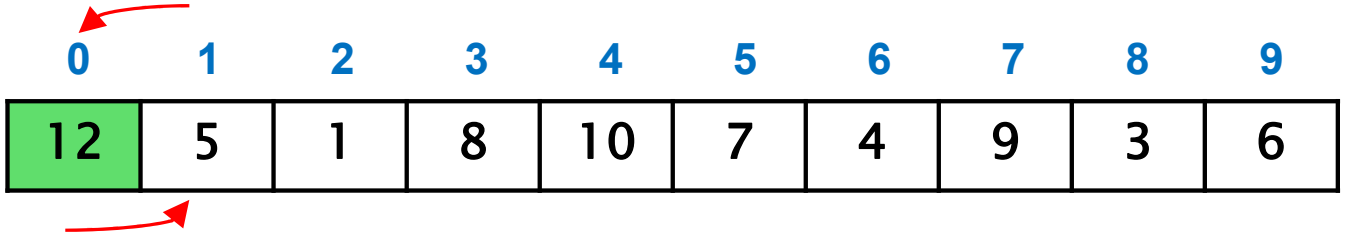
L'idée s 'appuie sur le principe Suivant : les bulles les plus grosses remontent en premier dans une boisson gazeuse.



Hypothèse de travail pour comprendre:

- Le tableau est trié définitivement entre les indices i et n-1.
- On échange deux à deux les éléments de t qui ne sont pas correctement ordonnés entre les indices 0 et i-1.

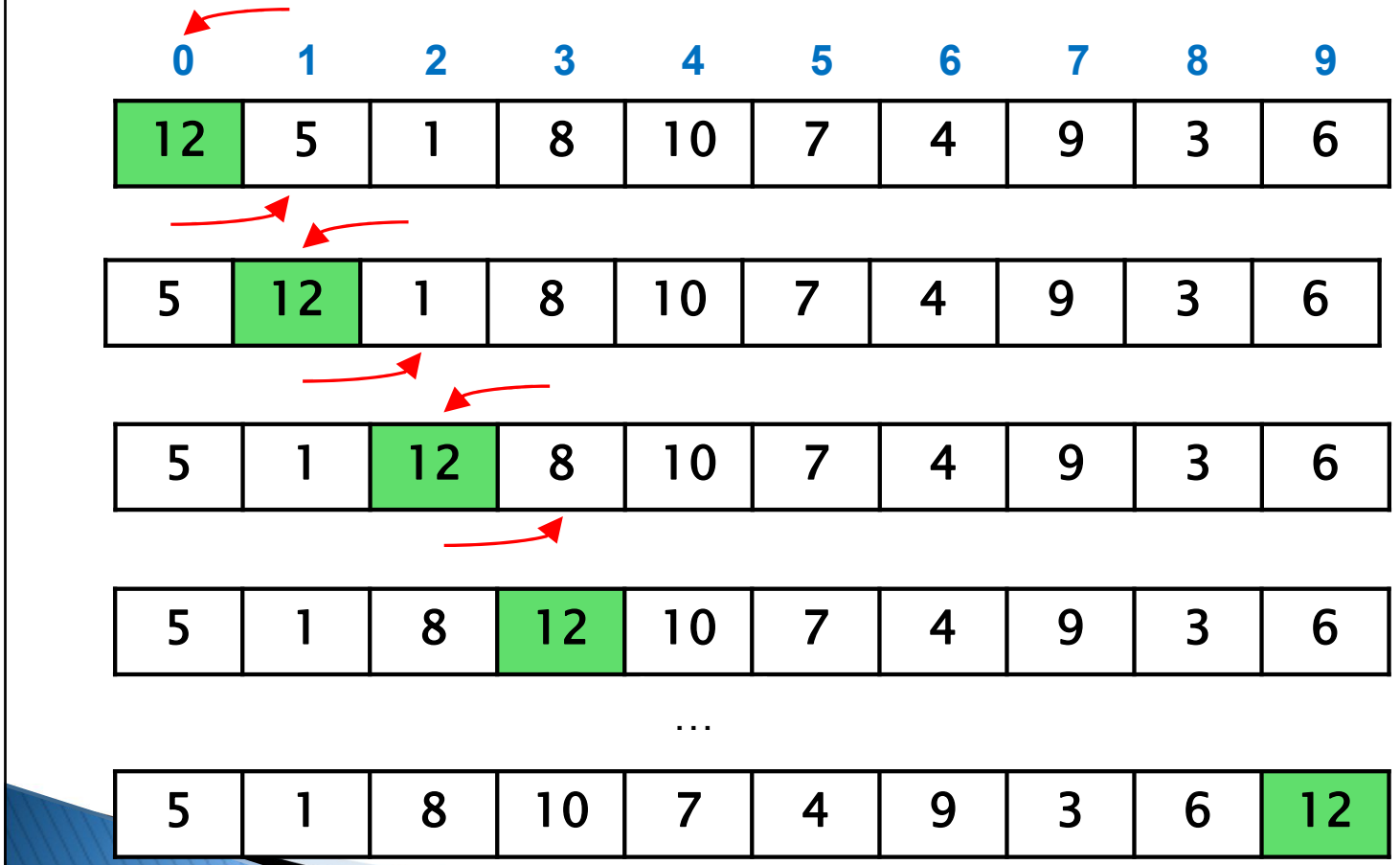
Tri à bulles : légende



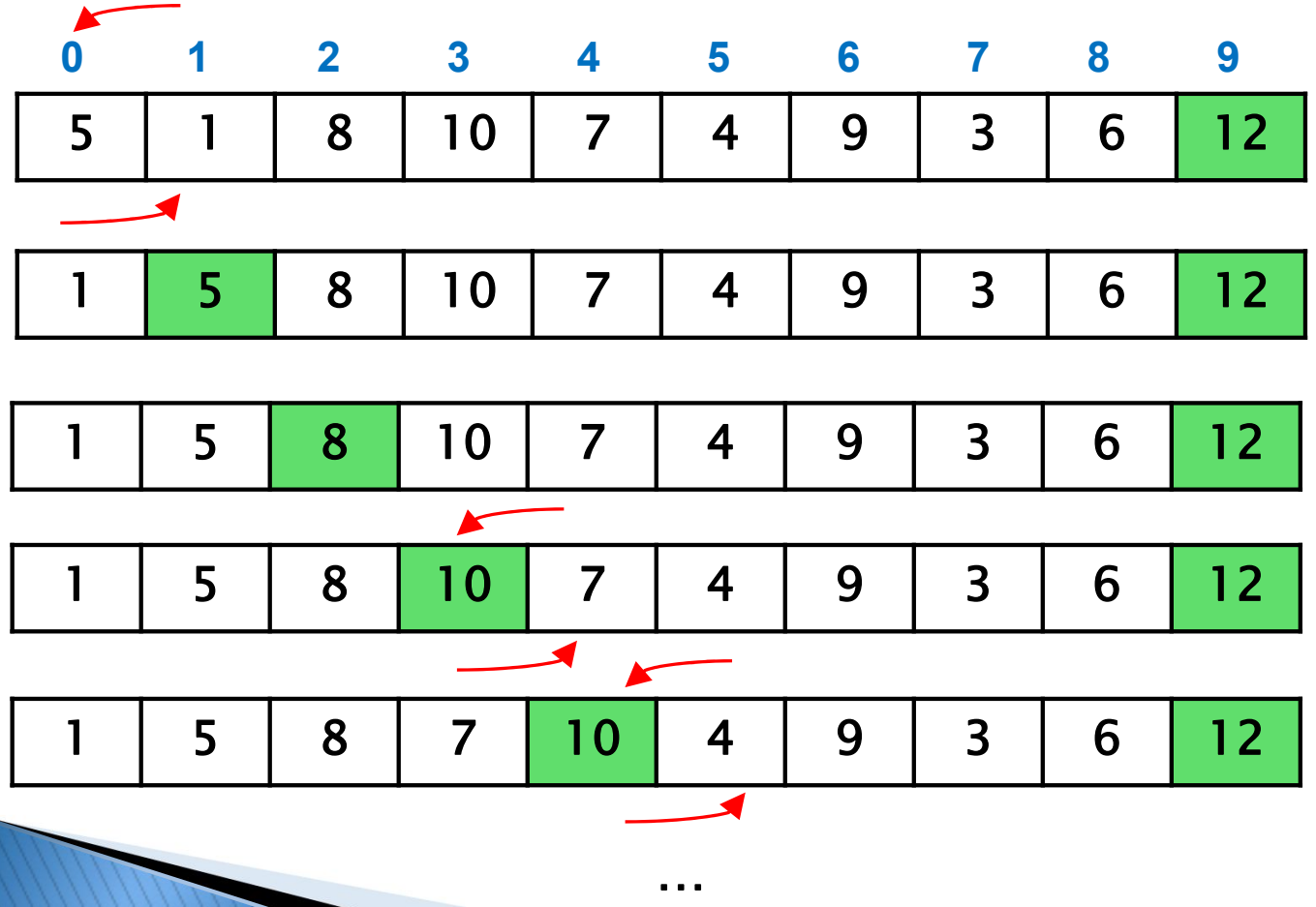
L'élément dans la case verte représente la bulle qui se déplace vers la droite (remonte)

Les flèches rouges représentent les éléments qui échangent leur position après une comparaison

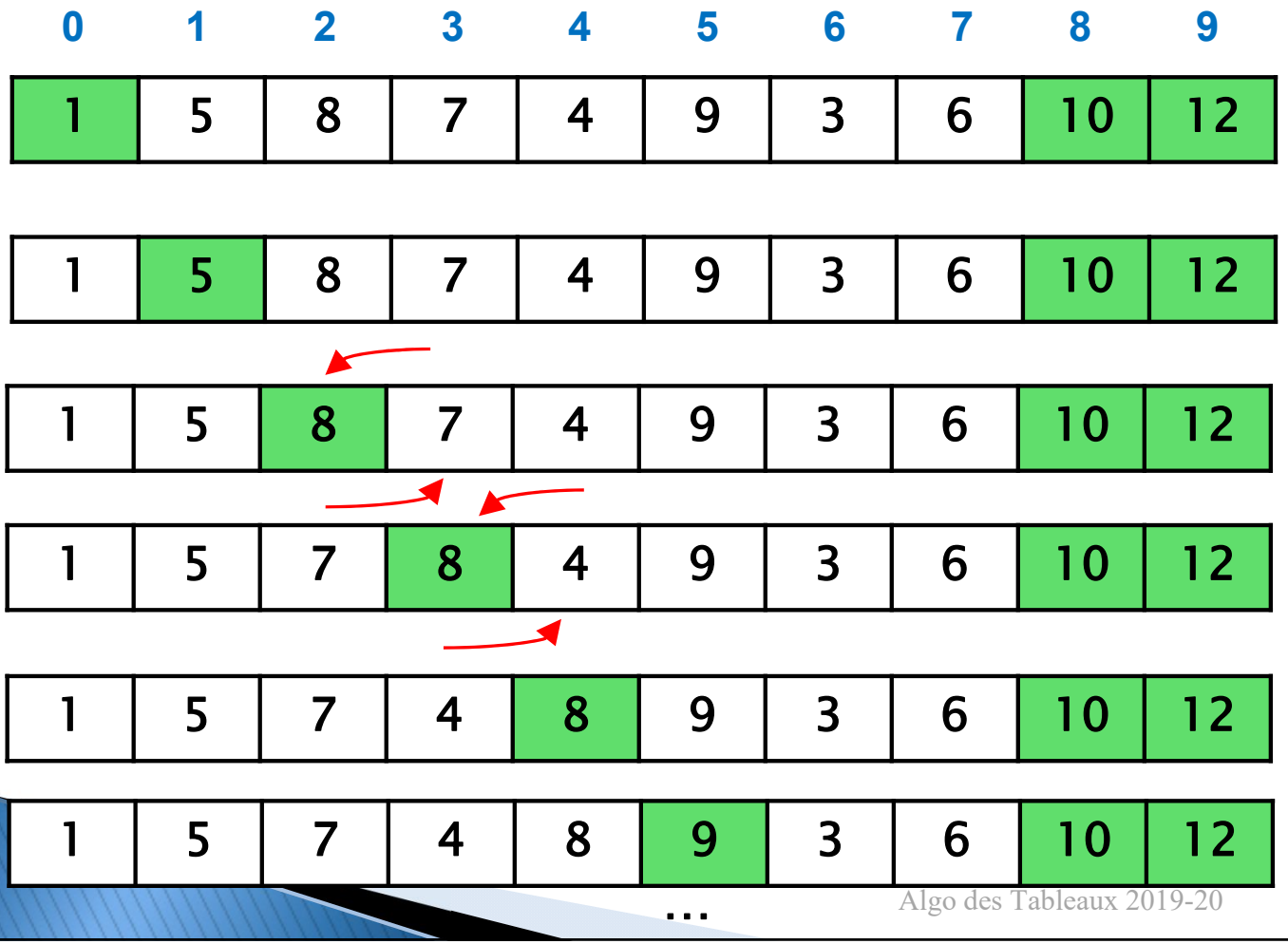
Tri à bulles


0 1 2 3 4 5 6 7 8 9
12 5 1 8 10 7 4 9 3 6
5 12 1 8 10 7 4 9 3 6
5 1 12 8 10 7 4 9 3 6
5 1 8 12 10 7 4 9 3 6
…
5 1 8 10 7 4 9 3 6 12

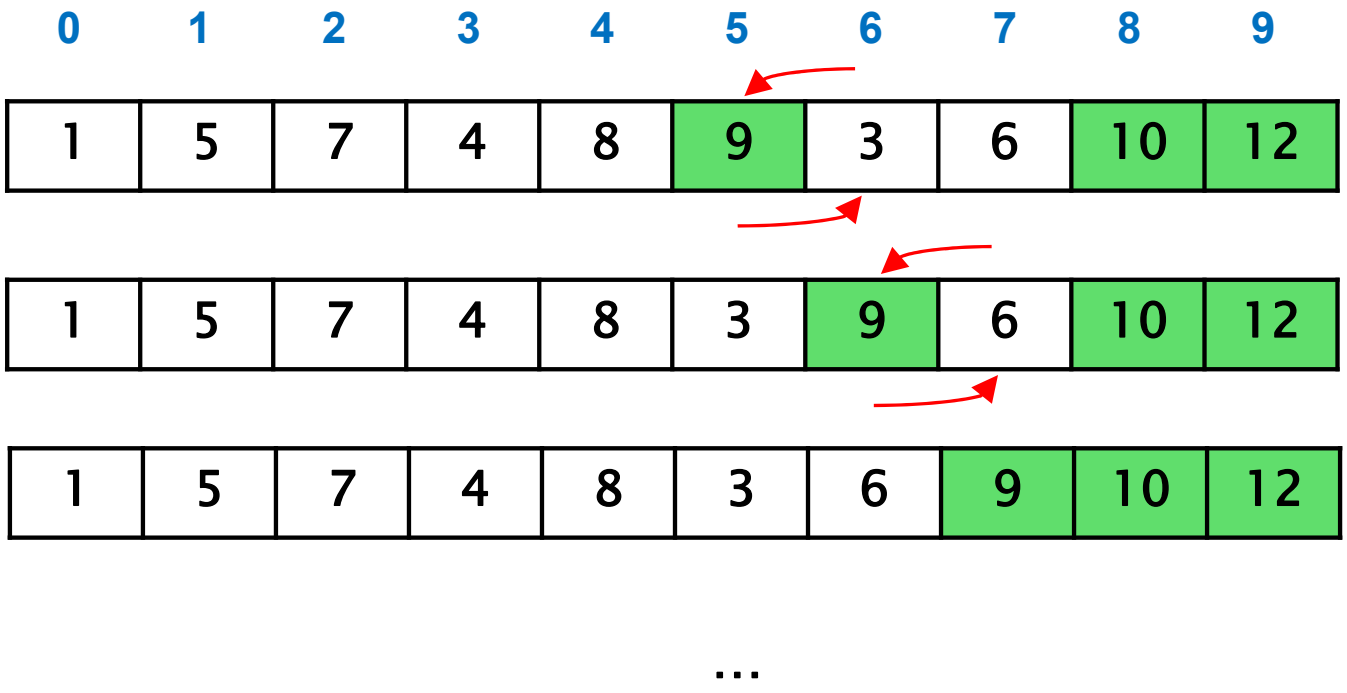
Tri à bulles


0 1 2 3 4 5 6 7 8 9
5 1 8 10 7 4 9 3 6 12
1 5 8 10 7 4 9 3 6 12
1 5 8 10 7 4 9 3 6 12
1 5 8 10 7 4 9 3 6 12
1 5 8 7 10 4 9 3 6 12
...

Tri à bulles


0 1 2 3 4 5 6 7 8 9
1 5 8 7 4 9 3 6 10 12
1 5 8 7 4 9 3 6 10 12
1 5 8 7 4 9 3 6 10 12
1 5 7 8 4 9 3 6 10 12
1 5 7 4 8 9 3 6 10 12
1 5 7 4 8 9 3 6 10 12
...

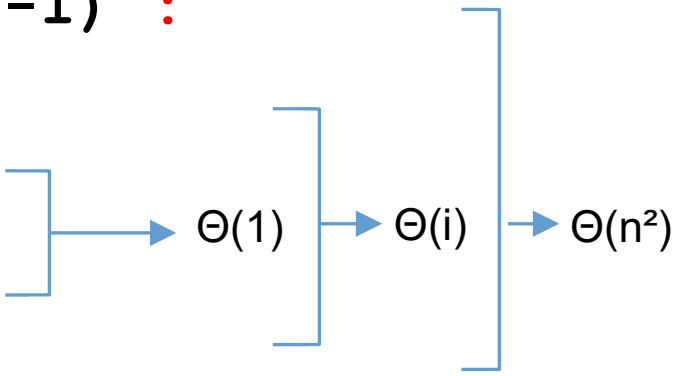
Tri à bulles



...

Tri à bulles

```
def triBulle(t,n):
 for i in range(n-1,0,-1) :
  for j in range(i) :
    if(t[j]>t[j+1]) :
      echange(t,j,j+1)
```



Complexité en temps ? $\Theta(n^2)$
Complexité en mémoire ? $\Theta(1)$
Stabilité ? Oui car test strict

Tri à bulles

Exemple :
https://www.youtube.com/user/AlgoRythmics/videos

Tris itératifs

Tri	Complexité en temps	Complexité mémoire	Stabilité
Sélection	$\Theta(n^2)$	$\Theta(1)$	non
Insertion	$\Omega(n)$ $O(n^2)$	$\Theta(1)$	oui
À bulles	$\Theta(n^2)$	$\Theta(1)$	oui

Question

- Trouver les 100 meilleurs notes en licence à l'université de Bordeaux en 2017 ?