Algorithmique et Programmation

Algorithmique – Tri d'une liste

Elise Bonzon

elise.bonzon@mi.parisdescartes.fr

LIPADE - Université Paris Descartes
http://www.math-info.univ-paris5.fr/∽bonzon/

Algorithmique

1. Tri par sélection
2. Tri par insertion
3. Tri par comptage
4. Algorithme du drapeau
5. Pour conclure

Tri par sélection

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple :

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 0$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 0$, $min = 5$ pour $j = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 0$, $min = 5$ pour $j = 2$. On échange `liste[0]` et `liste[2]`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	43	12	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 1$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	43	12	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 1$, $min = 9$ pour $j = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	43	12	9	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 1$, $min = 9$ pour $j = 3$. On échange `liste[1]` et `liste[3]`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	43	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	43	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 2$, $min = 12$ pour $j = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	43	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	43	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 3$, $min = 18$ pour $j = 4$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	43	18

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = 3$, $min = 18$ pour $j = 4$. On échange `liste[3]` et `liste[4]`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	18	43

Tri par sélection

- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- On veut **trier** la `liste`
- Parcourir la liste : pour tout $i \in [0, n-2]$
  - On cherche le plus petit élément de `liste` pour $j \in [i, n-1]$
  - On **échange** ce minimum avec `liste[i]`

Exemple : $i = n - 2$, la liste est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	3	9	12	18	43

Echange de deux éléments d'une liste

L'algorithme de tri par sélection va utiliser la procédure `echange(liste, i, j)`

- Prend en entrée une liste et deux indices
- Echange les éléments de la liste correspondant à ces deux indices
- Le type `list` étant mutable, il n'est pas utile de retourner la liste donnée en argument d'appel, elle est directement modifiée par la procédure !

Echange de deux éléments d'une liste

L'algorithme de tri par sélection va utiliser la procédure `echange(liste, i, j)`

- Prend en entrée une liste et deux indices
- Echange les éléments de la liste correspondant à ces deux indices
- Le type `list` étant mutable, il n'est pas utile de retourner la liste donnée en argument d'appel, elle est directement modifiée par la procédure !

```
def echange(liste, i, j):
    """List x Int x Int --> None
    Echange les élements de liste en position i et j"""

    elem = liste[i]
    liste[i] = liste[j]
    liste[j] = elem
```

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Opérations significatives :

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Echange de deux éléments de la liste

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Echange de deux éléments de la liste
- Le nombre de comparaisons ne dépend pas des données de la liste à trier. Tous les cas sont équivalents en terme de complexité

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Echange de deux éléments de la liste
- Le nombre de comparaisons ne dépend pas des données de la liste à trier. Tous les cas sont équivalents en terme de complexité
- Le nombre d'échanges dépend de la liste à trier

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1 \rightarrow$ **(n-1)** comparaisons

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1 \rightarrow$ **(n-1)** comparaisons
  - $i = 1$; boucle de $j = 2$ à $j = n - 1 \rightarrow$ **(n-2)** comparaisons

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1$ $\rightarrow$ **(n-1)** comparaisons
  - $i = 1$; boucle de $j = 2$ à $j = n - 1$ $\rightarrow$ **(n-2)** comparaisons
  - ...

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1 \rightarrow$ **(n-1)** comparaisons
  - $i = 1$; boucle de $j = 2$ à $j = n - 1 \rightarrow$ **(n-2)** comparaisons
  - ...
  - $i = n - 3$; boucle de $j = n - 2$ à $j = n - 1 \rightarrow$ **2** comparaisons

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1 \rightarrow$ **(n-1)** comparaisons
  - $i = 1$; boucle de $j = 2$ à $j = n - 1 \rightarrow$ **(n-2)** comparaisons
  - ...
  - $i = n - 3$; boucle de $j = n - 2$ à $j = n - 1 \rightarrow$ **2** comparaisons
  - $i = n - 2$; boucle de $j = n - 1$ à $j = n - 1 \rightarrow$ **1** comparaisons

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexité pour les comparaisons :
  - $i = 0$; boucle de $j = 1$ à $j = n - 1 \rightarrow$ **(n-1)** comparaisons
  - $i = 1$; boucle de $j = 2$ à $j = n - 1 \rightarrow$ **(n-2)** comparaisons
  - ...
  - $i = n - 3$; boucle de $j = n - 2$ à $j = n - 1 \rightarrow$ **2** comparaisons
  - $i = n - 2$; boucle de $j = n - 1$ à $j = n - 1 \rightarrow$ **1** comparaisons

$\Rightarrow (n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$ comparaisons.
Complexité de l'ordre de $O(n^2)$

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexite pour les échanges :

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexite pour les échanges :
  - **Meilleur cas** : la liste est déjà triée, on ne fait aucun échange

Tri par sélection

```
def tri_selection(liste) :
    """List --> None -- Trie la liste donnée en paramètre.
    La liste est directement modifiée par la procédure."""
    n = len(liste)
    for i in range(n-1):
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < liste[indice_min]:
                indice_min = j
        if indice_min != i:
            echange(liste, i, indice_min)
```

- Complexite pour les échanges :
  - **Meilleur cas** : la liste est déjà triée, on ne fait aucun échange
  - **Pire cas** : on fait un échange à chaque tour de boucle : n échanges

Tri par insertion

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos == 0$ ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple :

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 1$, `liste[0:0]` est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 1$, `liste[0:0]` est triée, $pos = 1$, `elem` = 43

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 1$, `liste[0:0]` est triée, $pos = 1$, `elem` = 43 ; `liste[0]` $<$ `elem`, `elem` est en bonne position

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 2$, `liste[0:1]` est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 2$, `liste[0:1]` est triée, $pos = 2$, `elem` = 5

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 2$, `liste[0:1]` est triée, $pos = 2$, `elem` = 5 ; `liste[1]` $>$ `elem`, `liste[2] = liste[1]`, $pos = 1$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	43	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 2$, `liste[0:1]` est triée, $pos = 1$, `elem` = 5 ; `liste[0]` $>$ `elem`, `liste[1] = liste[0]`, $pos = 0$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	12	43	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 2$, `liste[0:1]` est triée, $pos = 0$, `elem` = 5 ;
$pos ==$ 0, `liste[0]` = `elem`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	12	43	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 3$, `liste[0:2]` est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	12	43	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 3$, `liste[0:2]` est triée, $pos = 3$, `elem` = 9

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	12	43	9	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 3$, `liste[0:2]` est triée, $pos = 3$, `elem` = 9 ; `liste[2]` $>$ `elem`, `liste[3] = liste[2]`, $pos = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	12	43	43	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 3$, `liste[0:2]` est triée, $pos = 2$, `elem` = 9 ; `liste[1]` $>$ `elem`, `liste[2] = liste[1]`, $pos = 1$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	12	12	43	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 3$, `liste[0:2]` est triée, $pos = 1$, `elem` = 9 ;
`liste[0]` $<$ `elem`, `liste[1]` = `elem`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	43	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 4$, `liste[0:3]` est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	43	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 4$, `liste[0:3]` est triée, $pos = 4$, `elem` = 18

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	43	18

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 4$, `liste[0:3]` est triée, $pos = 4$, `elem` = 18 ; `liste[3]` $>$ `elem`, `liste[4] = liste[3]`, $pos = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	43	43

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : $i = 4$, `liste[0:3]` est triée, $pos = 3$, `elem` = 18 ; `liste[2]` $<$ `elem`, `liste[3]` = `elem`

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	18	43

Tri par insertion

- Algorithme qu'utilise naturellement l'être humain pour trier des objets, comme par exemple des cartes à jouer
- Soit `liste` une liste non triée de longueur n, de type `list[elem]`
- Soit $i \in [1, n-1]$. A l'étape i
  - On suppose que les éléments d'indice 0 à $i-1$ sont déjà triés
  - On insère l'élément d'indice i à sa place dans la liste `liste[0:i-1]` :
    - $pos = i$, sauvegarde de `elem` = `liste[i]`
    - Tant que `liste[pos-1]` $>$ `elem`, `liste[pos]` = `liste[pos-1]` ; $pos = pos - 1$
    - Si $pos ==$ 0 ou `liste[pos-1]` $<=$ `elem`, `liste[pos]` = `elem`

Exemple : `liste` est triée

Indice	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	18	43

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

- Opérations significatives :

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Affectations d'un élément de la liste à `elem`, ou d'un élément de la liste à un autre

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Affectations d'un élément de la liste à `elem`, ou d'un élément de la liste à un autre
- Le nombre de comparaisons et d'affectations dépend de la liste à trier

Tri par insertion

```
def tri_insertion(liste):
    """List --> None
    Tri la liste donnée en paramètre"""
    for i in range(1, len(liste)):
        elem = liste[i]
        pos = i
        while pos > 0 and liste[pos - 1] > elem :
            liste[pos] = liste[pos-1]
            pos = pos - 1
        liste[pos] = elem
```

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Affectations d'un élément de la liste à `elem`, ou d'un élément de la liste à un autre
- Le nombre de comparaisons et d'affectations dépend de la liste à trier
- Complexité vu en TD

Tri par comptage

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple :

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**0**	**0**	**0**	**0**	**0**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 0$, $j = 1$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**0**	**1**	**0**	**0**	**0**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 0$, $j = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**1**	**1**	**0**	**0**	**0**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 0$, $j = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**1**	**0**	**0**	**0**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 0$, $j = 4$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**1**	**0**	**0**	**1**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 1$, $j = 2$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**2**	**0**	**0**	**1**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 1$, $j = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**3**	**0**	**0**	**1**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 1$, $j = 4$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**4**	**0**	**0**	**1**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 2$, $j = 3$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**4**	**0**	**1**	**1**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 2$, $j = 4$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**4**	**0**	**1**	**2**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple : $i = 3$, $j = 4$

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**4**	**0**	**1**	**3**

Tri par comptage

- Principe : déterminer pour chaque élément de `liste` le nombre d'éléments qui lui sont inférieurs ou égaux
- Pour trouver `ind(i)`, $0 \leq i \leq n-2$, donnant la position de `liste[i]` dans la liste triée, on compare `liste[i]` à tous les `liste[j]`, $j \in [i+1, n-1]$:
  - Soit `liste[j] <= liste[i]` : on incrémente $ind(i)$ de 1
  - Soit `liste[j] > liste[i]` : on incrémente $ind(j)$ de 1

Exemple :

Indice	*0*	*1*	*2*	*3*	*4*
Elément	12	43	5	9	18
Ind	**2**	**4**	**0**	**1**	**3**

Liste triée :

Ind	*0*	*1*	*2*	*3*	*4*
Elément	5	9	12	18	43

Tri par comptage

```
def tri_comptage(liste):
    """List --> List
    Tri la liste donnée en paramètre"""
    ind = []
    result = []
    n = len(liste)
    # initialisation des listes indices et résultat
    for i in range(n) :
        ind.append(0)
        result.append(0)
    for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1

    for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]

    return result
```

Tri par comptage complexité (1)

- Opérations significatives :
  - Comparaison de deux éléments de la liste
  - Affectations d'un élément à la liste `ind`, et à la liste `result`
- Le nombre de comparaisons et d'affectations ne dépend pas de la liste à trier. Tous les cas sont équivalents en terme de complexité.

Tri par comptage complexité (2)

- Comparaisons :

```
for i in range(n - 1): #comptage
    for j in range(i+1, n) :
        if liste[j] > liste[i] :
```

Tri par comptage complexité (2)

- Comparaisons :

```
for i in range(n - 1): #comptage
    for j in range(i+1, n) :
        if liste[j] > liste[i] :
```

  - Pour $i = 0$, j va de 1 à $(n-1) \longrightarrow n-1$ comparaisons

Tri par comptage complexité (2)

- Comparaisons :

```
for i in range(n - 1): #comptage
    for j in range(i+1, n) :
        if liste[j] > liste[i] :
```

  - Pour $i = 0$, j va de 1 à $(n-1) \longrightarrow n-1$ comparaisons
  - Pour $i = 1$, j va de 2 à $(n-1) \longrightarrow n-2$ comparaisons

    $\vdots$

Tri par comptage complexité (2)

- Comparaisons :

```
for i in range(n - 1): #comptage
    for j in range(i+1, n) :
        if liste[j] > liste[i] :
```

  - Pour $i = 0$, j va de 1 à $(n-1) \longrightarrow n-1$ comparaisons
  - Pour $i = 1$, j va de 2 à $(n-1) \longrightarrow n-2$ comparaisons

    $\vdots$
  - Pour $i = n-2$, j va de $(n-1)$ à $(n-1) \longrightarrow 1$ comparaison

Tri par comptage complexité (2)

- Comparaisons :

```
for i in range(n - 1): #comptage
    for j in range(i+1, n) :
        if liste[j] > liste[i] :
```

  - Pour $i = 0$, j va de 1 à $(n-1) \longrightarrow n-1$ comparaisons
  - Pour $i = 1$, j va de 2 à $(n-1) \longrightarrow n-2$ comparaisons

    $\vdots$
  - Pour $i = n-2$, j va de $(n-1)$ à $(n-1) \longrightarrow 1$ comparaison

$\Rightarrow (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$ comparaisons

Complexité de l'ordre de $O(n^2)$

Tri par comptage complexité (3)

- Affectations :

```
(1) for i in range(n) :
        ind.append(0)
        result.append(0)
(2) for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1
(3) for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]
```

Tri par comptage complexité (3)

- Affectations :

```
(1) for i in range(n) :
        ind.append(0)
        result.append(0)
(2) for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1
(3) for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]
```

(1) n append pour `ind`, n append pour `result` $\longrightarrow$ $2n$ affectations

Tri par comptage complexité (3)

- Affectations :

```
(1) for i in range(n) :
        ind.append(0)
        result.append(0)
(2) for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1
(3) for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]
```

(1) n append pour `ind`, n append pour `result` $\longrightarrow$ $2n$ affectations

(2) Autant d'affectations que de comparaisons $\longrightarrow$ $\frac{n(n-1)}{2}$ affectations

Tri par comptage complexité (3)

- Affectations :

```
(1) for i in range(n) :
        ind.append(0)
        result.append(0)
(2) for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1
(3) for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]
```

(1) n append pour `ind`, n append pour `result` $\longrightarrow$ $2n$ affectations
(2) Autant d'affectations que de comparaisons $\longrightarrow$ $\frac{n(n-1)}{2}$ affectations
(3) n affectations

Tri par comptage complexité (3)

- Affectations :

```
(1) for i in range(n) :
        ind.append(0)
        result.append(0)
(2) for i in range(n - 1): #comptage
        for j in range(i+1, n) :
            if liste[j] > liste[i] :
                ind[j] = ind[j] + 1
            else :
                ind[i] = ind[i] + 1
(3) for i in range(n) : #liste triée résultat
        result[ind[i]] = liste[i]
```

(1) n append pour `ind`, n append pour `result` $\longrightarrow 2n$ affectations

(2) Autant d'affectations que de comparaisons $\longrightarrow \frac{n(n-1)}{2}$ affectations

(3) n affectations

$\Rightarrow$ Complexité de l'ordre de $O(n^2 + 3n)$

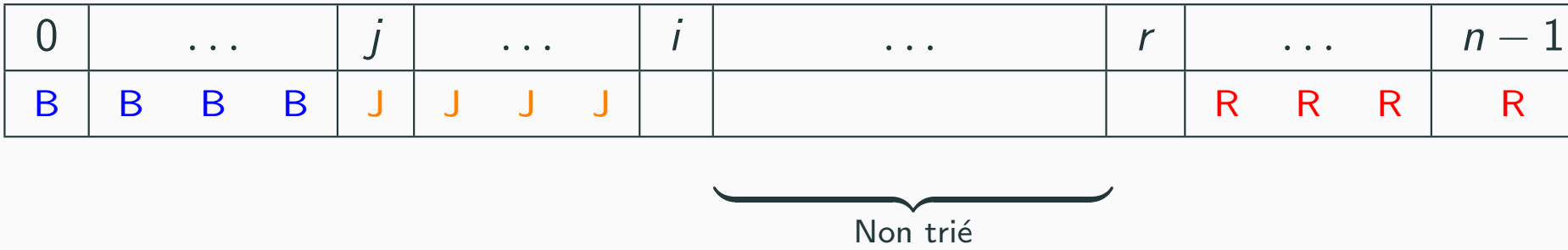
Algorithme du drapeau

Algorithme du drapeau à 3 couleurs

- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.

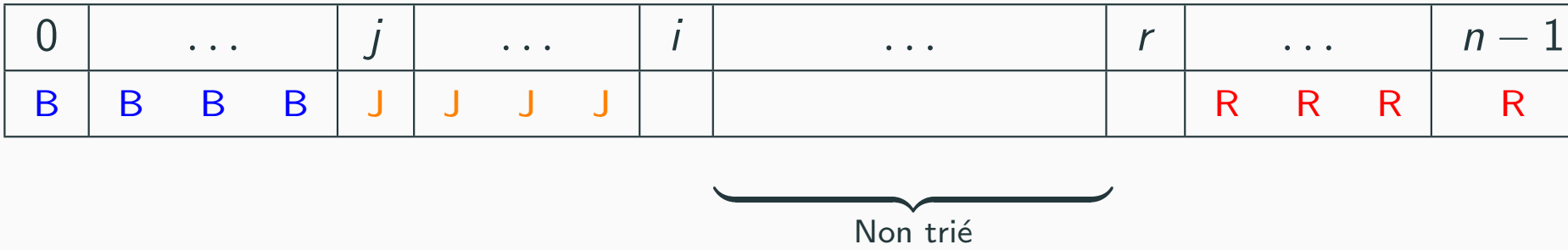
Algorithme du drapeau à 3 couleurs

- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



Algorithme du drapeau à 3 couleurs

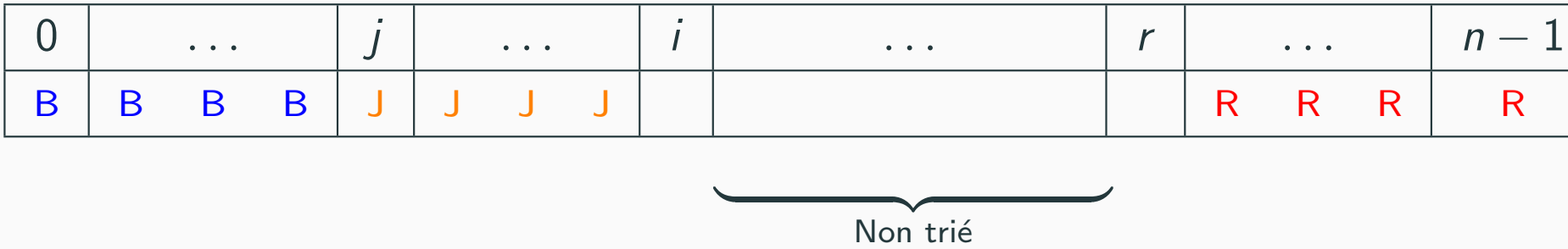
- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



- Deux cas possibles
  - $i = r + 1$. C'est fini, la liste est triée

Algorithme du drapeau à 3 couleurs

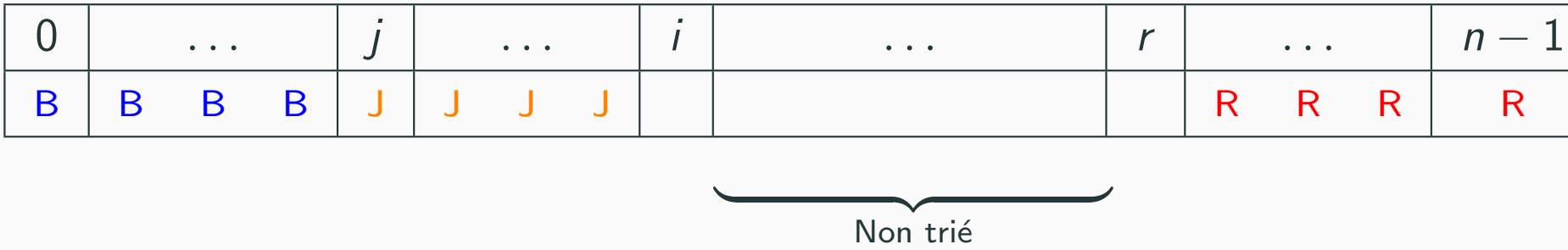
- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



- Deux cas possibles
  - $i = r + 1$. C'est fini, la liste est triée
  - $i \leq r$, 3 cas possibles

Algorithme du drapeau à 3 couleurs

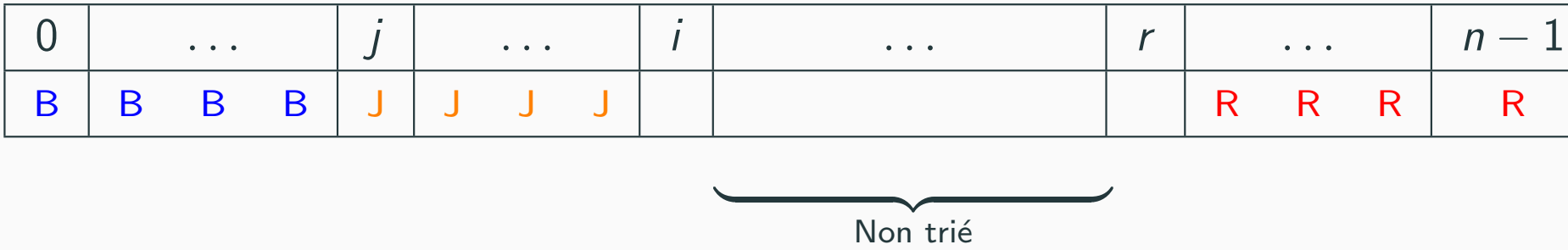
- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



- Deux cas possibles
  - $i = r + 1$. C'est fini, la liste est triée
  - $i \leq r$, 3 cas possibles
    - `liste[i]` $== J$, $i = i + 1$

Algorithme du drapeau à 3 couleurs

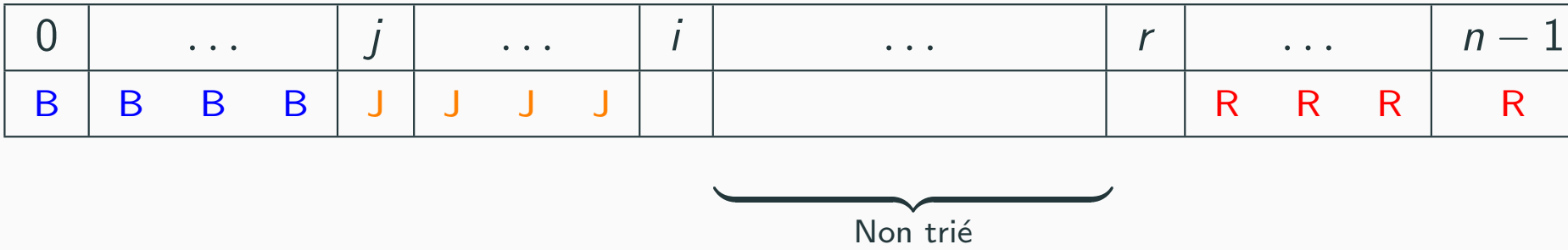
- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



- Deux cas possibles
  - $i = r + 1$. C'est fini, la liste est triée
  - $i \leq r$, 3 cas possibles
    - `liste[i]` $==$ *J*, $i = i + 1$
    - `liste[i]` $==$ *B*, `echange(liste, j, i)`; $i = i + 1$; $j = j + 1$

Algorithme du drapeau à 3 couleurs

- Soit `liste` une liste non triée de longueur n contenant des données de 3 types : *R*ouge, *B*leu et *J*aune. On veut trier la liste de façon à ce que les premiers éléments soient bleus, les suivants jaunes, puis enfin les derniers rouges.



- Deux cas possibles
  - $i = r + 1$. C'est fini, la liste est triée
  - $i \leq r$, 3 cas possibles
    - `liste[i]` $==$ *J*, $i = i + 1$
    - `liste[i]` $==$ *B*, `echange(liste, j, i)`; $i = i + 1$; $j = j + 1$
    - `liste[i]` $==$ *R*, `echange(liste, r, i)`; $r = r - 1$

Algorithme du drapeau

```
def drapeau(liste):
    """List --> None
    Tri la liste, contenant 3 couleurs R, J, B, donnée en paramètr
    i = 0
    j = 0
    r = len(liste) - 1
    while i <= r:
        if liste[i] == "J":
            i = i + 1
        elif liste[i] == "B" :
            echange(liste, j, i)
            i = i + 1
            j = j + 1
       else :
           echange(liste, r, i)
           r = r - 1
```

Pour conclure

Résumé du cours

Aujourd'hui, on a vu 4 algorithmes classiques de tri :

- Tri par sélection
- Tri par insertion
- Tri par comptage
- Algorithme du drapeau
- Il existe de nombreux autres algorithmes de tri, plus efficaces, que vous verrez ultérieurement !