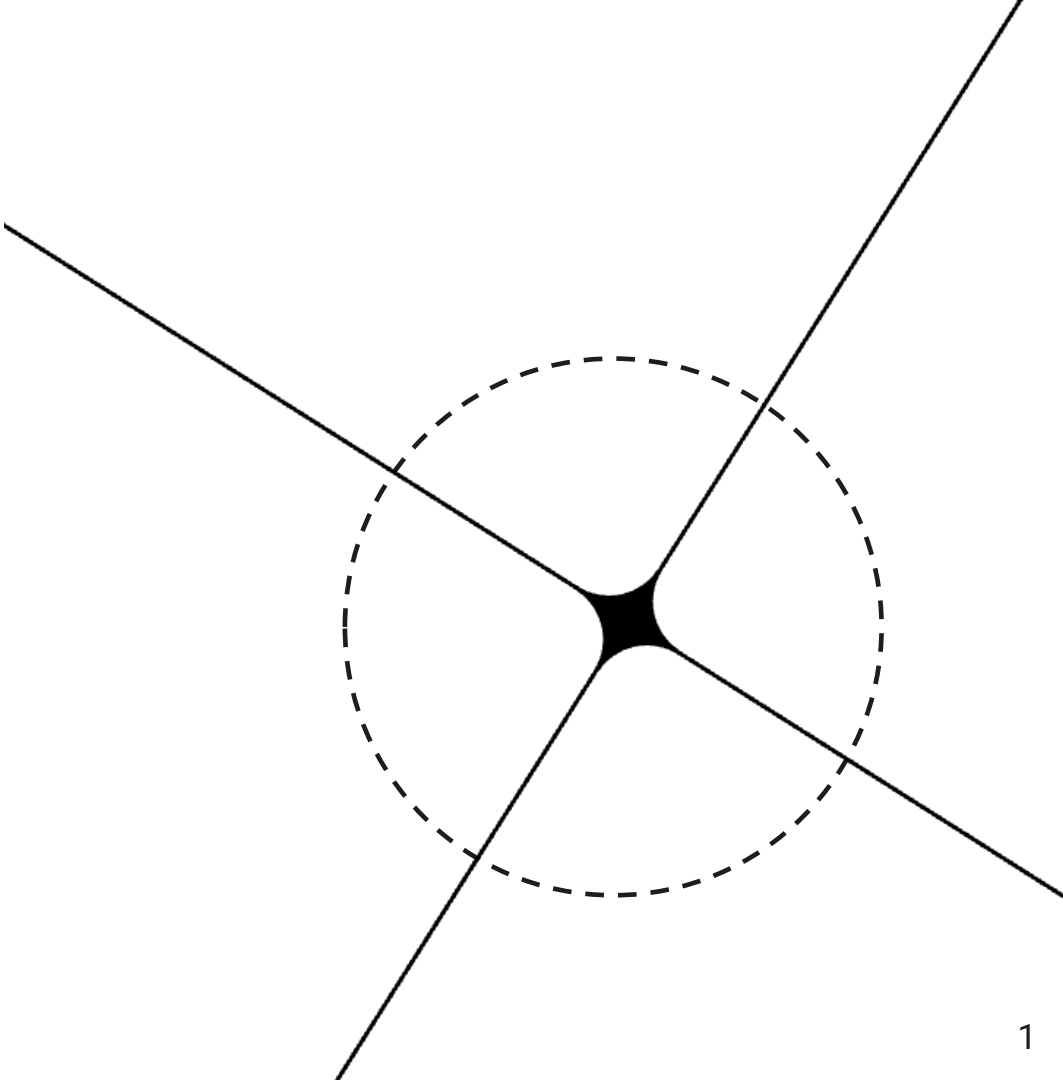
INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

Introduction TP 1



Pourquoi utiliser un IDE

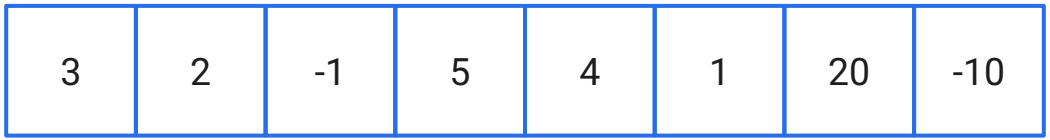
- Les IDE (Environnements de Développement Intégré) simplifient le développement.
- Comparaison avec des éditeurs de texte simples comme gedit, emacs, ou vim :
  - Avantages de l'IDE :
    - Complétion automatique du code
    - Débogage intégré
    - Gestion de projet simplifiée
    - Support pour la gestion des dépendances
    - Visualisation des erreurs et avertissements en temps réel
  - Inconvénients des éditeurs de texte :
    - Moins d'outils intégrés
    - Moins de support pour le débogage et la gestion de projet
- Importance de maîtriser son IDE : Accélère le développement et réduit les erreurs.
- Exemples d'IDE pour Java :
  - IntelliJ IDEA
  - Eclipse
  - VSCode avec l'extension Java

Algorithmes simples

- **PGCD (Plus Grand Commun Diviseur)** : Calculer le PGCD de deux nombres
  - Itératif, sans méthode
  - Pratique des structures algorithmiques
- **Calculatrice** : Implémentation d'une calculatrice de base avec des opérations simples.
  - Lecture des arguments du programme
- **Histogramme** : Créer un histogramme à partir d'un tableau d'entiers.
  -
- **Sudoku** (complexe) : Validation d'un Sudoku en utilisant une approche itérative ou récursive.
  - Algorithme du retour arrière (backtracking)
- **Palindrome** : Vérifier si une chaîne de caractères est un palindrome (symétrique).
  - Palindrome: mot ou chaîne de caractères qui reste identique en inversant l'ordre des lettres
  - Pratique des méthodes de classe

Tri à bulle

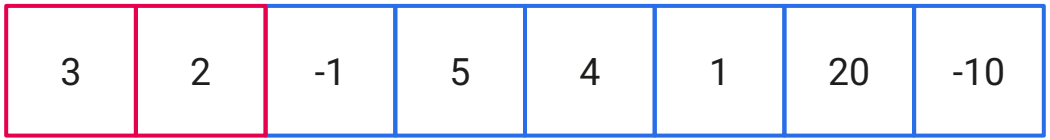
- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

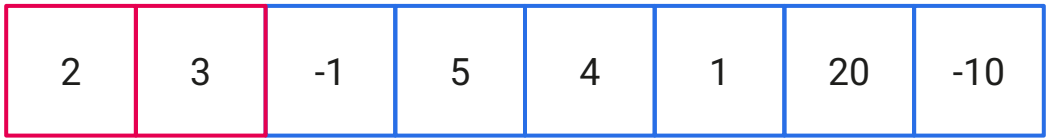
On fait remonter une bulle le long du tableau



Tri à bulle

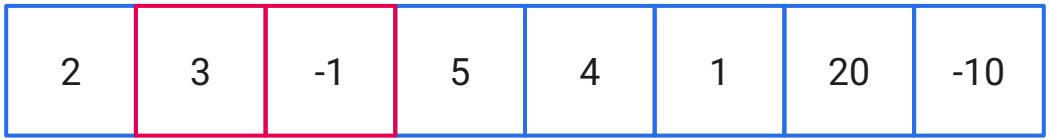
- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Quand nous rencontrons une paire inversé, nous la remettons dans l'ordre



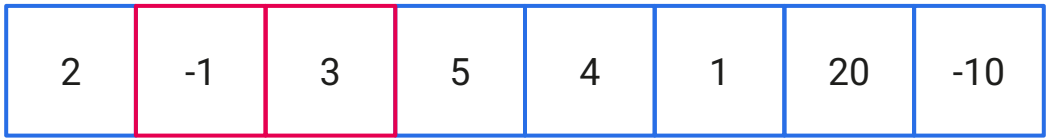
Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



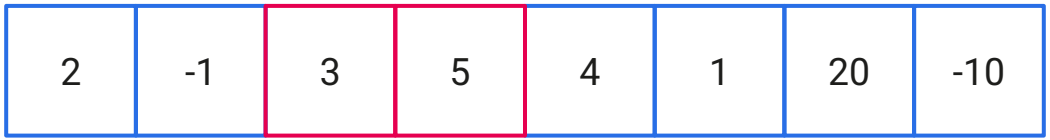
Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Tri à bulle

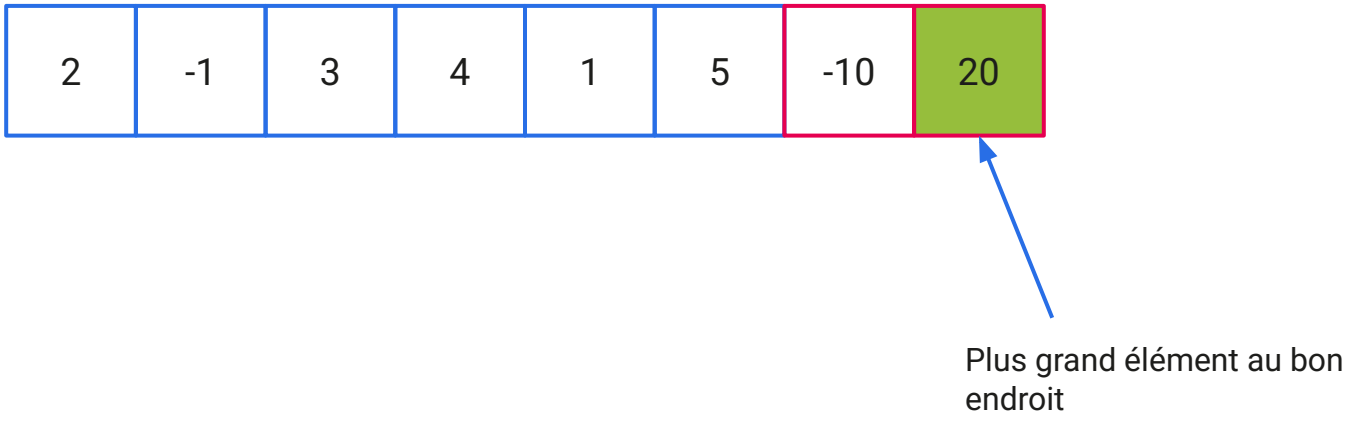
- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

On recommence

Tri à bulle

- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Tri à bulle

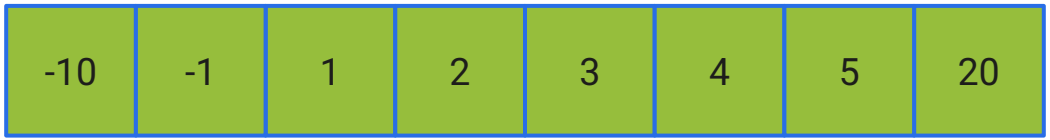
- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Encore…

Tri à bulle

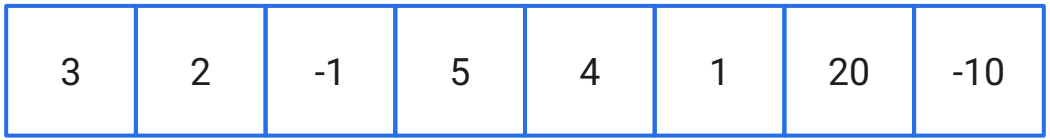
- **Principe** : Comparer des paires d'éléments voisins et les échanger si nécessaire jusqu'à ce que la liste soit triée.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**

Jusqu'à ce que le tableau soit trié (n fois, où n est la taille du tableau)

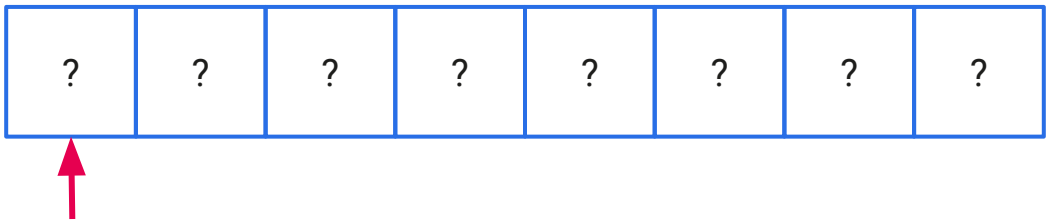


Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



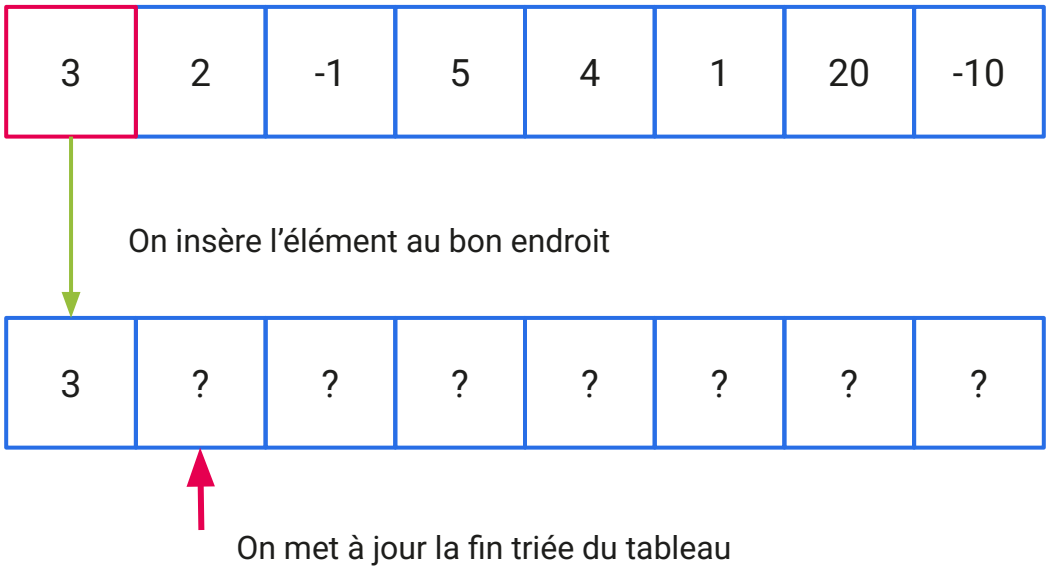
On crée un deuxième tableau qui va contenir le résultat trié.



On se rappelle jusqu'où le tableau est trié

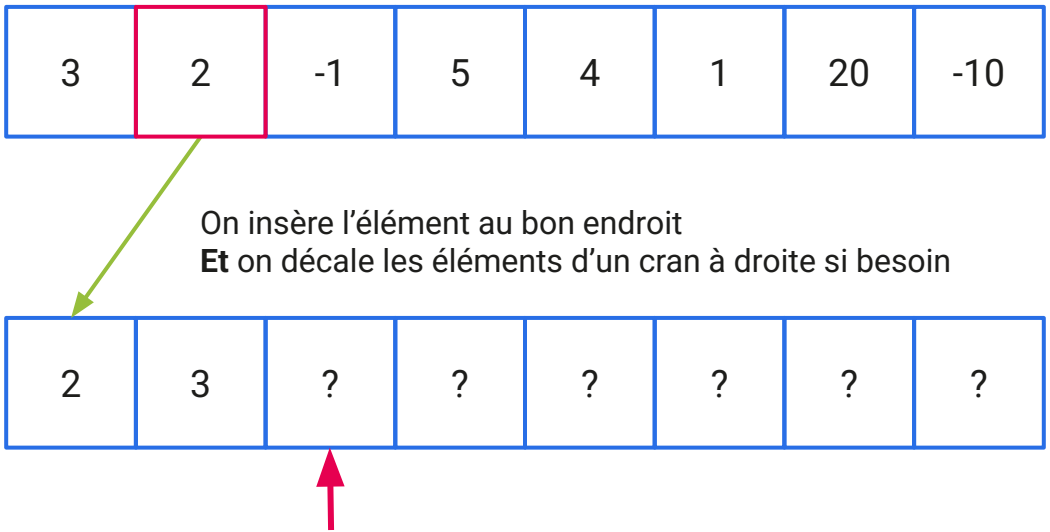
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



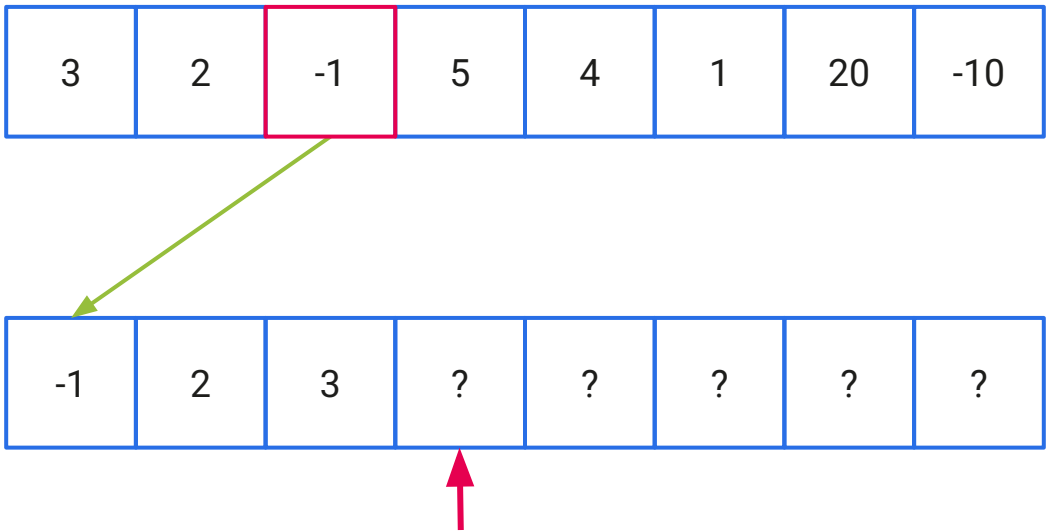
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



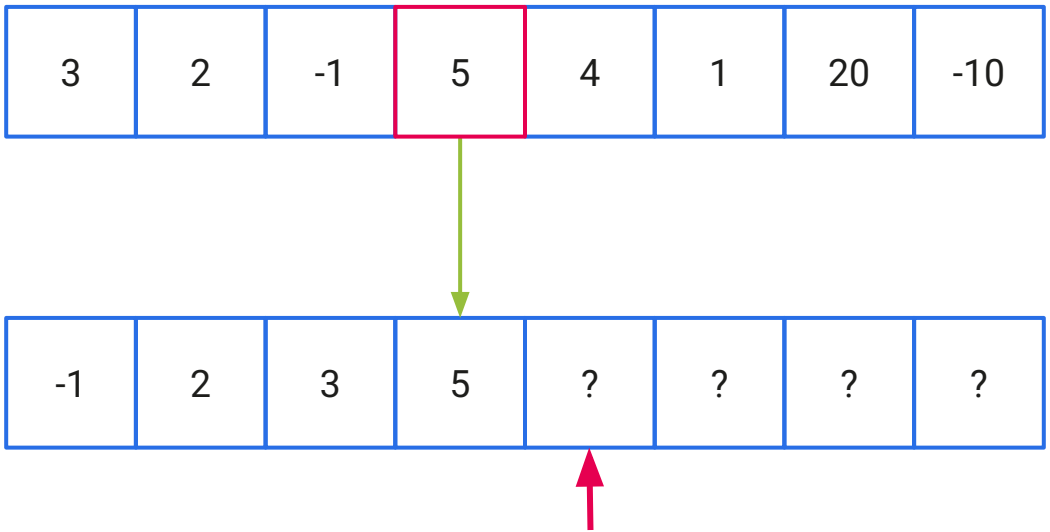
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



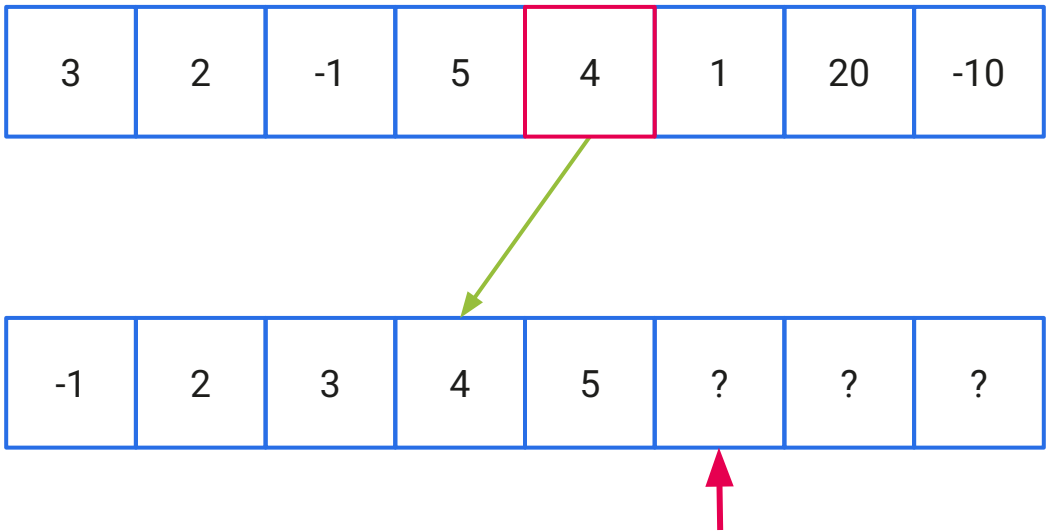
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



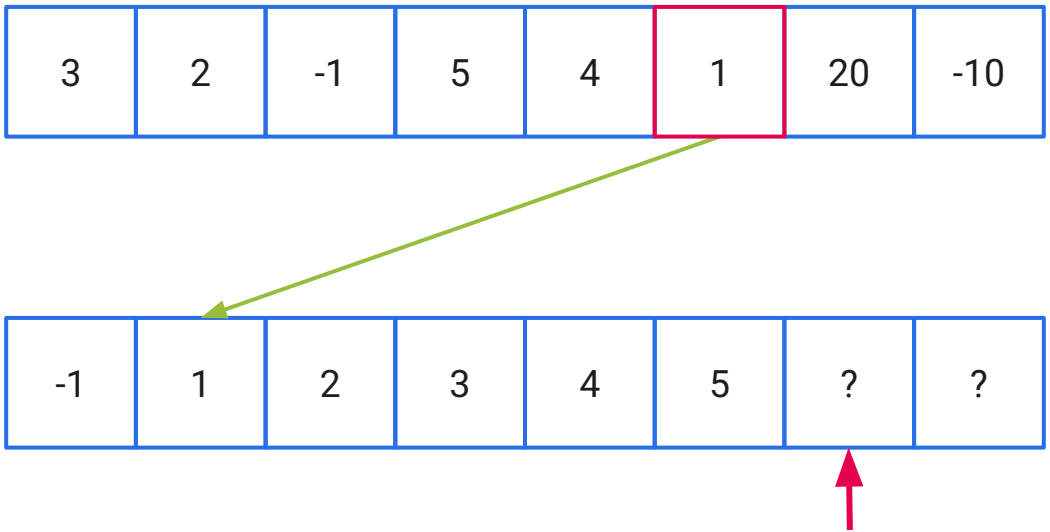
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



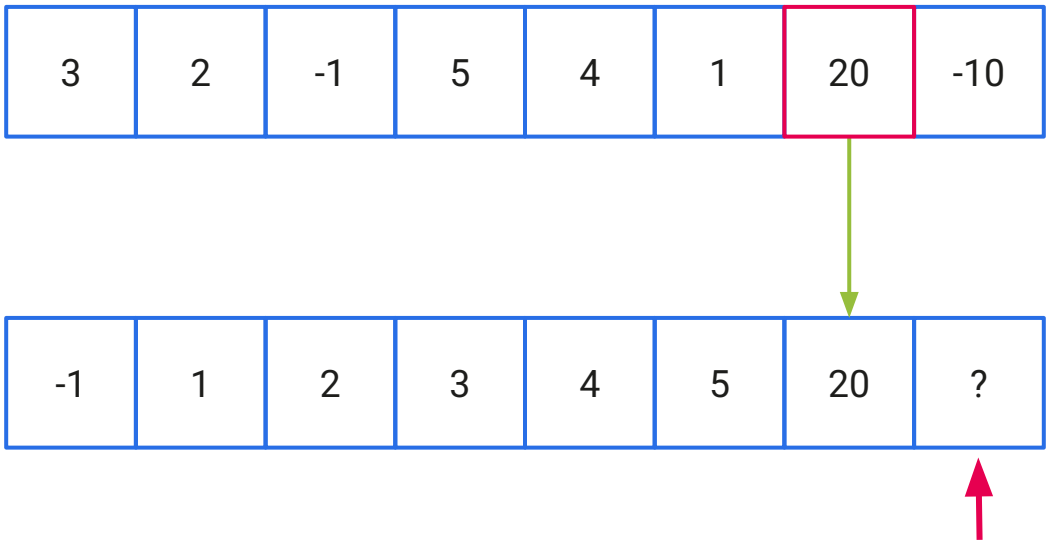
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



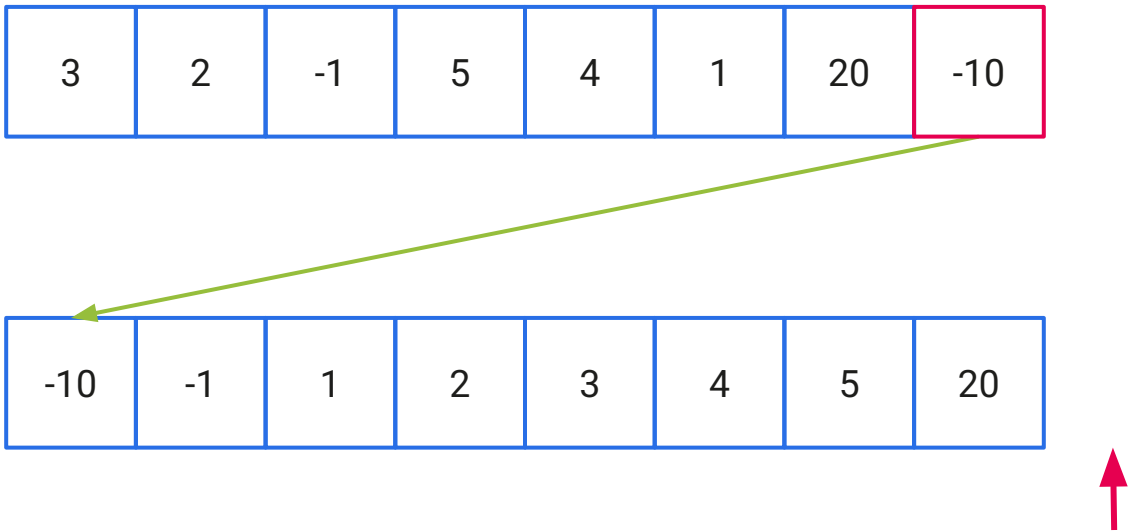
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



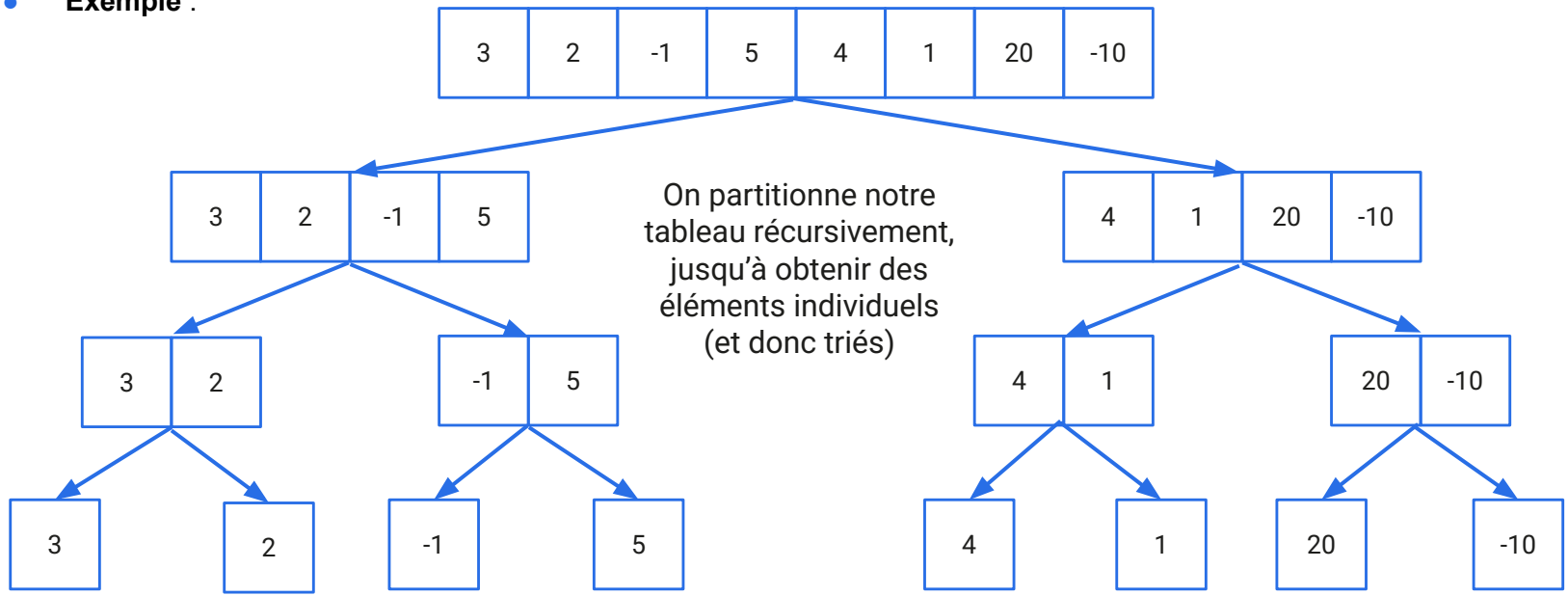
Tri par insertion

- **Principe** : Construire progressivement une sous-liste triée en insérant chaque élément dans sa position correcte.
- **Complexité** : O(n²) dans le pire des cas.
- **Exemple**



Tri par fusion

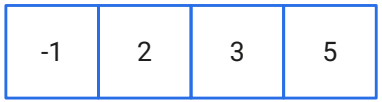
- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :

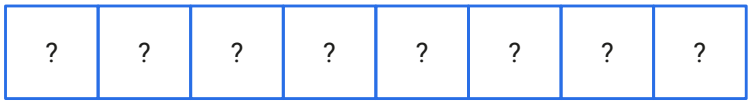


Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :

On utilise une fonction de fusion qui fusionne
intelligemment deux tableaux pour conserver le tri

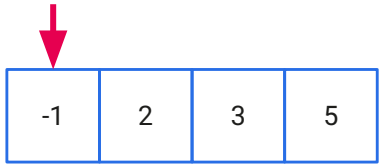


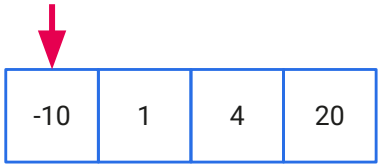


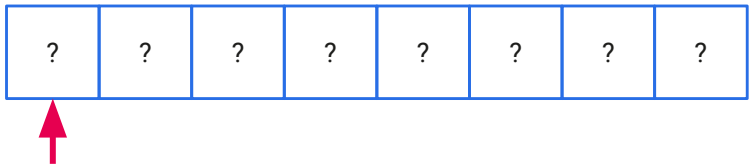
On crée un deuxième tableau qui va contenir le résultat de la fusion

Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



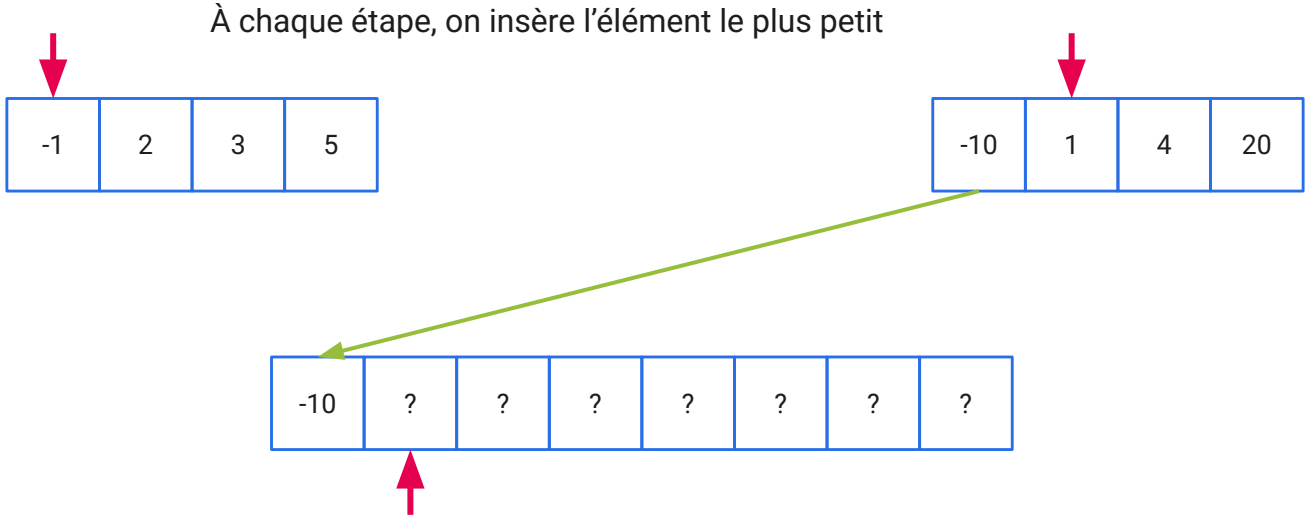




On note où on en est dans chaque tableau

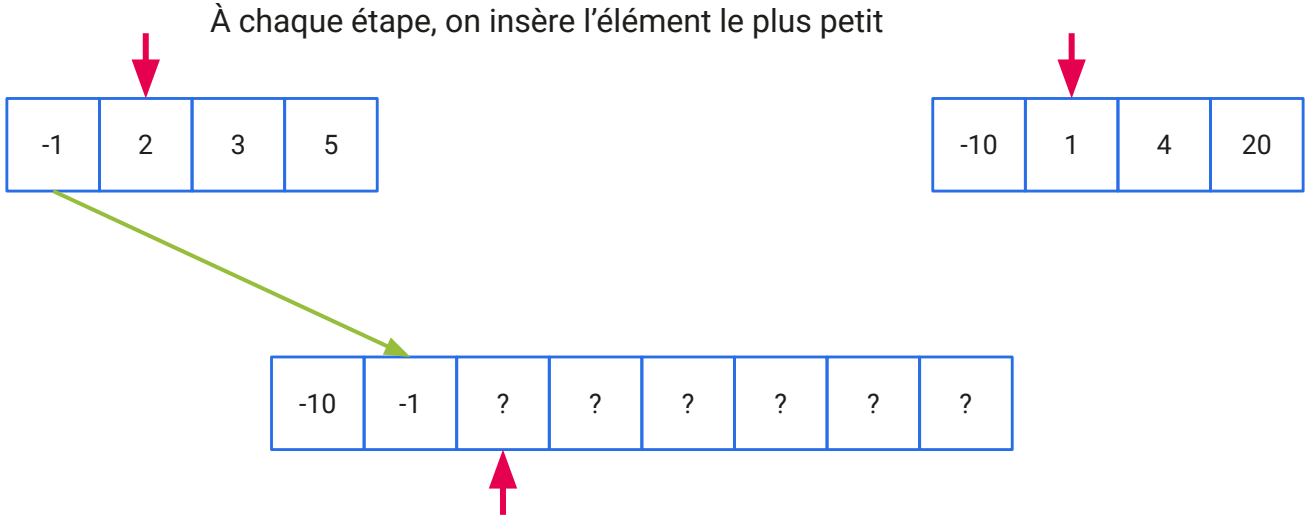
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



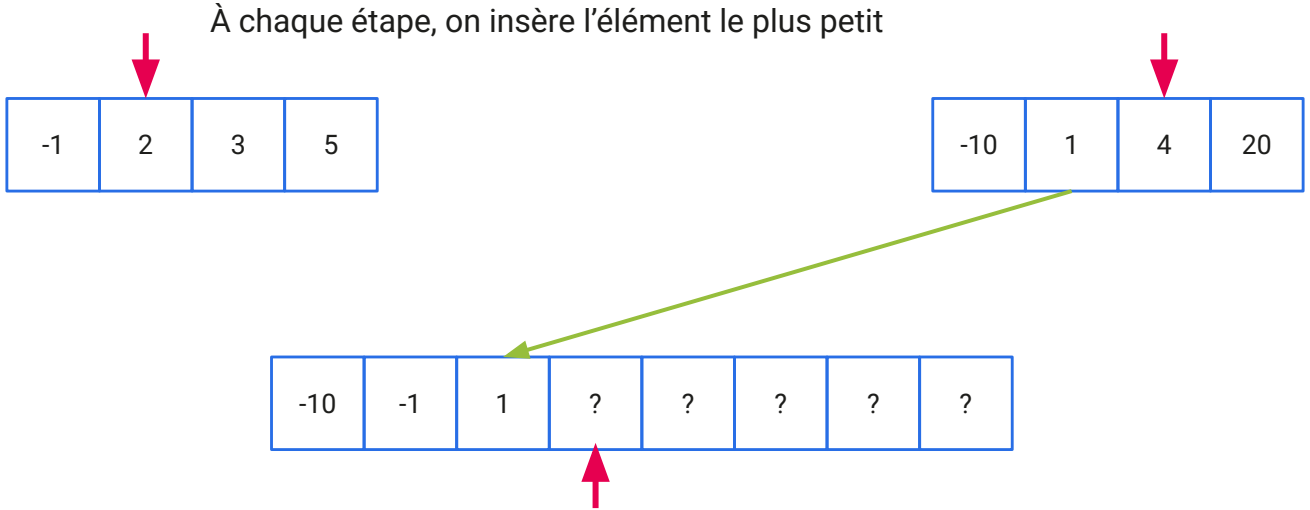
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



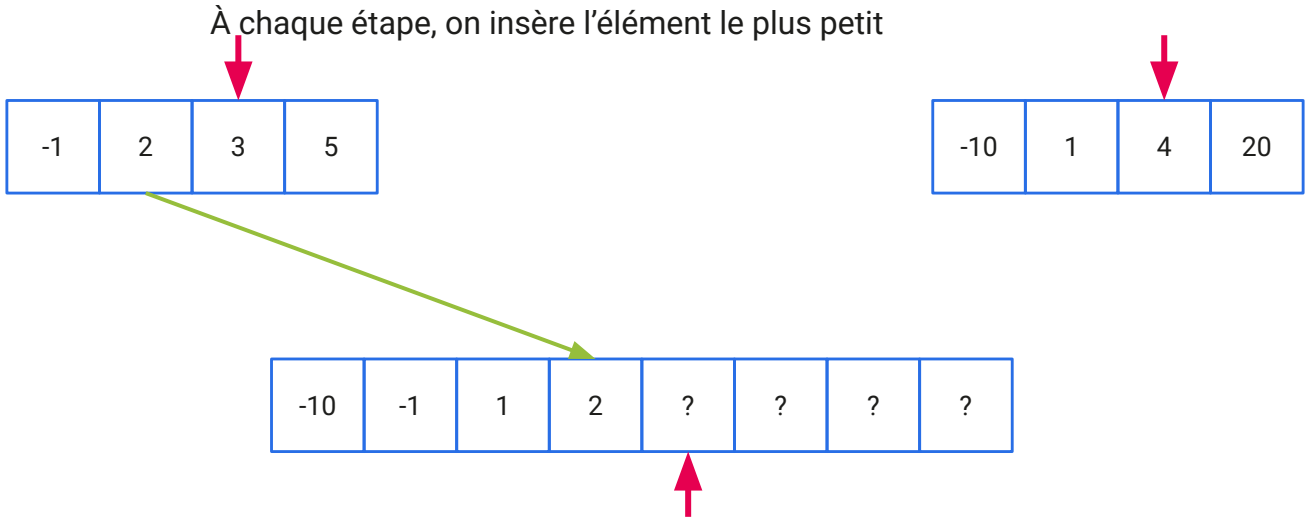
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



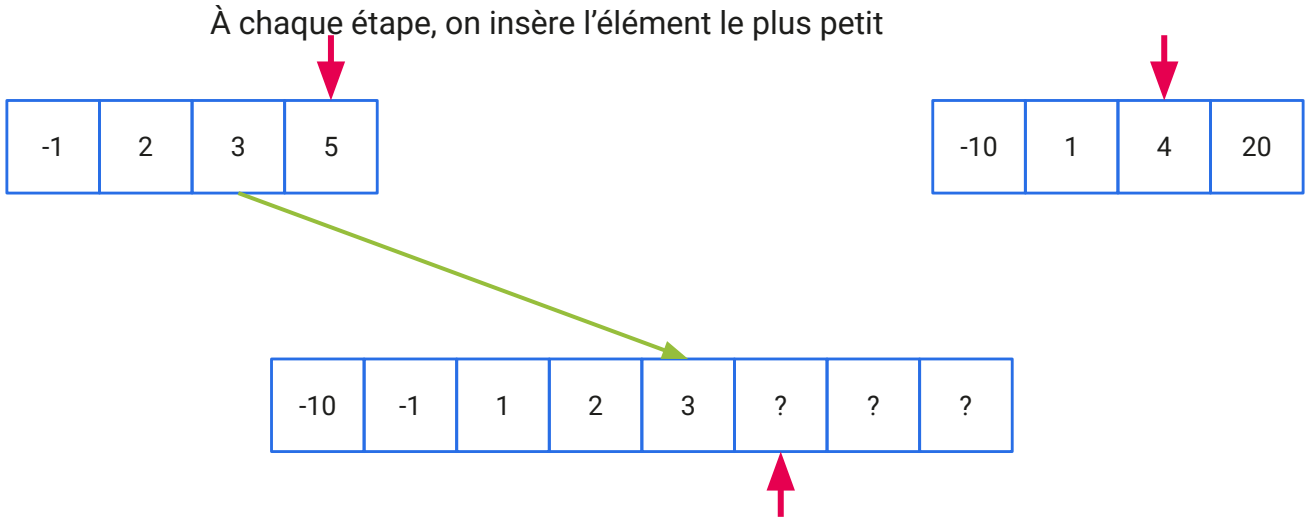
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



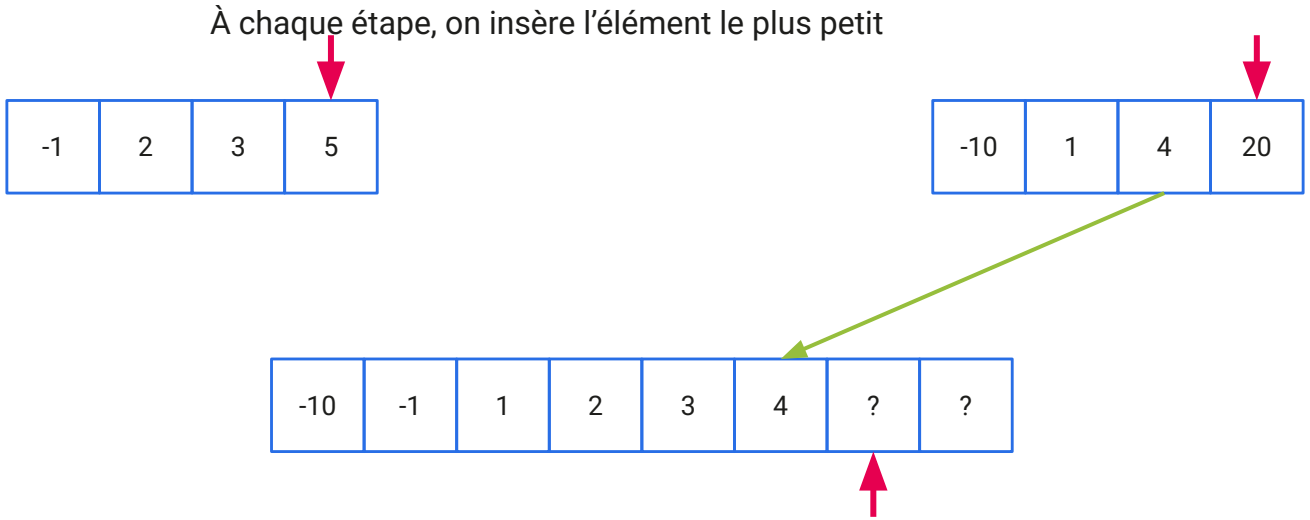
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



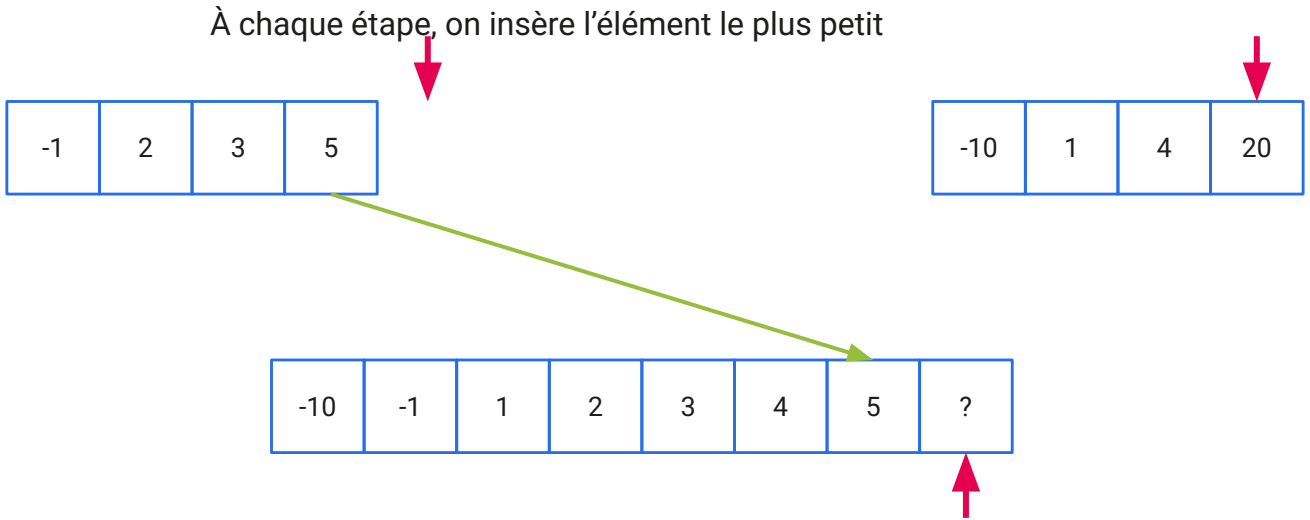
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



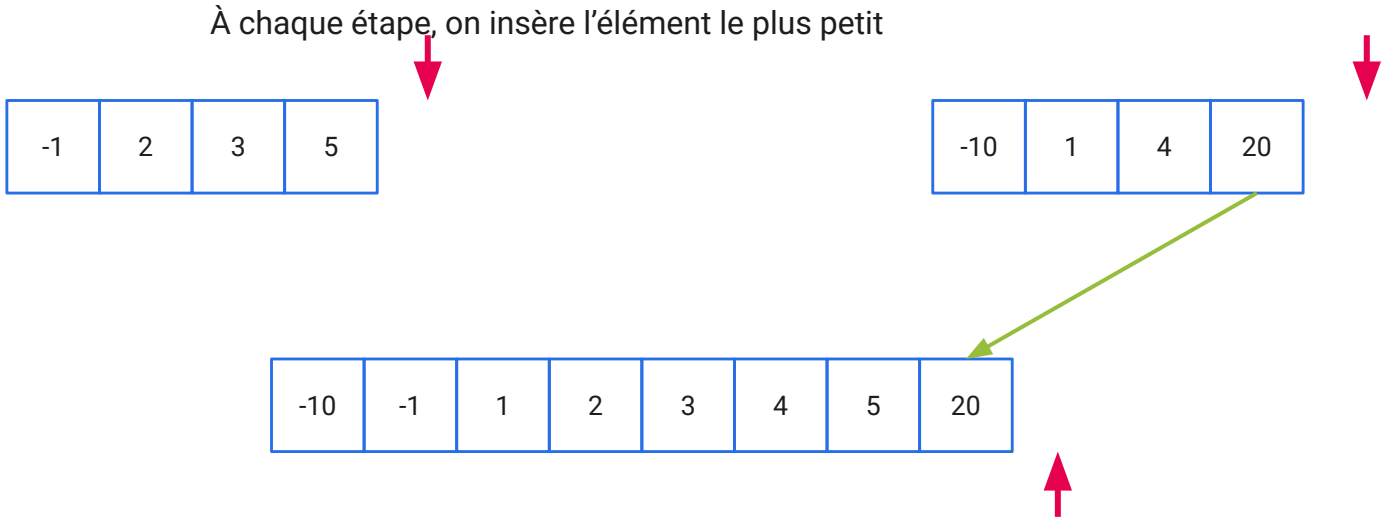
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



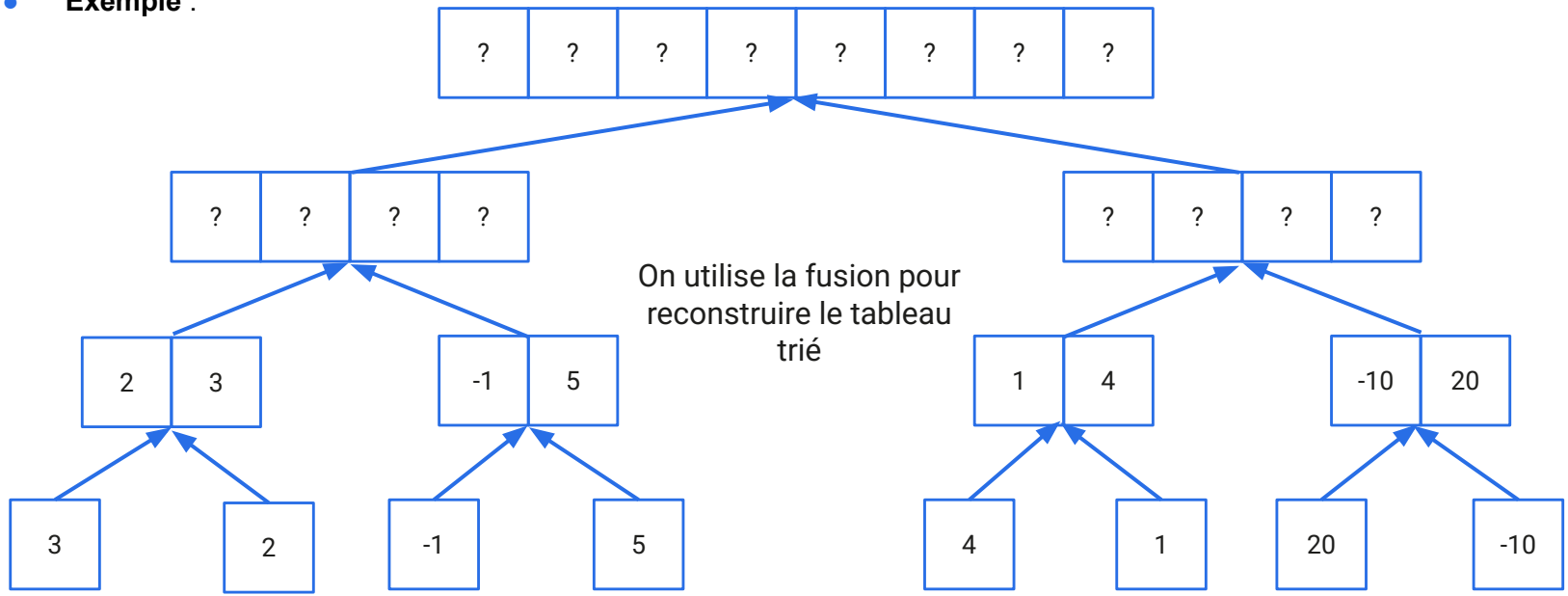
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



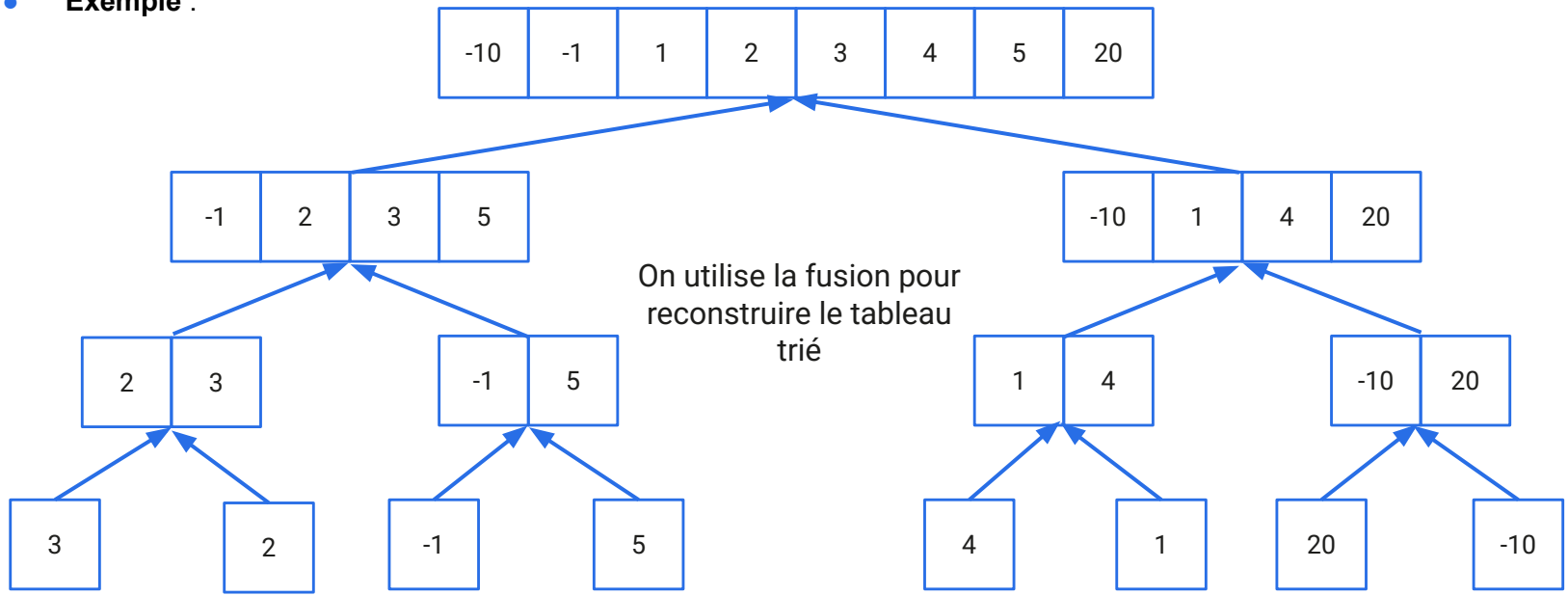
Tri par fusion

- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



Tri par fusion

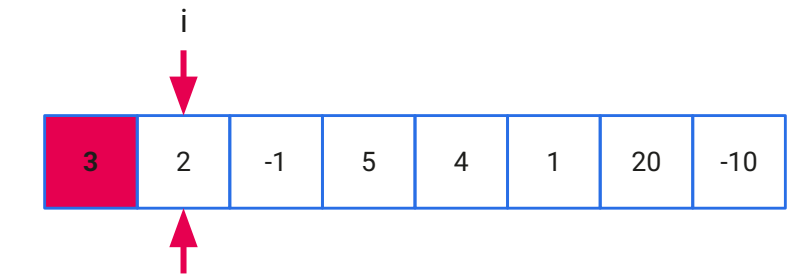
- **Principe** : Diviser la liste en sous-listes de plus en plus petites et les fusionner dans un ordre trié.
- **Complexité** : O(n log n) dans tous les cas.
- **Exemple** :



Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

On choisi un pivot (3 ici) et l'on organise la liste pour avoir les élément inférieurs au pivot, suivis du pivot suivi des éléments supérieurs au pivot.

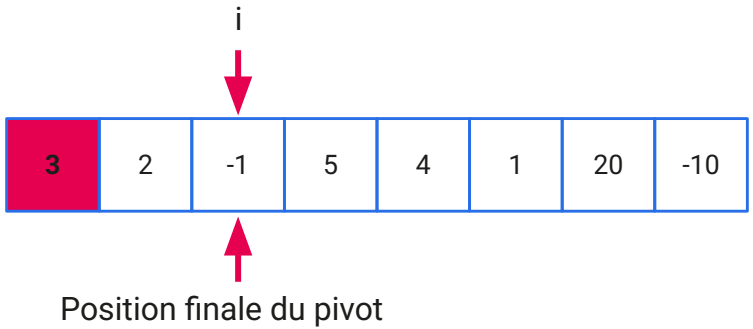


Position finale du pivot

Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

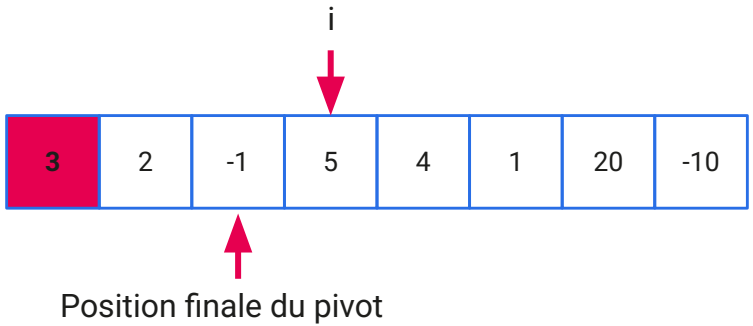
Quand on rencontre un élément plus petit que le pivot, on décale la position finale du pivot de 1 à droite et on échange l'élément à cette nouvelle position avec l'élément i (sans effect ici)



Tri rapide

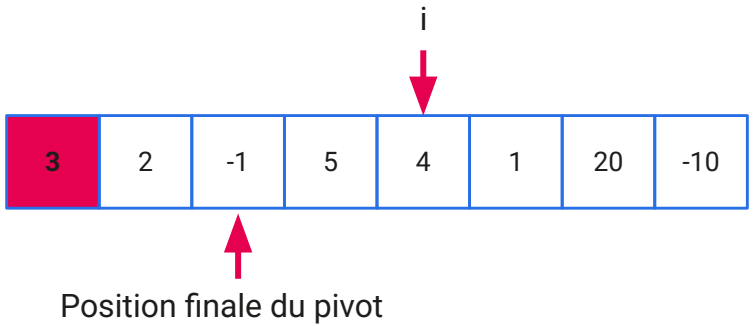
- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

Quand on rencontre un élément plus grand que le pivot, on ne fait rien



Tri rapide

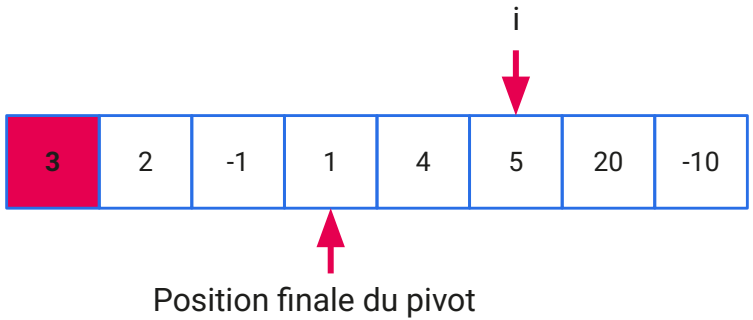
- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :



Tri rapide

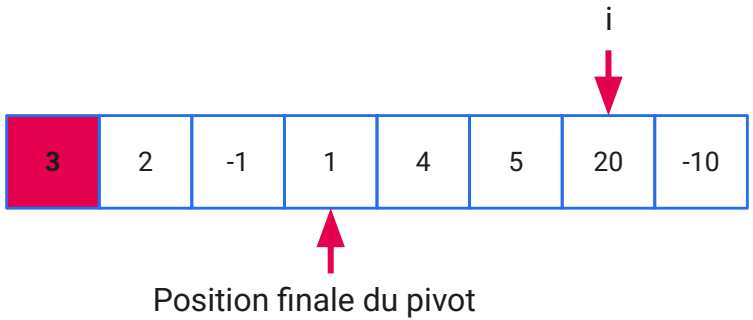
- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

Quand on rencontre un élément plus petit que le pivot, on décale la position finale du pivot de 1 à droite et on échange l'élément à cette nouvelle position avec l'élément i



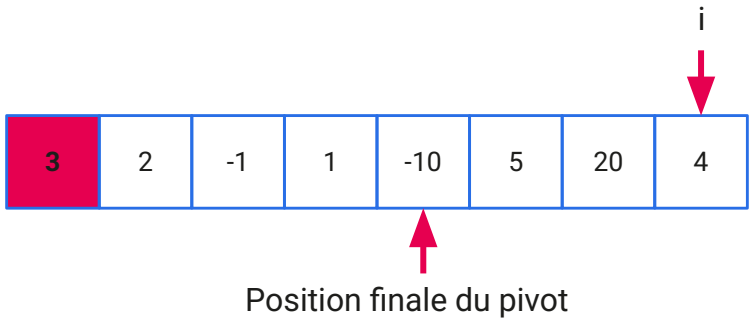
Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :



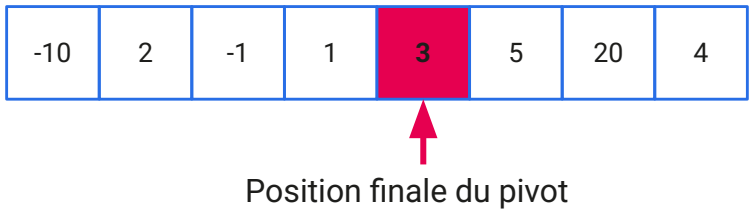
Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :



Tri rapide

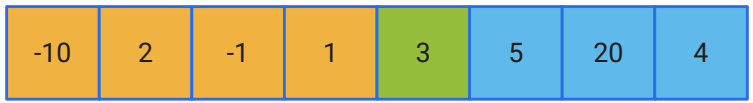
- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :



Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

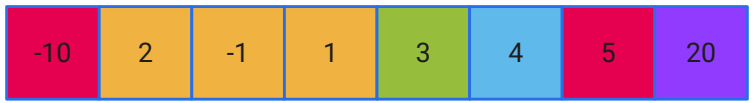
On réitère l'opération à droite et à gauche jusqu'à avoir un tableau trié (quand on tri un seul élément)



Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

On réitère l'opération à droite et à gauche jusqu'à avoir un tableau trié (quand on tri un seul élément)



Tri rapide

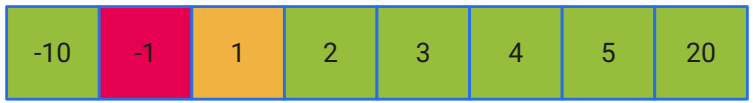
- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

On réitère l'opération à droite et à gauche jusqu'à avoir un tableau trié (quand on tri un seul élément)

Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

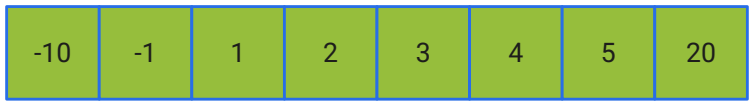
On réitère l'opération à droite et à gauche jusqu'à avoir un tableau trié (quand on tri un seul élément)



Tri rapide

- **Principe** : Choisir un pivot et partitionner la liste en deux sous-listes (inférieures et supérieures au pivot), puis trier récursivement ces sous-listes.
- **Complexité** : O(n log n) en moyenne, O(n²) dans le pire des cas.
- **Exemple** :

On réitère l'opération à droite et à gauche jusqu'à avoir un tableau trié (quand on tri un seul élément)



Il existe de nombreux algorithmes de tri, plus ou moins exotiques

- Tri par sélection :
  - Principe : Trouver le plus petit (ou plus grand) élément à chaque itération et l'échanger avec l'élément en cours de traitement.
  - Complexité : $O(n^2)$ dans tous les cas.
- Tri radix :
  - Principe : Tri non-comparatif qui classe les éléments en fonction de leurs chiffres (ou bits) à partir du moins significatif ou du plus significatif.
  - Complexité : $O(nk)$, où k est le nombre de chiffres.
- Tri par tas :
  - Principe : Construire un tas (c.f. futur TP) et extraire l'élément maximal pour obtenir un tri croissant.
  - Complexité : $O(n \log n)$.
- Cocktail shaker sort :
  - Principe : Variante du tri à bulle qui trie dans les deux directions (gauche → droite et droite → gauche).
  - Complexité : $O(n^2)$ dans tous les cas.
- Tri bitonique :
  - Principe : Créer une séquence bitonique (d'abord croissante, puis décroissante) et utiliser un tri fusion.
  - Complexité : $O(n \log^2 n)$, mais implémentation complexe.
- Bogo sort :
  - Principe : Algorithme de tri "stochastique" qui génère des permutations aléatoires jusqu'à ce que la liste soit triée (très inefficace).
  - Complexité : $O((n+1)!)$ en moyenne, considéré comme impraticable.

Visualisation: https://www.youtube.com/watch?v=kPRA0W1kECg https://youtu.be/3San3uKKHgg?si=wQQOtT0wgM9T9Q-u

Culture: Propriétés des tris

- La complexité optimale (pire des cas et moyenne) d'un algorithme de tri basé sur une fonction de comparaison est **O(nlog(n))**
  - La preuve est basée sur la construction d'un arbre binaire (voir plus tard dans le cours) où chaque nœud représente un comparaison.
- Dans certains cas (tris d'entiers), les complexités peuvent être meilleures (tri comptage, tri par base)
  - Ils n'utilisent pas de comparaisons
- Le tri peut se faire **en place** s'il modifie directement la structure à trier
  - Utile si pas beaucoup de mémoire
- Certains tris sont plus faciles à paralléliser que d'autres
  - On peut obtenir de meilleures performances
  - Voir les cours de calculs distribués et programmation GPU