INF155 – SÉANCE 6
LES TABLEAUX – SUITE ET FIN (2 DIMENSIONS ET TRI)

Anis Boubaker, Ph.D.
Maître d'enseignement
École de Technologie Supérieure

PLAN DE LA SÉANCE

- Les tableaux à deux dimensions
  - Déclaration et accès
  - Parcours d'un tableau 2D
  - Allocation mémoire
  - Passage en paramètre
- Les algorithmes de tri sur place
  - Introduction
  - Tri par insertion
  - Tri par sélection
  - Tri à bulles

TABLEAUX À PLUSIEURS DIMENSIONS

TABLEAUX 1D, 2D ... XD

- Jusqu'à maintenant, nous avons vu comment créer des tableaux linéaires, à une dimension.

- Il est possible de créer des tableaux à plusieurs dimensions, pour ainsi mieux organiser les données.

- Par exemple, avec deux dimensions, nous pouvons répartir les données en ligne et en colonnes (comme dans un chiffrier Excel...)

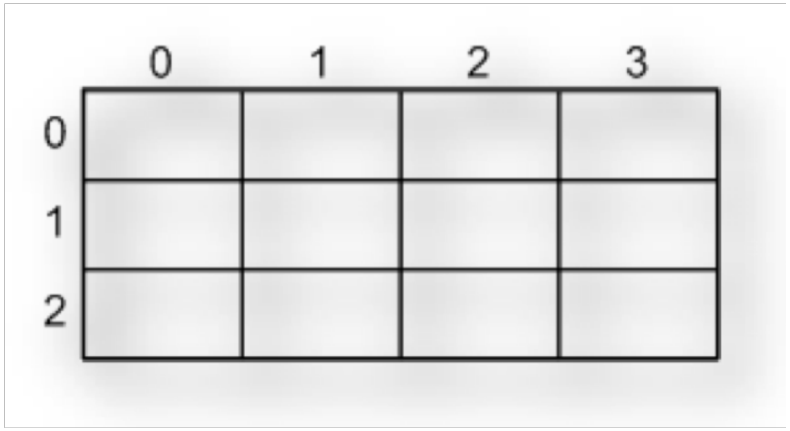
TABLEAUX À 2 DIMENSIONS

Déclaration:

```
type identifiant[nb_lignes][nb_colonnes];
```

- Exemple:

```
int tab[3][4];
```



TABLEAUX À 2 DIMENSIONS

Initialisation:

```
type identifiant[nb_l][nb_c]= {{v0_0,v0_1,...},{v1_0,v1_1,...},...};
```

- Exemple:

```
int tab[3][4] = { {1,2,4,5} , {8,4,3,2}, {7,0,0,1}};
```

	0	1	2	3
0	1	2	4	5
1	8	4	3	2
2	7	0	0	1

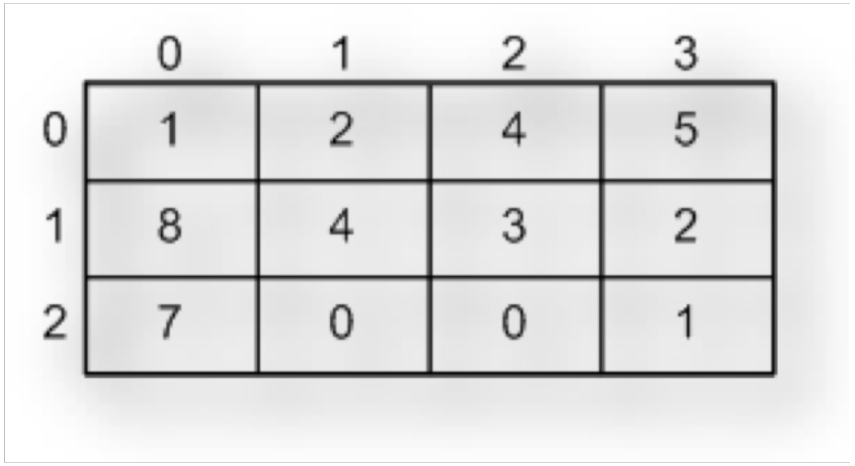
TABLEAUX À 2 DIMENSIONS

Accès à une case du tableau:

```
identifiant[indice_ligne][indice_colonne]
```

- Exemple:

```
//Affiche 2
printf("%d", tab[1][3]);
```



	0	1	2	3
0	1	2	4	5
1	8	4	3	2
2	7	0	0	1

- Rappel: Les indices commencent toujours à 0 (sur toutes les dimensions!)

PARCOURIR UN TABLEAU 2D

```
int tab[3][4]= { {1,2,4,5} , {8,4,3,2}, {7,0,0,1} };
int ligne,
    col; //Compteurs de boucle
for(ligne=0; ligne<3 ; ligne++) //Boucle sur les lignes
{
   for(col=0; col<4; col++) //Boucle sur les colonnes
   {
       printf("%d ", tab[ligne][col]);
   }
   printf("\n");
}
```

TABLEAUX 2D ET MÉMOIRE

- Lors de l'allocation d'un tableau 2D de **n lignes*m colonnes**, le compilateur réserve un espace mémoire contigu de **n*m cases** (la taille des cases dépend du type de données).
- Exemple : int tab[3][4] ;

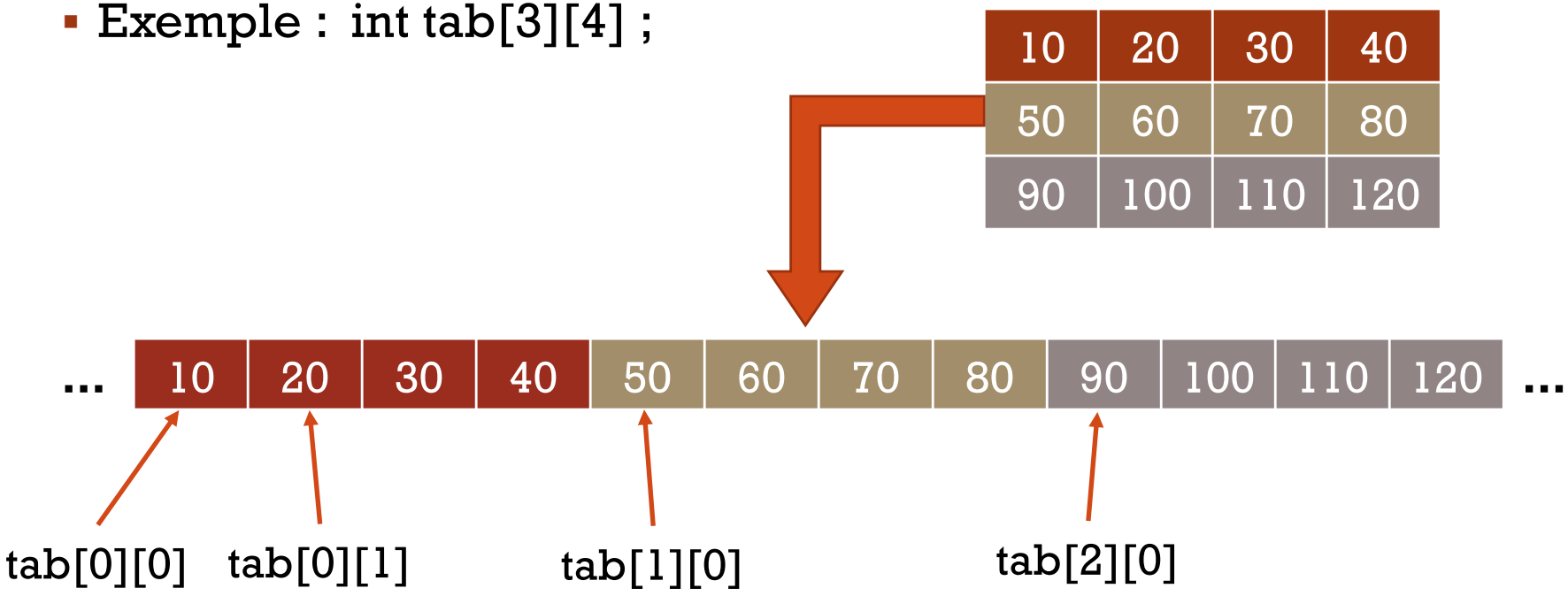


TABLEAU 2D EN PARAMÈTRE

- Comme pour un tableau 1D, il est possible de passer un tableau 2D (ou plus) en paramètre d'une fonction.
- Comme pour un tableau 1D, **il faut fournir en paramètre la taille effective du tableau sur chacune des dimensions!**

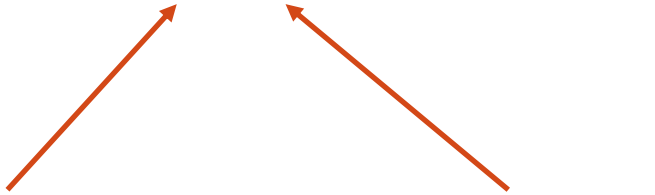
```
int ma_fonction(int tab[10][10], int nb_lignes, int nb_colonnes);
```

Taille effective sur chaque dimension

TABLEAU 2D EN PARAMÈTRE

- Contrairement aux tableaux 1D, il est obligatoire de mentionner la taille **maximale** du tableau dans le prototype, sauf pour la première dimension:

```
int ma_fonction(int tab[][10], int nb_lignes, int_nb_colonnes);
```



optionnel **obligatoire!**

EXEMPLE – FONCTION REMPLIR

```
//Note: la taille 10 devrait être une constante ici!
void remplir(int tab[][10], int nb_l, int nb_c, int val)
{
    int ligne,
        col; //Compteurs de boucle
    for(int ligne=0; ligne<nb_l; i++)
    {
       for(int col=0; col<nb_c; col++)
       {
           tab[ligne][col] = val;
       }
    }
}
```

LES ALGORITHMES DE TRI

ÉTS

TRI D'UN TABLEAU

- Un traitement très récurrent en programmation qui consiste à trier des listes d'éléments

- **Trier un tableau** consiste à mettre ses éléments dans un ordre spécifique - typiquement un ordre numérique (croissant/décroissant) ou lexicographique.

TRI D'UN TABLEAU

- Par exemple, on dispose d'un tableau d'entiers

3	10	8	10	78	55	145

- Nous souhaitons obtenir le tableau ci-dessous, **sans utiliser un autre tableau**.

3	8	10	22	55	78	145

- Pour y parvenir, nous utilisons des **algorithmes de tri**.

ALGORITHMES DE TRI

- Il existe plusieurs types d'algorithmes de tri, qui se distinguent selon:
  - leur complexité;
  - leur empreinte mémoire;
  - l'utilisation de méthodes récursives; ou
  - leur stabilité (deux élément égaux restent dans le même ordre une fois le tableau trié, pas pertinent lorsqu'on trie de nombres...).

- Beaucoup de recherche s'est faite (et se fait encore) pour améliorer les algorithmes de tri.

ALGORITHMES DE TRI

ALGORITHME	CAS OPTIMAL	CAS MOYEN	PIRE CAS
TRI RAPIDE(*)	n.log(n)	n.log(n)	n^2
TRI FUSION	n.log(n)	n.log(n)	n.log(n)
TRI PAR TAS(*)	n.log(n)	n.log(n)	n.log(n)
TRI PAR INSERTION	n	n^2	n^2
TRI À BULLES	n	n^2	n^2
TRI PAR SÉLECTION(*)	n^2	n^2	n^2

ALGORITHMES ANIMÉS

- Plusieurs site Internet proposent des animations visuelles pour comprendre le comportement des divers algorithmes de tri.

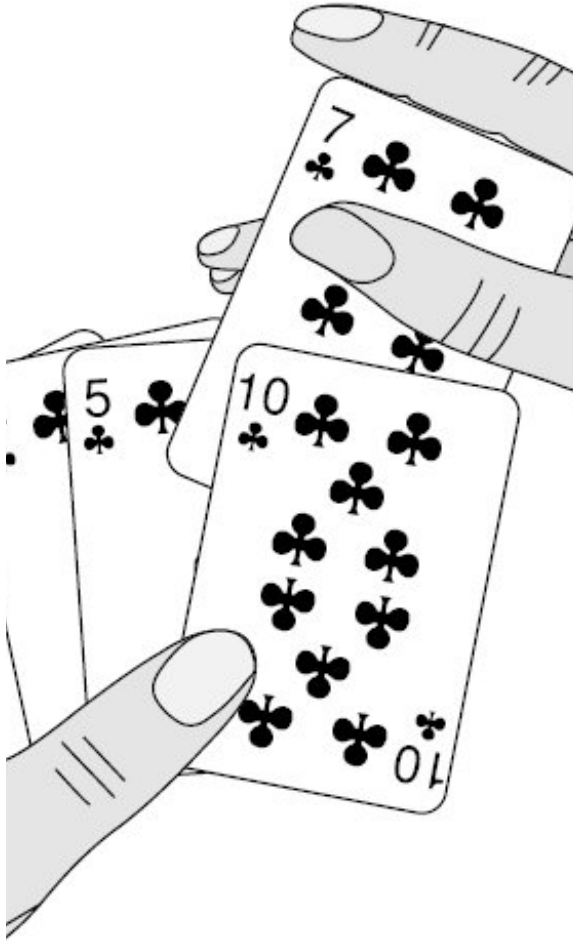
- Un site recommandé:

https://interstices.info/jcms/c_6973/les-algorithmes-de-tri

TRI PAR INSERTION

TRI PAR INSERTION - INTUITION

- **Jeu de cartes**
  - On considère le 2e élément du tableau: si il est plus petit que le 1er on les permute
  - La première partie du tableau (deux premiers éléments) est maintenant triée. On considère le 3e élément et on chercher quelle devrait être sa position dans la partie triée. Si c'est au début, on décale les éléments 1 et 2 et on positionne le 3e élément en premier. Si c'est en 2e position, on décale seulement le 2e, etc.
  - On recommence on considérant le 4e élément.
  - L'algorithme se termine lorsqu'on aura considéré le dernier élément.



Tri par insertion

10	3	78	8	10	145	55

On commence par considérer la 2ieme case, on supposant que la partie droite du tableau est triée.

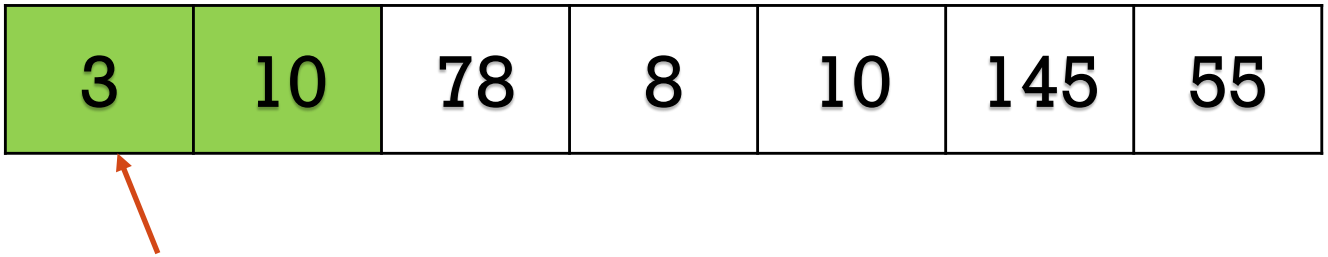
10	3	78	8	10	145	55

On décale tous les éléments à gauche jusqu'à tomber sur une valeur qui est inférieures à la case considérée ou qu'on atteint le début du tableau.

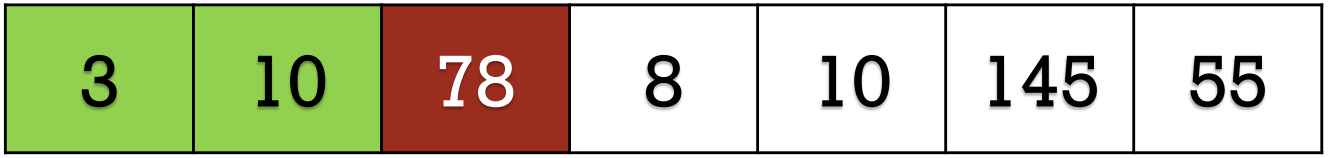
10	3	78	8	10	145	55

	10	78	8	10	145	55

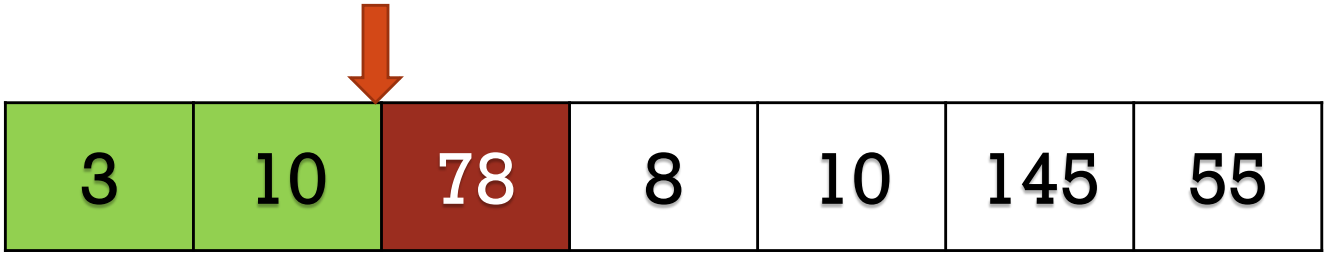
Tri par insertion



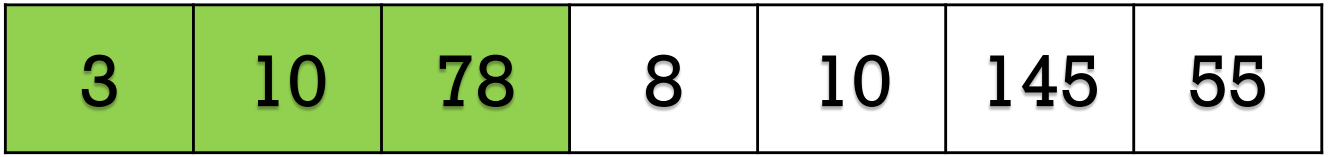
On insère l'élément considéré à la place de la dernière case décalée. On dispose maintenant de deux cases triées (en vert)



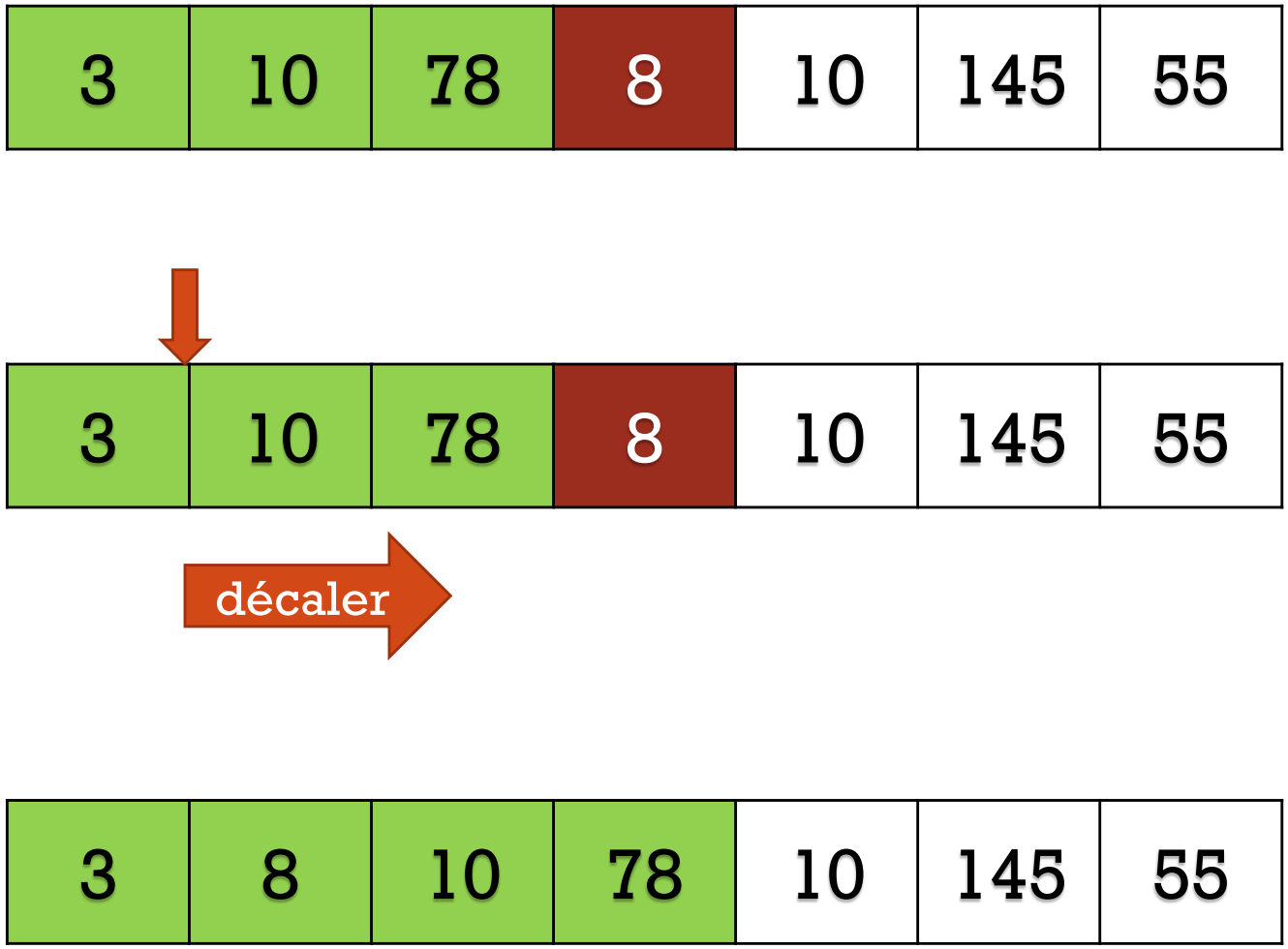
On recommence en considérant maintenant la 3ième case, que l'on va insérer dans la partie triée (verte)



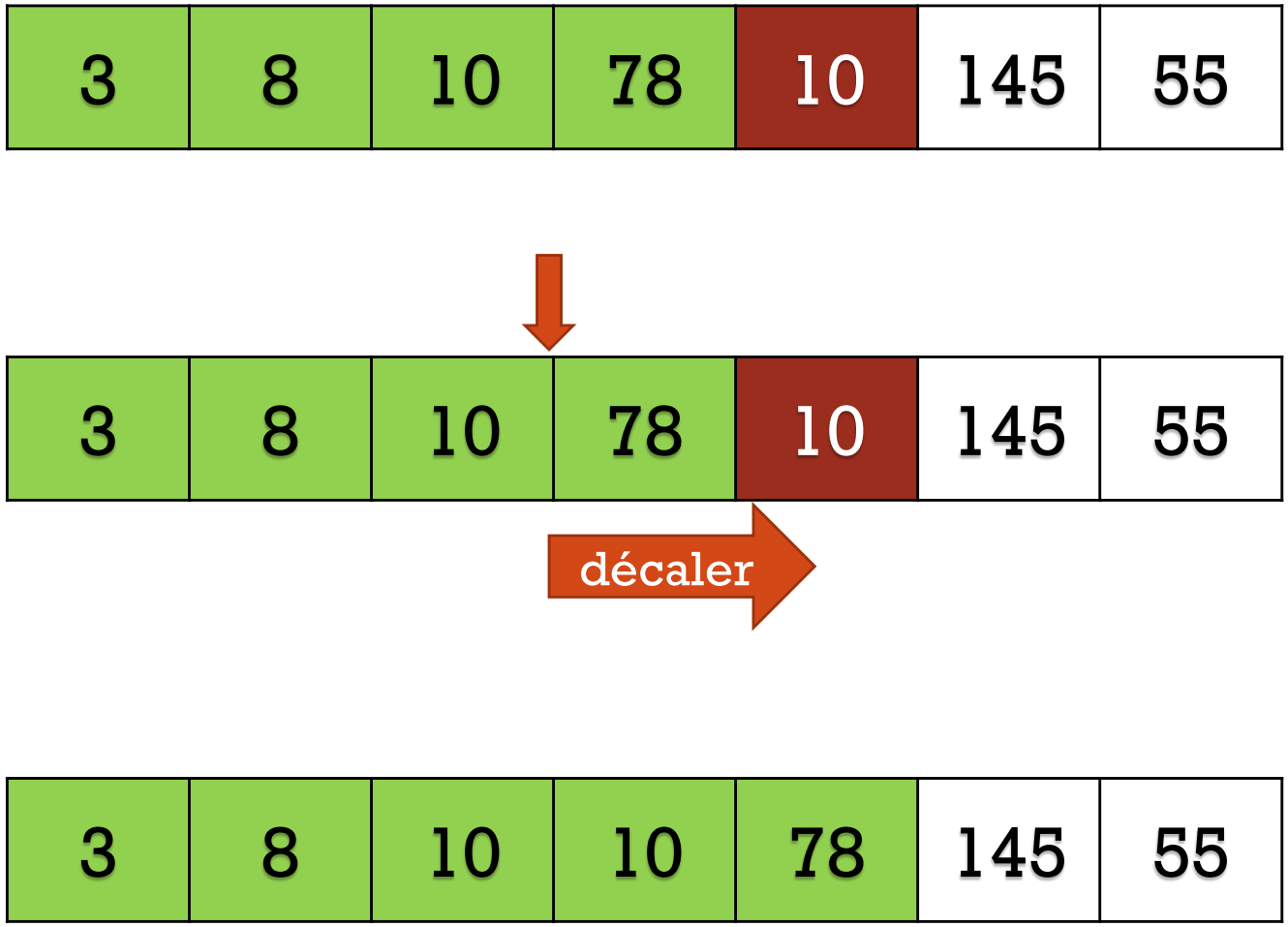
Rien à décaler car 10 est déjà inférieur à 78.



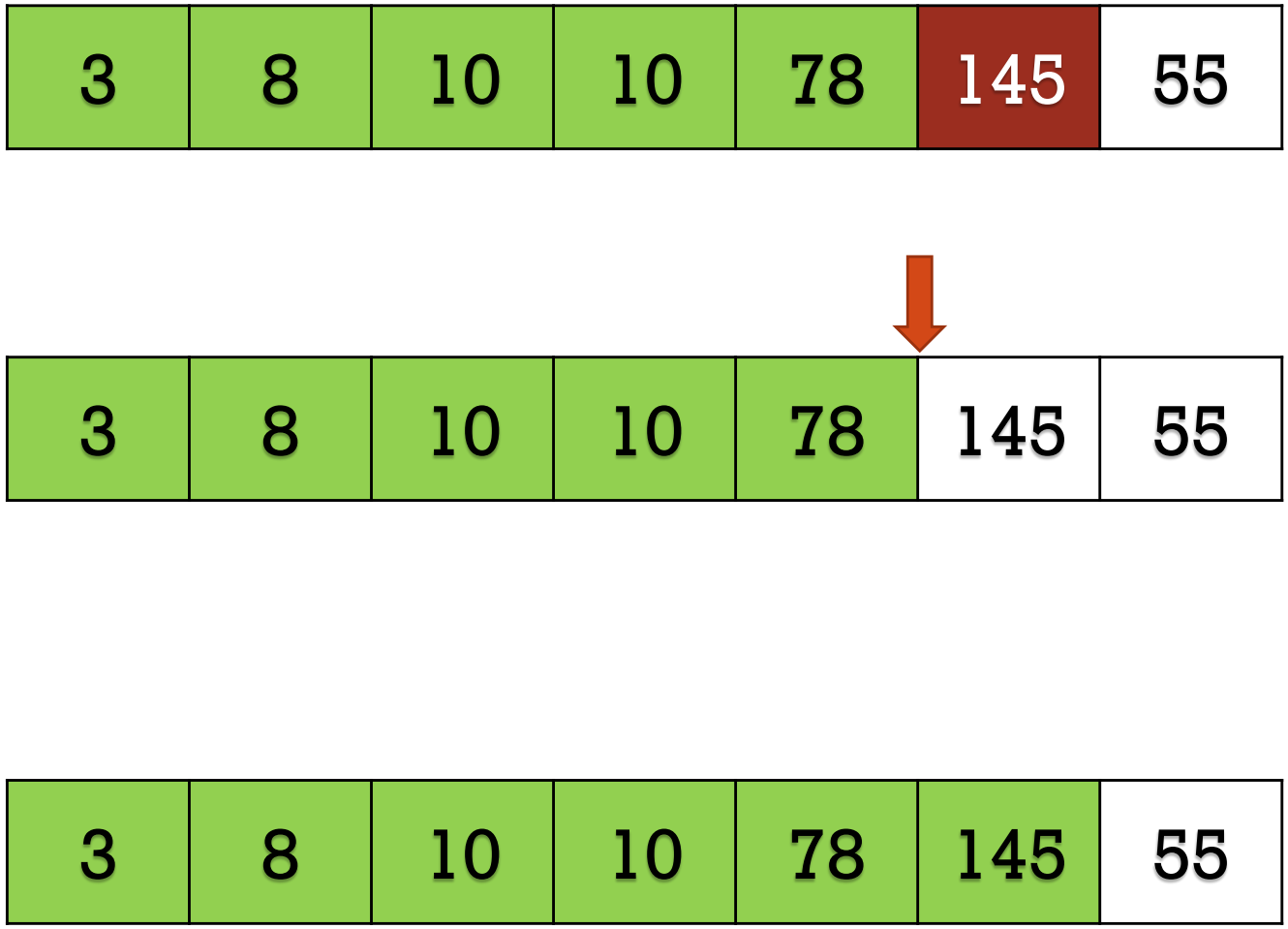
Tri par insertion



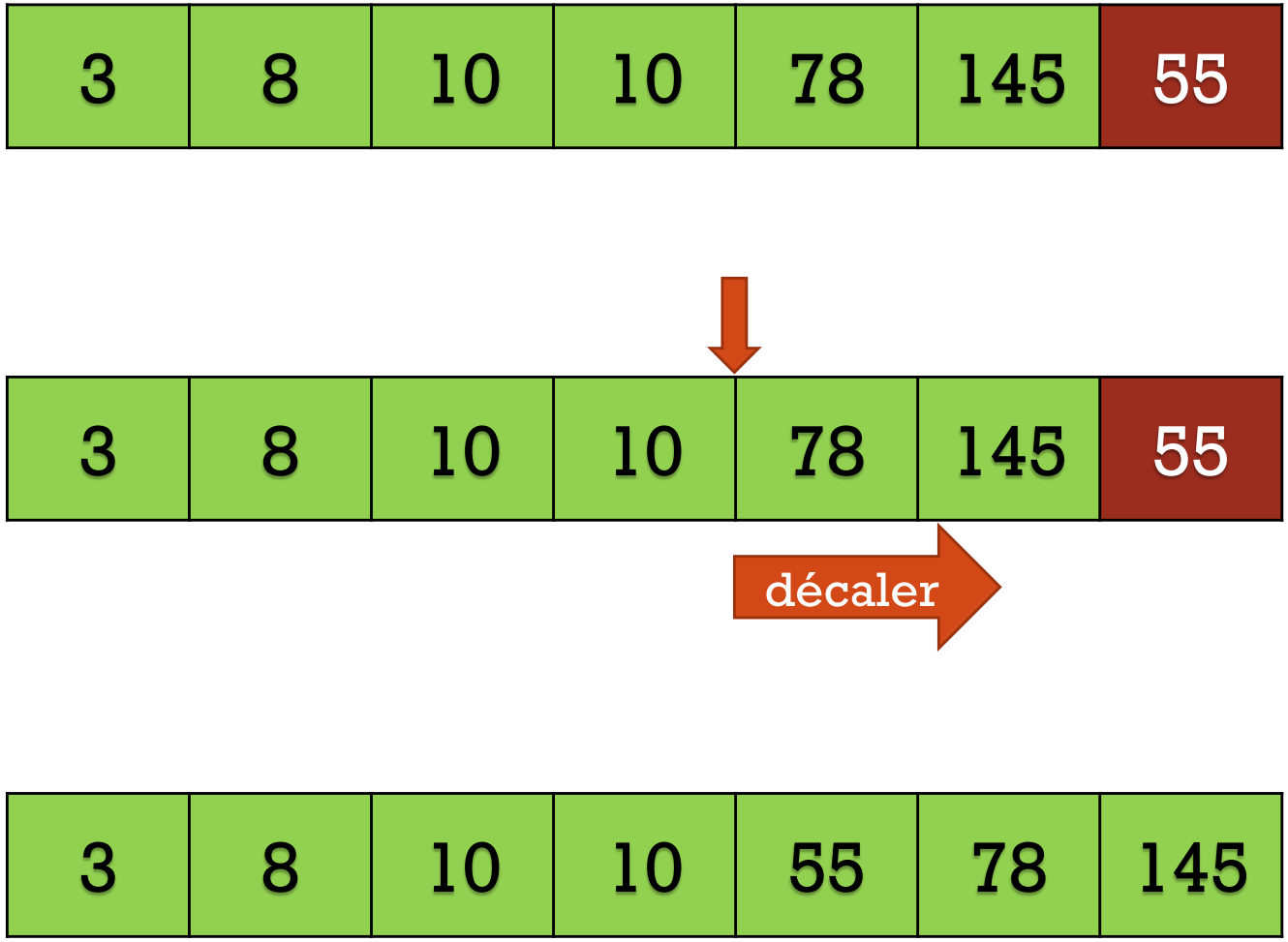
Tri par insertion



Tri par insertion



Tri par insertion



TRI PAR INSERTION - ALGORITHME

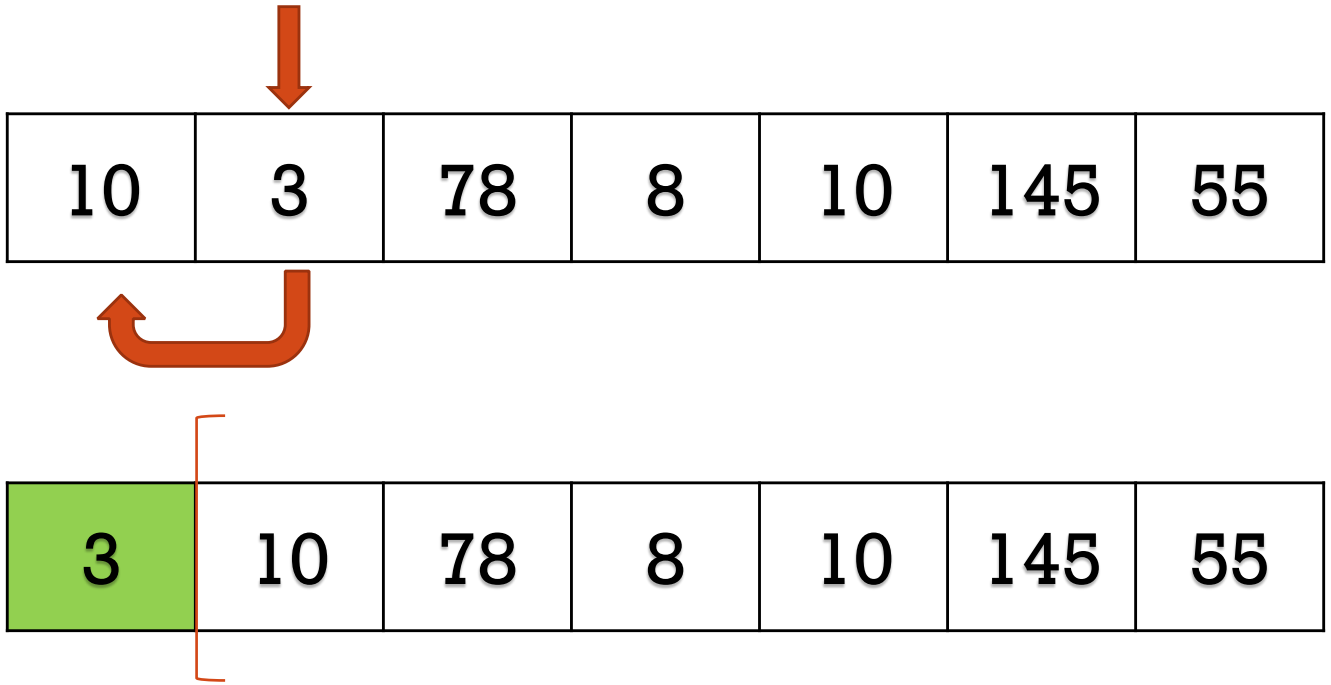
```
Variable tab[] (entier)
tab = [44, 10, 20, 55, 10, 24, 35]

Pour i De 1 À taille(tab)-1 Faire
      x ← tab[i]
      j ← i
      Tant Que j > 0 ET tab[j - 1] > x Faire
            tab[j] ← tab[j - 1]
            j ← j - 1
      Fin Tant Que
      tab[j] ← x
Fin Pour
```

TRI PAR SÉLECTION

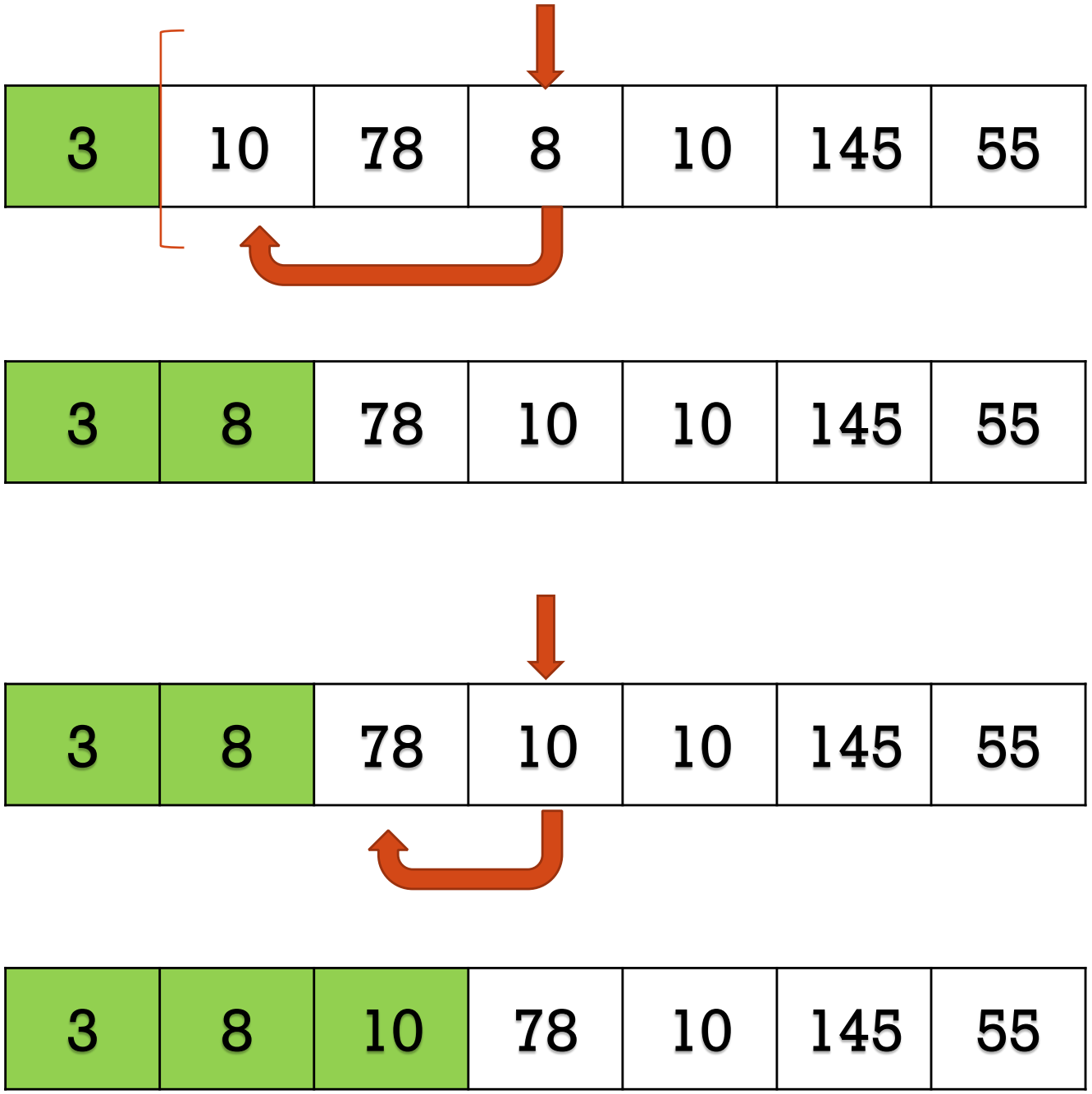
TRI PAR SÉLECTION - INTUITION

- Chercher l'élément le plus petit du tableau et l'échanger avec l'élément à la case 0.

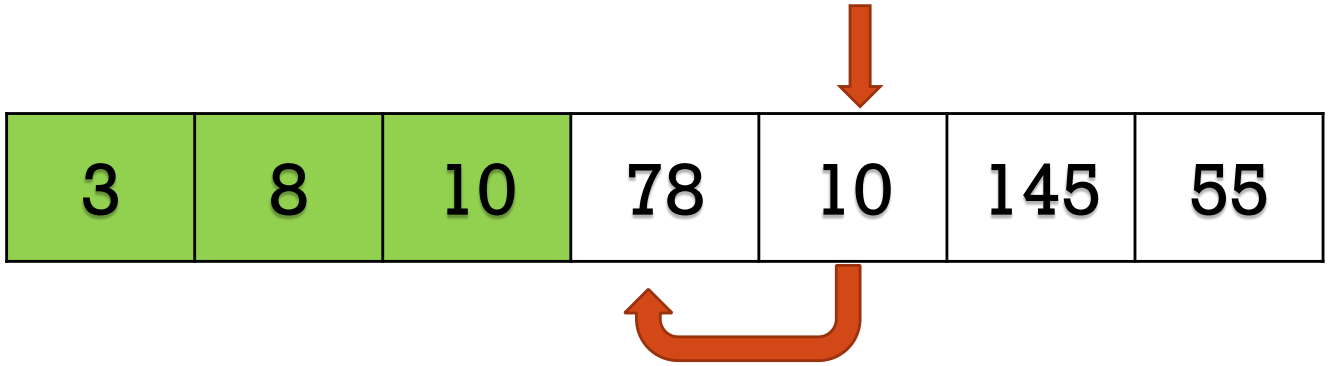


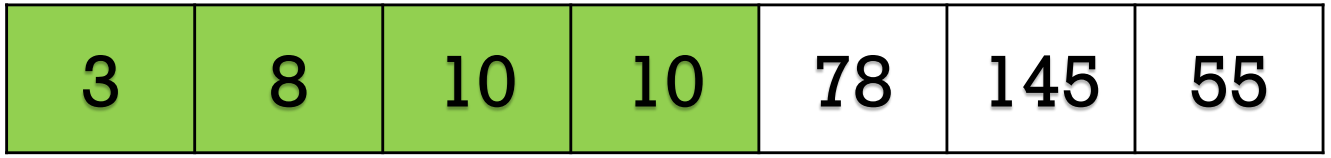
Le début du tableau est maintenant trié, on procède de la même façon avec le reste du tableau.

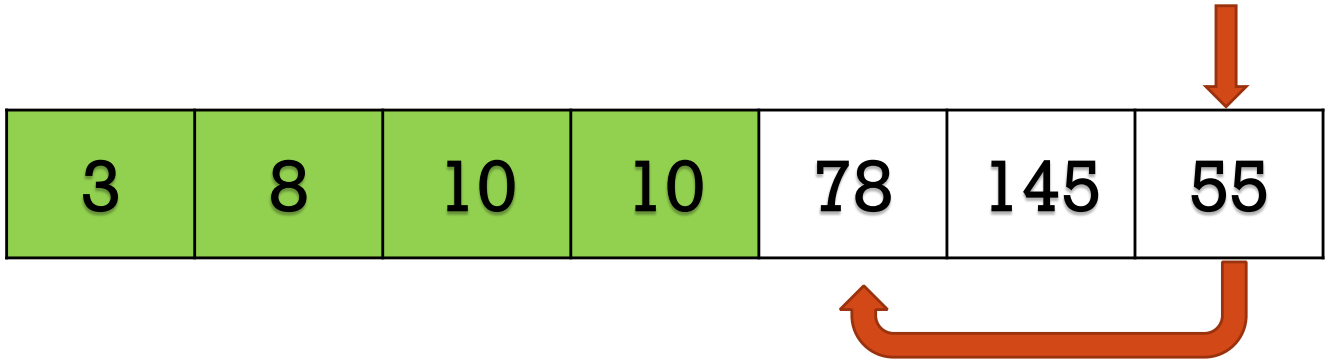
Tri par sélection

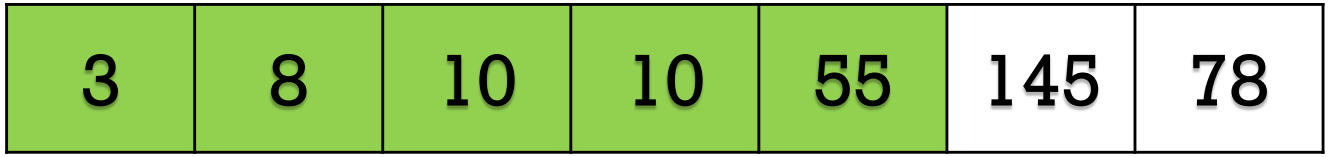


Tri par sélection

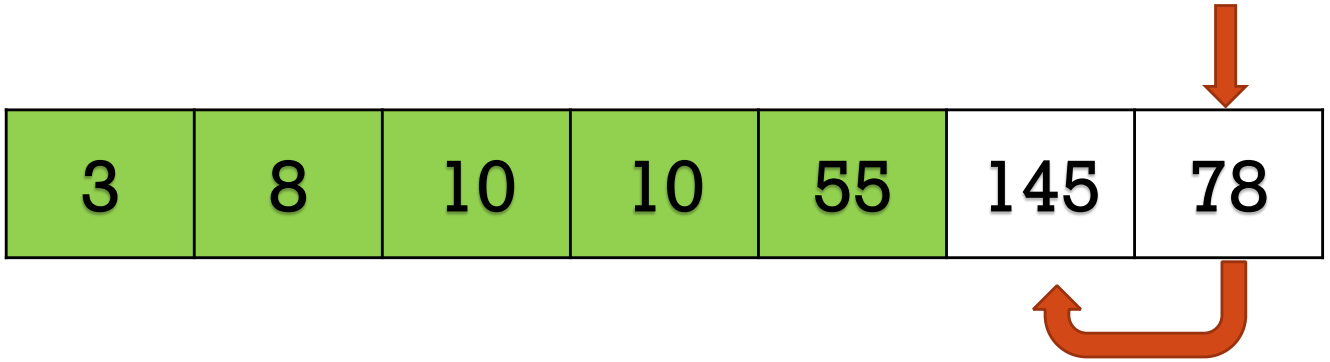


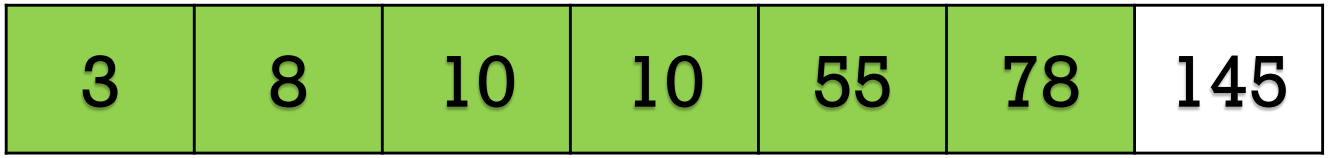




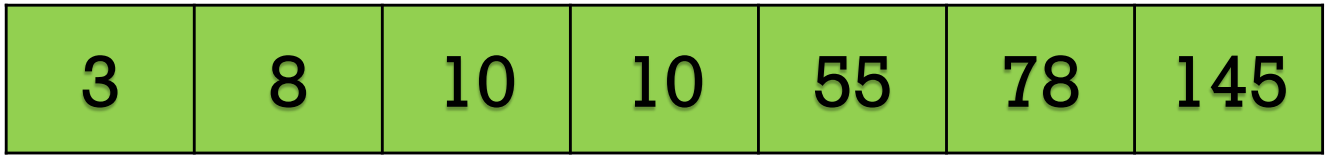


Tri par sélection





Il ne reste plus qu'un seul élément dans le sous-tableau à trier. L'algorithme est terminé et le tableau global est trié.



TRI PAR SÉLECTION - ALGORITHME

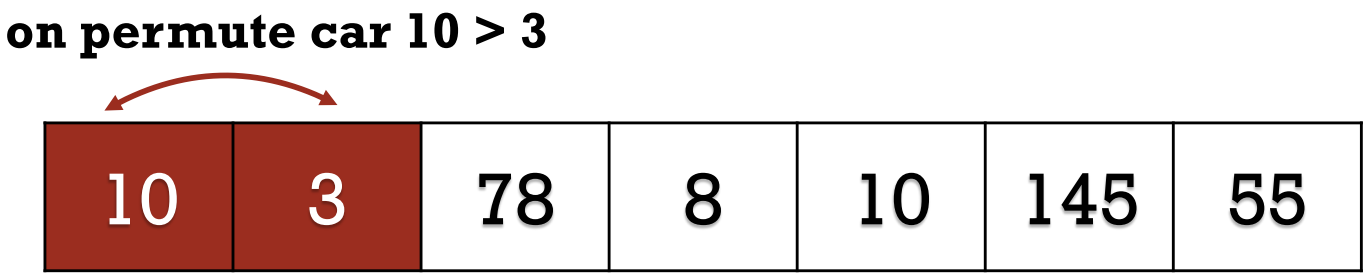
```
Variable tab[7] (entier)
tab = [44, 10, 20, 55, 10, 24, 35]

POUR i DE 0 À taille(tab)-1
    indice_min ← i
    POUR j DE i+1 À taille(tab)-1
        SI tab[j] < tab[indice_min] ALORS
            indice_min ← j
        FIN SI
    FIN POUR
    SI indice_min ≠ i ALORS
        échanger tab[i] et tab[indice_min ]
    FIN SI
FIN POUR
```

TRI À BULLES

TRI À BULLES - INTUITION

- Nous voulons pousser les plus grandes valeurs vers la fin du tableau.
- Nous considérons les cases successives, 2-à-2 en partant de la gauche - si la case de gauche est plus grande que la case de droite, on permute les valeurs, sinon on ne fait rien.



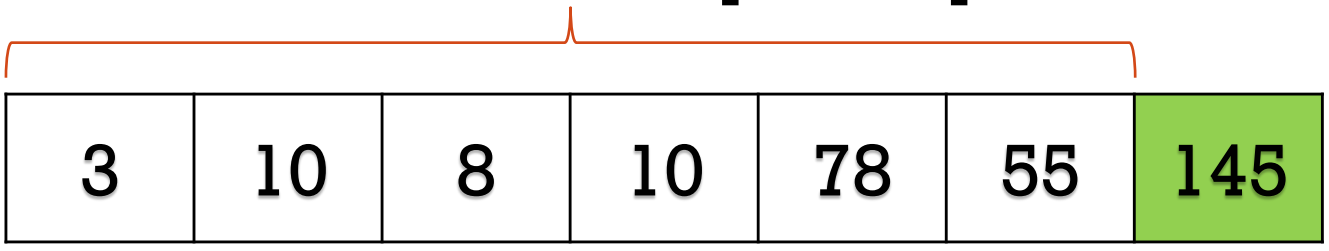
TRI À BULLES - INTUITION

- Quand on est arrivé à la fin du tableau (on a considéré toutes les valeurs 2-à-2), la valeur la plus grande est en queue de tableau.

3	10	8	10	78	55	145

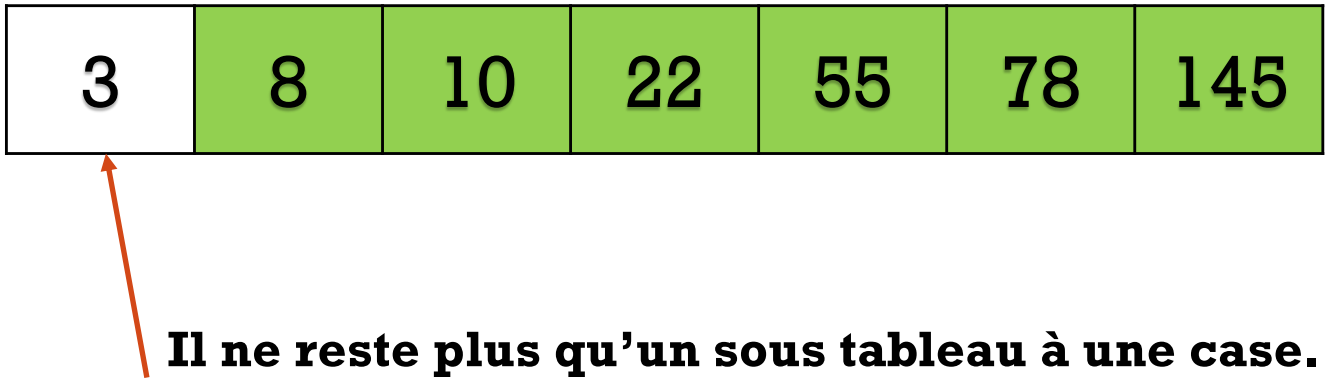
- On considère la dernière valeur comme figée, et on recommence le processus en considérant le tableau amputé de sa dernière case.

On recommence en ne considérant que cette portion du tableau



TRI À BULLES - INTUITION

- On recommence jusqu'à ce que le tableau considéré ne contienne plus qu'une seule case.
- Le tableau obtenu est alors trié.



TRI À BULLES - ALGORITHME

```
Variable monTableau[] (entier)
Variable copie (entier)

monTableau = [44, 10, 20, 55, 10, 24, 35]
Pour i De taille(monTableau) à 1 Faire
      Pour j De 0 à i-1 Faire
          Si monTableau(j)>monTableau(j+1) Alors
              Variable tmp (entier)
              tmp ← monTableau(j)
              monTableau(j) ← monTableau(j+1)
              monTableau(j+1) ← tmp
          Fin Si
      Fin Pour
Fin Pour
```

Tri à bulles – Itération 1

ÉTS

Tri à bulles – Itération 2

Tri à bulles – Itération 3

no -swap

3	8	10	10	55	78	145

no -swap

3	8	10	10	55	78	145

no -swap

3	8	10	10	55	78	145

no -swap

3	8	10	10	55	78	145

3	8	10	22	55	78	145

Tri à bulles – Itération 4

no -swap

no -swap

no -swap

Tri à bulles – Itération 5

no -swap

no -swap

Tri à bulles – Itération 6

3	8	10	22	55	78	145

3	8	10	22	55	78	145

Résultat d'exécution de l'algorithme:

3	8	10	22	55	78	145

L'INTRA 2 SERA DANS 2 SEMAINES!