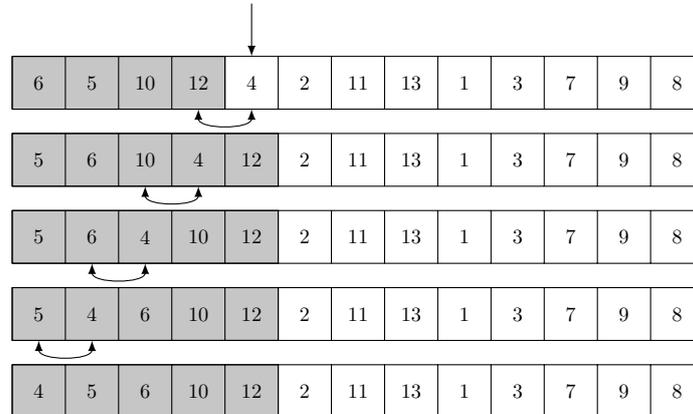


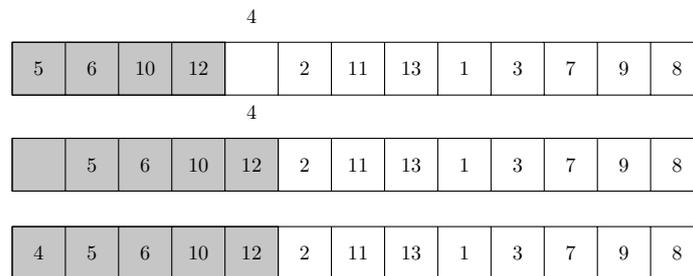
**Exercice 1 :** En considérant que la durée d'exécution du tri par sélection est proportionnelle à  $n^2$  et qu'il prend 6,8 secondes pour trier 16000 éléments, combien de temps prendra-t-il pour trier 1 million d'éléments?

**Exercice 2 :** Le tri par insertion fait remonter l'élément en cours jusqu'à sa place, en effectuant des échanges de proche en proche.



Une autre méthode consiste à :

- stocker l'élément en cours,
- décaler vers la droite tous les éléments supérieurs,
- placer l'élément en cours à la bonne place.



Écrire la fonction `tri_insertion(tab: list) → None` qui implémente ce nouvel algorithme.

**Exercice 3 :** On considère le tableau :

```
1 [3, 4, 1, 7, 2]
```

Donner l'état du tableau après chaque itération quand on effectue :

1. un tri par sélection,
2. un tri par insertion.

**Exercice 4 :** Pour comparer si deux tableaux de même longueur sont identiques, c'est-à-dire s'ils contiennent les mêmes éléments, une stratégie consiste à d'abord les trier puis comparer chaque élément un à un.

1. Construire la fonction `comparer(tab1: list, tab2: list) → bool` qui compare les éléments des tableaux un à un et renvoie `True` s'ils sont identiques.
2. Utiliser la fonction tri par insertion pour trier les tableaux :
  - [3, 5, 9, 0, 1, 8, 2],
  - [9, 5, 3, 2, 8, 1, 0].



3. Comparer les deux tableaux avec la fonction `comparer`.

**Exercice 5 :** Le tri par insertion effectue un tri en place. Pour ne pas modifier les données initiales, il faut construire un nouveau tableau.

1. Reprendre le tri de l'exercice 2 et écrire la fonction `tri_insertion(tab: list) → list` qui crée un nouveau tableau trié à partir des éléments de `tab`. La fonction suivra l'algorithme suivant :
  - parcourir `tab` et copier l'élément de rang `i` à la fin du nouveau tableau,
  - insérer cet élément à la bonne position dans le nouveau tableau.
2. Construire par compréhension un tableau de dix éléments aléatoires compris entre 0 et 100.
3. Tester la fonction de tri sur ce tableau.

**Exercice 6 :**

1. Adapter le tri par insertion vu en cours pour trier le tableau de tuples suivants en fonction du premier élément des tuples :

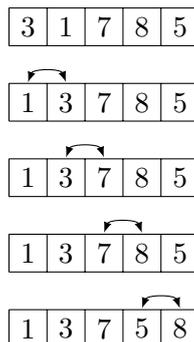
```
1 tab = [(5, "a"), (8, "b"), (1, "e"), (5, "d"), (7, "f"), (8, "c")]
```

2. Un *tri stable* conserve l'ordre relatif des éléments de même valeur. Le tri par insertion est-il stable ?

**Exercice 7 :**

1. Construire par compréhension un tableau de cent éléments aléatoires compris entre 0 et 10.
2. En s'aidant du tri par insertion, écrire la fonction `max_occurrences(tab: list) → int` qui renvoie l'élément le plus présent dans le tableau. Indication : dans un tableau trié les éléments identiques se suivent.

**Exercice 8 :** Le tri à bulles *fait remonter* les plus grands éléments d'un tableau, comme des bulles d'air qui remonteraient à la surface d'un liquide. Le code 1 présente la première itération qui permet de propager le plus grand élément en dernière place du tableau. Il suffit ensuite d'effectuer une deuxième itération pour propager le deuxième plus grand élément en avant-dernière position.



Code 1 – Tri à bulles - première itération de la boucle externe

1. Écrire la fonction `echanger(tab: list, i: int, j: int) → None` qui permute les éléments d'indices `i` et `j` de `tab`.
2. Écrire la fonction `tri_bulles(tab: list) → None` qui implémente le tri à bulles. Cette fonction utilisera la fonction `echanger`.
3. Construire par compréhension un tableau de vingt éléments d'entiers aléatoires compris entre 0 et 1000.
4. Tester la fonction de tri sur ce tableau.