

Module 01 – Révisions et compléments



Objectifs

- Généricité
- Révision des expressions lambda

En annexes : les tris et les recherches

Génériques – Avant un exemple !

- Prenons le cas suivant :
 - Écrire une fonction qui prend un tableau d'éléments et un valeur à trouver en paramètres et qui renvoie la position de la valeur si trouvée, -1 sinon. Le type d'éléments peut être de type entier, réel, chaîne, etc.

Génériques – En utilisant la surcharge

```
public static int Rechercher(int[] p_valeurs, int p_valeurARechercher)
{
    if (p_valeurs == null)
    {
        throw new ArgumentNullException(nameof(p_valeurs));
    }
    const int positionNonTrouvee = -1;
    int positionValeurARechercher = positionNonTrouvee;

    for (int indiceValeurCourante = 0;
        positionValeurARechercher == positionNonTrouvee && indiceValeurCourante < p_valeurs.Length;
        indiceValeurCourante++)
    {
        if (p_valeurs[indiceValeurCourante] == p_valeurARechercher)
        {
            positionValeurARechercher = indiceValeurCourante;
        }
    }

    return positionValeurARechercher;
}

public static int Rechercher(float[] p_valeurs, float p_valeurARechercher) { ... }

public static int Rechercher(decimal[] p_valeurs, decimal p_valeurARechercher) { ... }

public static int Rechercher(string[] p_valeurs, string p_valeurARechercher) { ... }

[...]
```

Répétition de code, donc non respect du principe DRY

Génériques

- But : éviter la répétition de code du simplement aux types de données en repoussant le type des paramètres à la déclaration et à l'instanciation d'une classe ou à l'utilisation d'une méthode
- Avec les génériques, les classes et méthodes sont paramétrées par des types
- Ces types peuvent ensuite être utilisés normalement comme n'importe quel type suivant sa portée

Génériques – C#

- En C#, pour **déclarer des paramètres génériques**, il suffit de suffixer le nom d'une classe ou d'une méthode par la liste des paramètres dont nous avons besoin **encadrée** par « < », et « > »
- Si vous avez besoin de déclarer **plusieurs types**, il suffit de les **séparer par une virgule** en suivant la syntaxe suivante : « <TypeA[, TypeB]*> »
- Les **noms de types** sont ensuite **utilisable comme n'importe quel type** pour les paramètres, les types de retour, les données membre, etc.
- La valeur par défaut d'un type générique est obtenue avec l'instruction « `default(NomDuType)` » (null pour les types références, ce qui correspond à 0 pour les types primitifs)
- Sur internet nous voyons souvent « T » comme nom : ce n'est pas une bonne pratique car le nom doit indiquer, au programmeur, à quoi fait référence le type

Génériques – C# – Retour sur l'exemple précédent

```
public static int Rechercher<TypeElement>(TypeElement[] p_valeurs, TypeElement p_valeurARechercher)
{
    if (p_valeurs == null) { throw new ArgumentNullException(nameof(p_valeurs)); }

    const int positionNonTrouvee = -1;
    int positionValeurARechercher = positionNonTrouvee;

    for (int indiceValeurCourante = 0;
        positionValeurARechercher == positionNonTrouvee && indiceValeurCourante < p_valeurs.Length;
        indiceValeurCourante++)
    {
        if (p_valeurs[indiceValeurCourante].Equals(p_valeurARechercher))
        {
            positionValeurARechercher = indiceValeurCourante;
        }
    }

    return positionValeurARechercher;
}
```

Génériques – C# – Retour sur l'exemple précédent

```
int[] valeursEntieres = new int[] { 13, 23, 42, 225 };

Console.Out.WriteLine(string.Join(", ", valeursEntieres));

int positionTrouveePour23 = Rechercher(valeursEntieres, 23);
Console.Out.WriteLine($"Position de 23 : {positionTrouveePour23}");

int positionTrouveePour42 = Rechercher<int>(valeursEntieres, 42);
Console.Out.WriteLine($"Position de 42 : {positionTrouveePour42}");
```

```
13, 23, 42, 225
Position de 23 : 1
Position de 42 : 2
```

Génériques – C# – Autre exemple échange de valeurs dans un tableau

```
public static void EchangerValeurs<TypeElement>(TypeElement[] p_valeurs, int p_position1, int p_position2)
{
    if (p_valeurs == null)
    {
        throw new ArgumentNullException(nameof(p_valeurs));
    }
    if (p_position1 < 0 || p_position1 >= p_valeurs.Length)
    {
        throw new ArgumentOutOfRangeException(nameof(p_position1));
    }
    if (p_position2 < 0 || p_position2 >= p_valeurs.Length)
    {
        throw new ArgumentOutOfRangeException(nameof(p_position2));
    }

    TypeElement temp = p_valeurs[p_position1];
    p_valeurs[p_position1] = p_valeurs[p_position2];
    p_valeurs[p_position2] = temp;
}
```

```
int[] valeursEntieres = new int[] { 13, 23, 42, 225 };

Console.Out.WriteLine(string.Join(", ", valeursEntieres));
EchangerValeurs(valeursEntieres, 0, valeursEntieres.Length - 1);
Console.Out.WriteLine(string.Join(", ", valeursEntieres));
```

13, 23, 42, 225
225, 23, 42, 13

Générique – C# - Exemple avec une classe

```
public class Vecteur2D<TypeCoordonnee> {
    public TypeCoordonnee X { get; set; }
    public TypeCoordonnee Y { get; set; }

    public static Vecteur2D<TypeCoordonnee> operator +(Vecteur2D<TypeCoordonnee> p_vecteur1, Vecteur2D<TypeCoordonnee> p_vecteur2) {
        // + préconditions : p_valeur1 != null et p_valeur2 != null
        return new Vecteur2D<TypeCoordonnee>()
        {
            X = (dynamic)p_vecteur1.X + (dynamic)p_vecteur2.X,
            Y = (dynamic)p_vecteur1.Y + (dynamic)p_vecteur2.Y
        };
    }

    public override string ToString() {
        return $"Vecteur2D({this.X}, {this.Y})";
    }
}
```

Générique – C# - Exemple avec une classe

```
public static void DemoVecteur2D() {  
    Vecteur2D<int> vecteurXEntier = new Vecteur2D<int>() { X = 1, Y = 0 };  
    Vecteur2D<int> vecteurYEntier = new Vecteur2D<int>() { X = 0, Y = 1 };  
    Vecteur2D<int> sommeVecteursEntiers = vecteurXEntier + vecteurYEntier;  
    Console.Out.WriteLine($"{vecteurXEntier} + {vecteurYEntier} = {sommeVecteursEntiers}");  
  
    Vecteur2D<float> vecteurXFloat = new Vecteur2D<float>() { X = 1.1f, Y = 0 };  
    Vecteur2D<float> vecteurYFloat = new Vecteur2D<float>() { X = 0, Y = 1.1f };  
    Vecteur2D<float> sommeVecteursFloats = vecteurXFloat + vecteurYFloat;  
    Console.Out.WriteLine($"{vecteurXFloat} + {vecteurYFloat} = {sommeVecteursFloats}");  
  
    //Erreur : Vecteur2D<float> sommeVecteursEntierFloat = vecteurXEntier + vecteurYFloat;  
}
```

```
Vecteur2D(1, 0) + Vecteur2D(0, 1) = Vecteur2D(1, 1)  
Vecteur2D(1,1, 0) + Vecteur2D(0, 1,1) = Vecteur2D(1,1, 1,1)
```

Générique – C# - Contraintes sur le type paramétré

```
public static TypeElement CalculerMinimum<TypeElement>(TypeElement p_valeur1, TypeElement p_valeur2)
{
    TypeElement minimum = p_valeur1;

    if (p_valeur2 < minimum)
    {
        minimum = p_valeur2;
    }

    return minimum;
}
```

- Ici, on ne peut pas faire de comparaison car TypeElement ne définit pas l'opérateur de comparaison « < »

Générique – C# - Contraintes sur le type paramétré

- Pour ajouter une contrainte sur le paramétrisé, il suffit d'ajouter l'instruction where à la fin de la déclaration du type générique

```
public static TypeElement CalculerMinimum<TypeElement>(TypeElement p_valeur1, TypeElement p_valeur2)
where TypeElement : IComparable<TypeElement>
{
    TypeElement minimum = p_valeur1;

    if (p_valeur2.CompareTo(minimum) < 0)
    {
        minimum = p_valeur2;
    }

    return minimum;
}
```

<https://docs.microsoft.com/en-us/dotnet/api/system.icomparable-1?view=netcore-3.1>

Tests unitaires des génériques

- Au niveau des tests unitaires, vous pouvez utiliser des types compatibles avec votre méthode ou classe générique
- Faites ensuite vos tests comme d'habitude avec la description en AAA (Arranger, Agir, Auditer)

Expressions lambda

- Une expression lambda est une **fonction anonyme**, c'est-à-dire sans nom
- Comme les méthodes, elles peuvent prendre des valeurs en paramètres
- Les types de paramètres peuvent généralement être déduit par le compilateur
- Les expressions lambda peuvent être passées en paramètres de fonctions / méthodes. Ce modèle est proche du patron de conception « stratégie »

Expression lambda – C#

- En C#, une expression lambda est décrite avec la syntaxe suivante :
 - $([[\text{NomParamètre}1][, \text{NomParamètre}i]*]) \Rightarrow \{ \text{instruction}1; \text{instruction}2; \dots \}$ ou instUnique
 - Devant le nom du paramètre, on peut spécifier un type, mais il est généralement déduit par le compilateur
 - Si vous avez un seul paramètre, les parenthèses sont optionnelles

```
v => v;  
(v) => v;  
v => { return v; };
```

```
n => n * n; // TypeValeur _(TypeValeur n)  
  
(valeur1, valeur2) => valeur1 + valeur2;  
// TypeValeur _(TypeValeur valeur1, TypeValeur valeur2)  
  
(valeur1, valeur2) => valeur1 <= valeur2;  
// bool _(TypeValeur valeur1, TypeValeur valeur2)
```

Expression lambda – C#

- En C#, une expression lambda peut être exploitée :
 - Comme une fonction représentée par les classes génériques
 - Exemple : pouvoir passer des fonctions en paramètres (Voir Linq)
 - Comme un arbre d'expressions : ensemble des classes filles d'« Expression »:
 - Exemple : pouvoir être traduit en SQL (ORM de type EntityFramework)

Expression lambda – C#

- Comme une fonction représentée par les classes génériques :
 - « Func<[TypeParam1][, TypeParam2]*], TypeRetour> » : avec valeur de retour
 - « Action<[TypeParam1][, TypeParam2]*> » : sans valeur de retour

```
Func<double, double> varRefFunc1 = v => v;
Func<double, double> varRefFunc2 = (v) => v;
Func<double, double> varRefFunc3 = v => { return v; };

Func<double, double> varRefFunc4 = n => n * n;

Func<double, double, double> varRefFunc5 = (valeur1, valeur2) => valeur1 + valeur2;

Func<double, double, bool> varRefFunc6 = (valeur1, valeur2) => valeur1 <= valeur2;

Action<int> varRefAction1 = v => Console.Out.WriteLine(v);
```

Expression lambda – C#

- Comme une fonction représentée par les classes génériques
 - Utilisation :

```
Func<double, double> varRefFuncCarre = n => n * n;  
  
double valeurCarre1 = varRefFuncCarre(42.0);  
  
Console.Out.WriteLine($"valeurCarre1 = {valeurCarre1}");
```

Expression lambda – C#

- Comme une fonction représentée par les classes génériques
 - Un peu fun :

```
double valeurCarre2 = ExecuterTraitement(varRefFuncCarre, 42.0);
Console.Out.WriteLine($"valeurCarre2 = {valeurCarre2}");
```

```
double valeurCarre3 = ExecuterTraitement(n => n * n, 42.0);
Console.Out.WriteLine($"valeurCarre3 = {valeurCarre3}");
```

```
public static TypeResultat ExecuterTraitement<TypeDonnee, TypeResultat>(Func<TypeDonnee, TypeResultat> p_traitement,
                                                                     TypeDonnee p_donnee)
{
    if (p_traitement == null) { throw new ArgumentNullException(nameof(p_traitement)); }

    return p_traitement(p_donnee);
}
```

Expression lambda – C#

- Comme une fonction représentée par les classes génériques
 - Une version simplifiée du « where » :

```
public static class ExtensionsListes
{
    public static List<TypeElement> Filtrer<TypeElement>(this List<TypeElement> p_valeurs,
                                                          Func<TypeElement, bool> p_aGarder) {
        List<TypeElement> valeursFiltrees = new List<TypeElement>();

        foreach (TypeElement valeur in p_valeurs) {
            if (p_aGarder(valeur))
            {
                valeursFiltrees.Add(valeur);
            }
        }

        return valeursFiltrees;
    }
}
```

Expression lambda – C#

- Comme une fonction représentée par les classes génériques
 - Utilisation :

```
List<int> listeEntiersAFiltrer = new List<int>() { 1, 2, 3, 4, 5, 6, 7 };

List<int> listeEntiersMultiples3 = listeEntiersAFiltrer.Filtrer(n => n % 3 == 0);

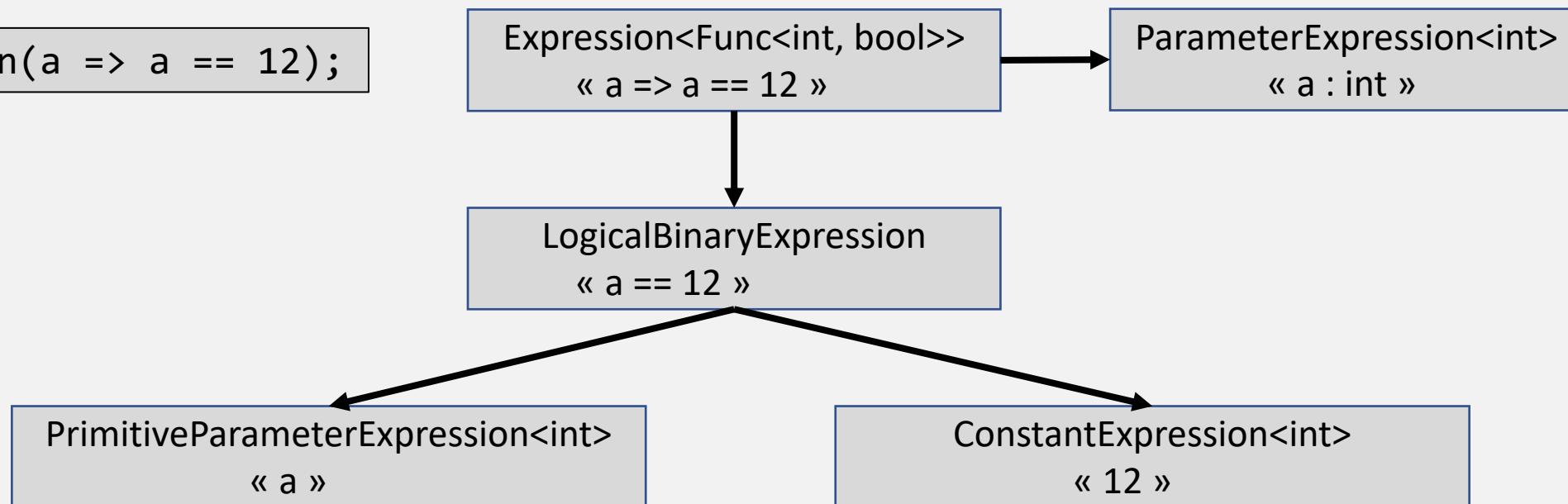
Console.Out.WriteLine($"listeEntiersAFiltrer : {string.Join(", ", listeEntiersAFiltrer)}");
Console.Out.WriteLine($"listeEntiersMultiples3 : {string.Join(", ", listeEntiersMultiples3)}");
```

```
listeEntiersAFiltrer : 1, 2, 3, 4, 5, 6, 7
listeEntiersMultiples3 : 3, 6
```

Expression lambda – C#

- Comme un arbre d'expressions : ensemble des classes filles d'« Expression » pour, par exemple, pouvoir être traduit en SQL (ORM de type EntityFramework)

```
FonctionAvecExpression(a => a == 12);
```



Annexes

- Tri à bulle
- Tri rapide
- Recherche simple
- Recherche dichotomique

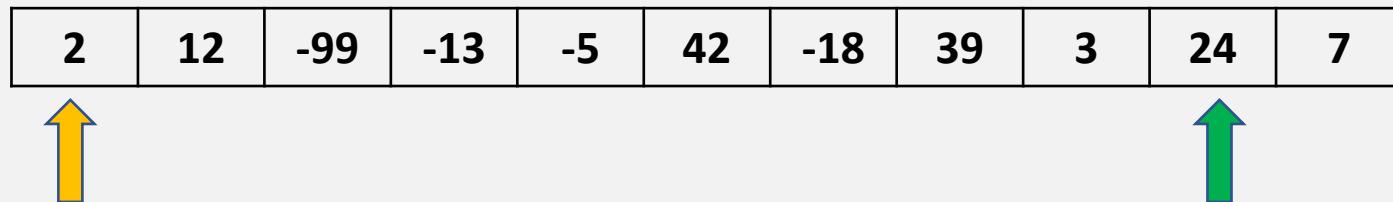
Tri à bulles

- Le nom provient du principe de ce tri : on parcours le tableau de gauche à droite. Si une valeur est plus grande que la suivante, on la décale vers la droite du tableau comme le feraient des bulles dans un verre

Tri à bulles

permutationAuDernierTour => **vrai**

2	12	-99	-13	-5	42	-18	39	3	24	7
---	----	-----	-----	----	----	-----	----	---	----	---



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

2	12	-99	-13	-5	42	-18	39	3	24	7
---	----	-----	-----	----	----	-----	----	---	----	---

The diagram illustrates a step in the bubble sort algorithm. A horizontal array of 11 cells contains the values: 2, 12, -99, -13, -5, 42, -18, 39, 3, 24, 7. A yellow arrow points upwards from the bottom left towards the cell containing -99, labeled 'indiceCourant'. A green arrow points upwards from the bottom right towards the cell containing 39, labeled 'indiceMax'.

↑ indiceCourant

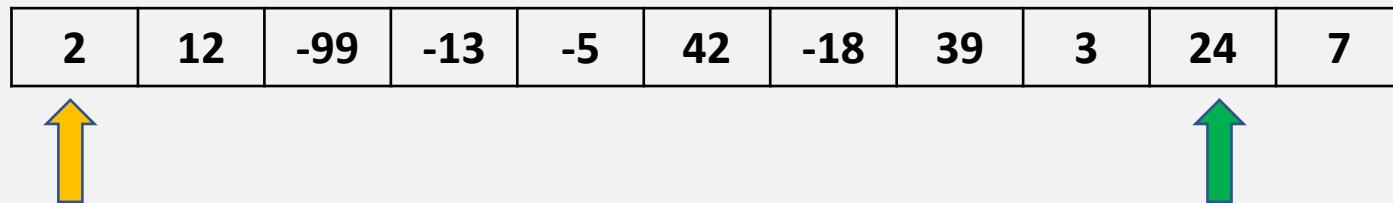
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > 12 \Rightarrow \text{faux}$

2	12	-99	-13	-5	42	-18	39	3	24	7
---	----	-----	-----	----	----	-----	----	---	----	---



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$12 > -99 \Rightarrow$ **vrai**

2	12	-99	-13	-5	42	-18	39	3	24	7
---	----	-----	-----	----	----	-----	----	---	----	---



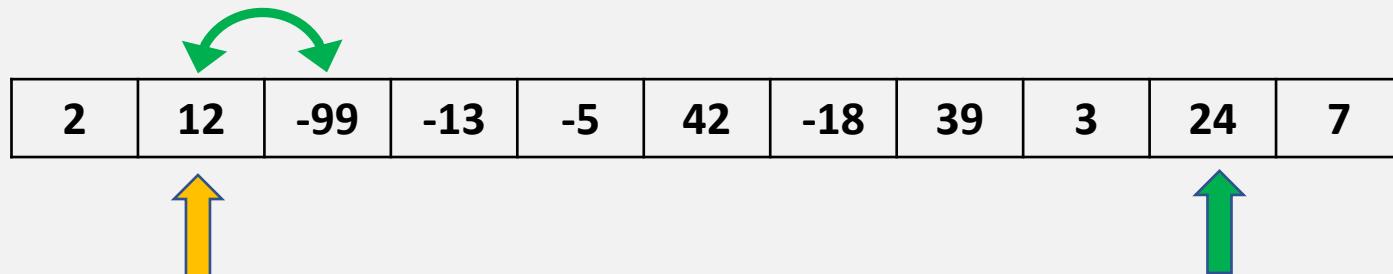
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$12 > -99 \Rightarrow$ **vrai**



↑ indiceCourant

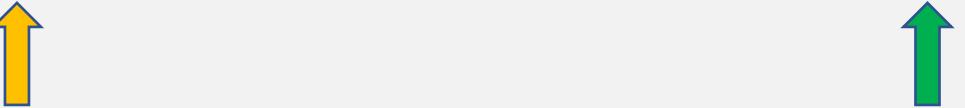
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -13 \Rightarrow \text{vrai}$

2	-99	12	-13	-5	42	-18	39	3	24	7
---	-----	----	-----	----	----	-----	----	---	----	---



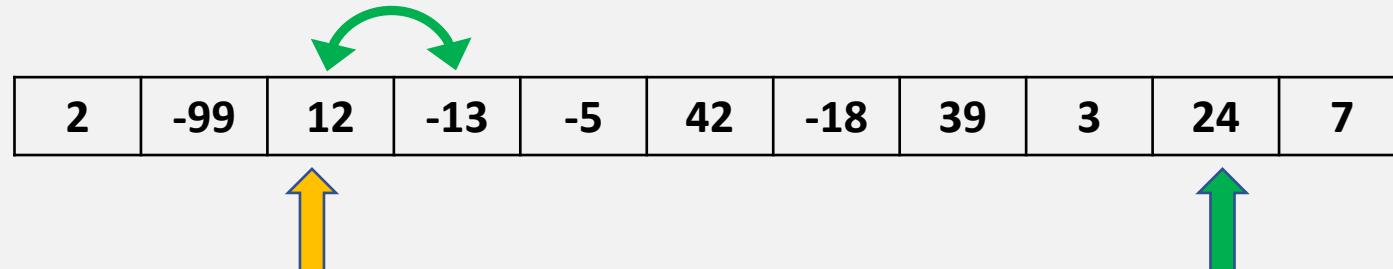
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -13 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -5 \Rightarrow \text{vrai}$

2	-99	-13	12	-5	42	-18	39	3	24	7
---	-----	-----	----	----	----	-----	----	---	----	---



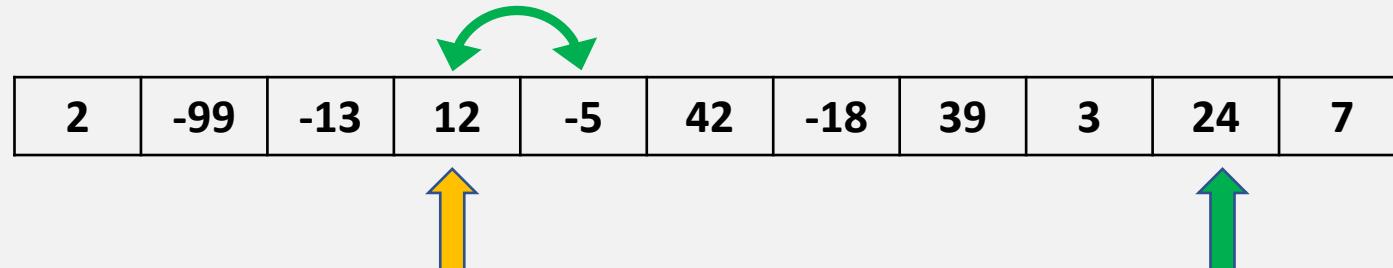
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -5 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 42 \Rightarrow \text{faux}$

2	-99	-13	-5	12	42	-18	39	3	24	7
---	-----	-----	----	----	----	-----	----	---	----	---

```
graph TD; A[2] --- B[-99] --- C[-13] --- D[-5] --- E[12] --- F[42] --- G[-18] --- H[39] --- I[3] --- J[24] --- K[7]; E --> F; F --> G; G --> H; H --> I; I --> J; J --> K; style E fill:#ffff00,stroke:#000,stroke-width:2px; style F fill:#00ff00,stroke:#000,stroke-width:2px; style G fill:#ffff00,stroke:#000,stroke-width:2px; style H fill:#00ff00,stroke:#000,stroke-width:2px; style I fill:#ffff00,stroke:#000,stroke-width:2px; style J fill:#00ff00,stroke:#000,stroke-width:2px; style K fill:#ffff00,stroke:#000,stroke-width:2px;
```

↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > -18 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	42	-18	39	3	24	7
---	-----	-----	----	----	----	-----	----	---	----	---



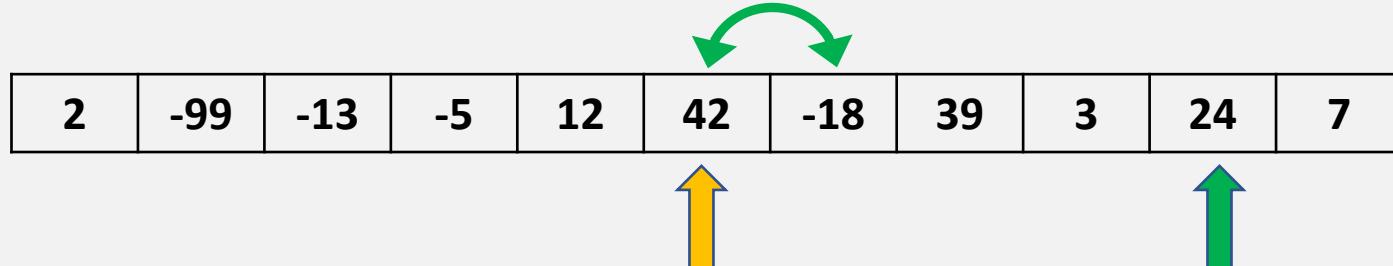
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > -18 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 39 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	42	39	3	24	7
---	-----	-----	----	----	-----	----	----	---	----	---



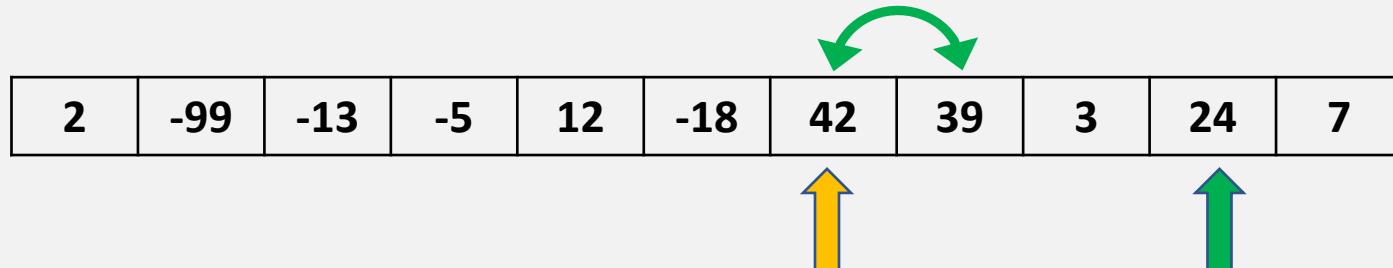
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 39 \Rightarrow \text{vrai}$



↑ indiceCourant

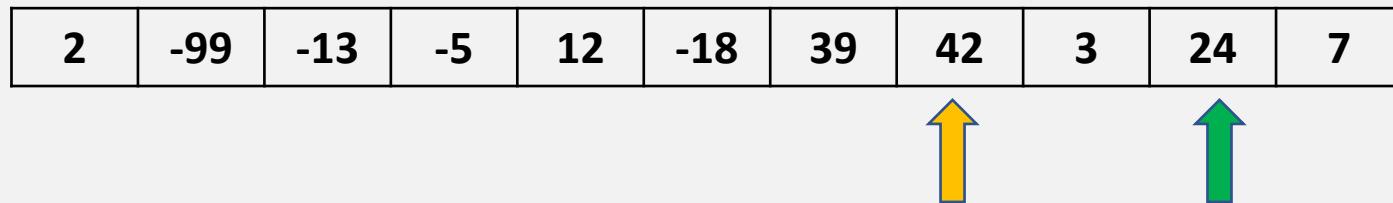
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 3 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	39	42	3	24	7
---	-----	-----	----	----	-----	----	----	---	----	---



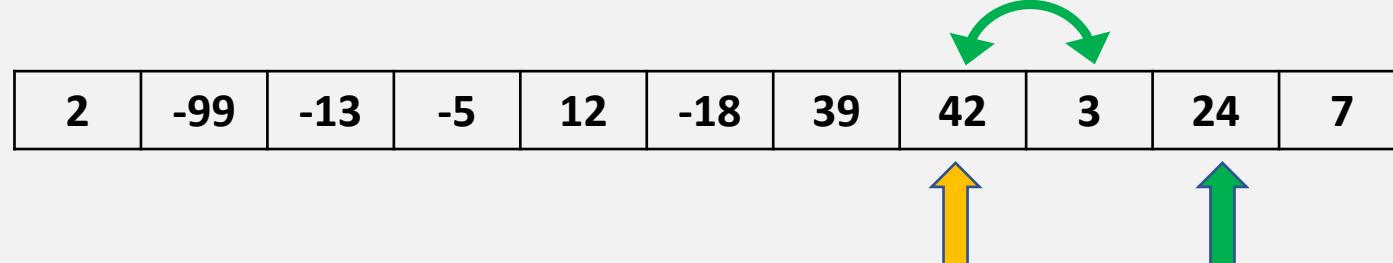
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 3 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 24 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	39	3	42	24	7
---	-----	-----	----	----	-----	----	---	----	----	---

A diagram illustrating the state of an array during bubble sort. The array is represented as a horizontal row of 11 boxes, each containing a value: 2, -99, -13, -5, 12, -18, 39, 3, 42, 24, 7. A yellow arrow points to the 10th element (24), labeled "indiceCourant". A green arrow points to the 11th element (7), labeled "indiceMax". A blue arrow points to the 9th element (39), labeled "permutationAuDernierTour".

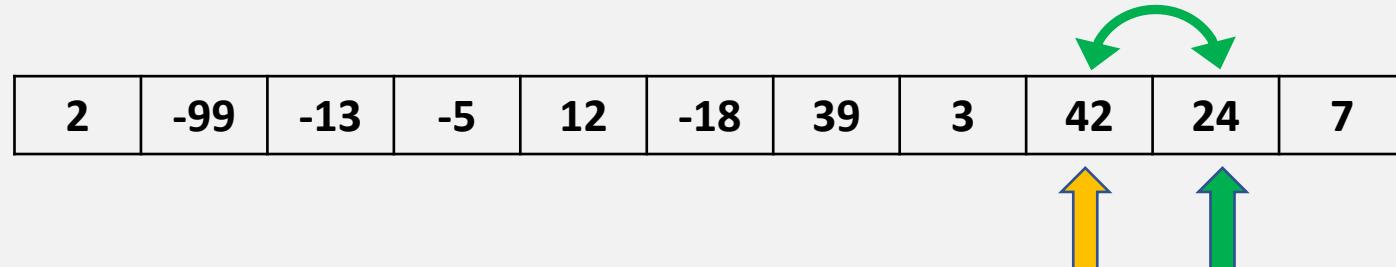
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 24 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 7 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	39	3	24	42	7
---	-----	-----	----	----	-----	----	---	----	----	---



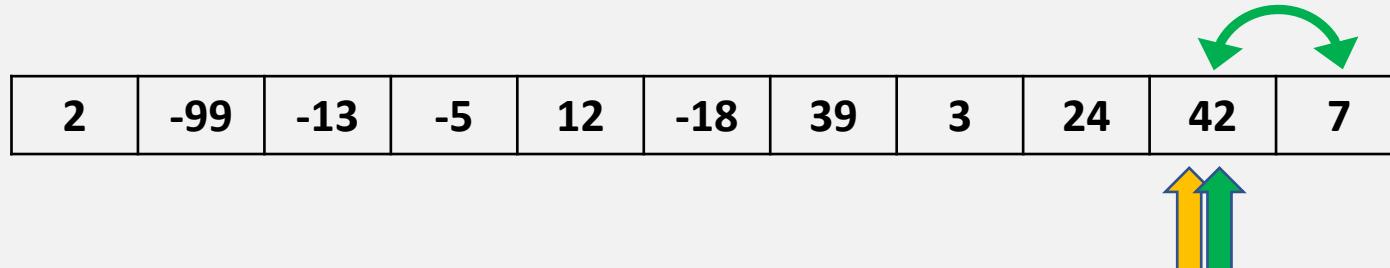
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 7 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$42 > 7 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	39	3	24	7	42
---	-----	-----	----	----	-----	----	---	----	---	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

2	-99	-13	-5	12	-18	39	3	24	7	42
---	-----	-----	----	----	-----	----	---	----	---	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > -99 \Rightarrow \text{vrai}$

2	-99	-13	-5	12	-18	39	3	24	7	42
---	-----	-----	----	----	-----	----	---	----	---	----



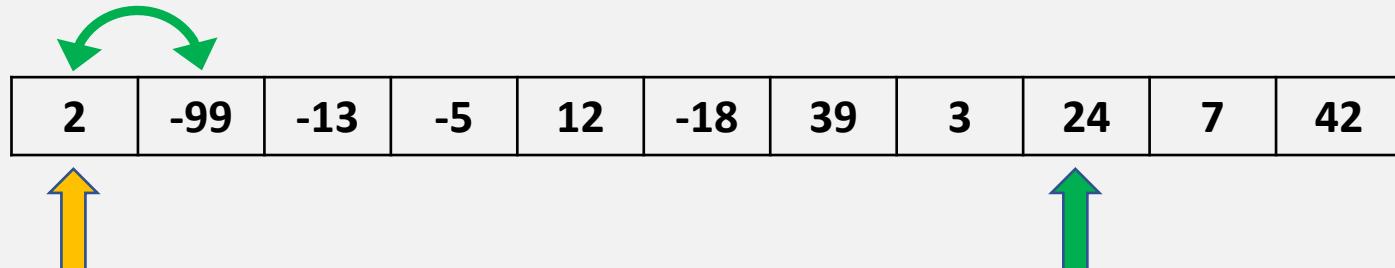
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > -99 \Rightarrow$ **vrai**



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -99 \Rightarrow \text{vrai}$

-99	2	-13	-5	12	-18	39	3	24	7	42
-----	---	-----	----	----	-----	----	---	----	---	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -13 \Rightarrow \text{vrai}$

-99	2	-13	-5	12	-18	39	3	24	7	42
-----	---	-----	----	----	-----	----	---	----	---	----



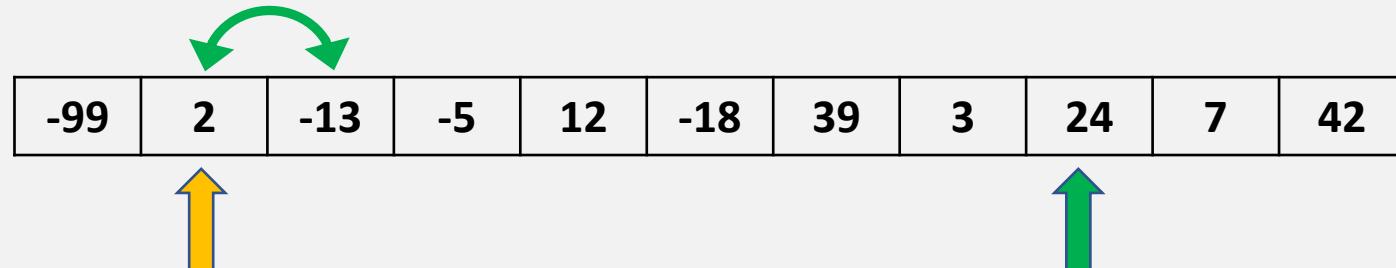
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -13 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -5 \Rightarrow \text{vrai}$

-99	-13	2	-5	12	-18	39	3	24	7	42
-----	-----	---	----	----	-----	----	---	----	---	----



indiceCourant

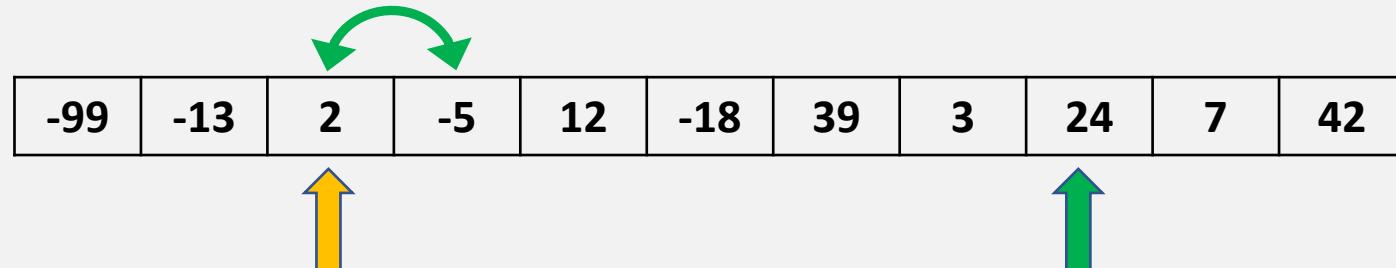


indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -5 \Rightarrow \text{vrai}$



↑ indiceCourant

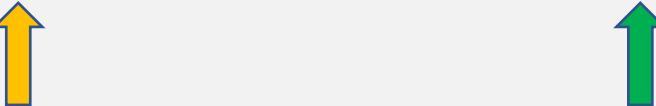
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 12 \Rightarrow \text{faux}$

-99	-13	-5	2	12	-18	39	3	24	7	42
-----	-----	----	---	----	-----	----	---	----	---	----



↑ indiceCourant

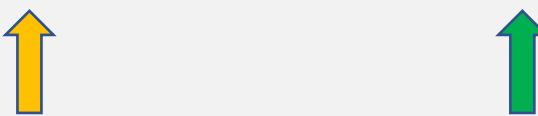
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -18 \Rightarrow \text{vrai}$

-99	-13	-5	2	12	-18	39	3	24	7	42
-----	-----	----	---	----	-----	----	---	----	---	----



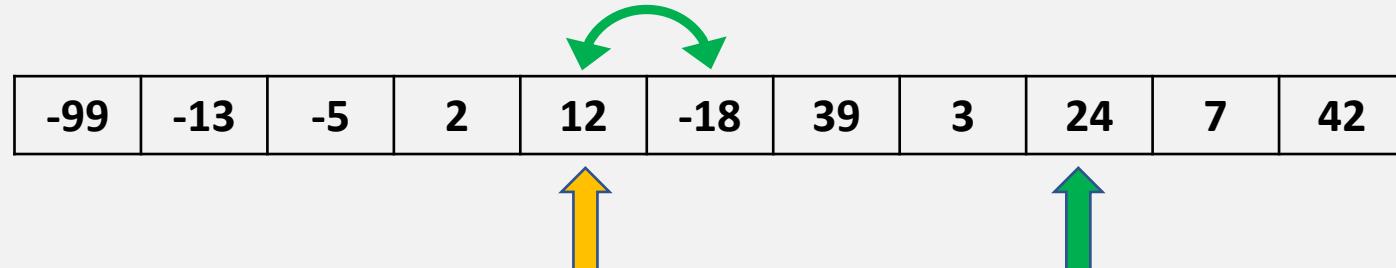
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -18 \Rightarrow \text{vrai}$



↑ indiceCourant

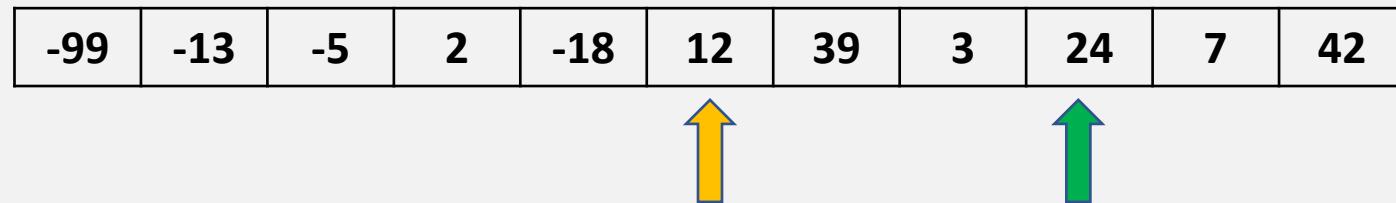
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 39 \Rightarrow \text{faux}$

-99	-13	-5	2	-18	12	39	3	24	7	42
-----	-----	----	---	-----	----	----	---	----	---	----



↑ indiceCourant

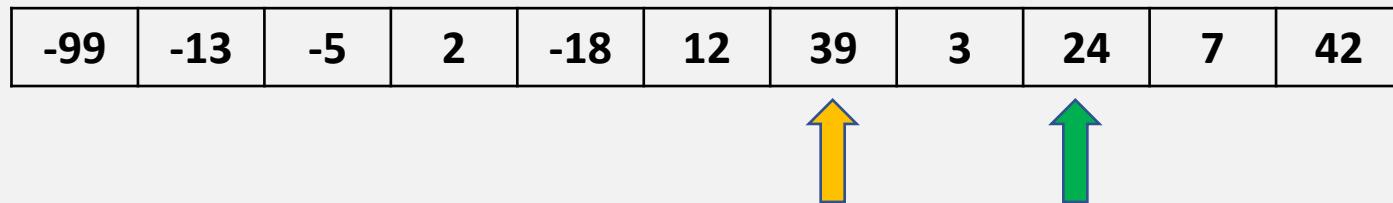
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 3 \Rightarrow \text{vrai}$

-99	-13	-5	2	-18	12	39	3	24	7	42
-----	-----	----	---	-----	----	----	---	----	---	----



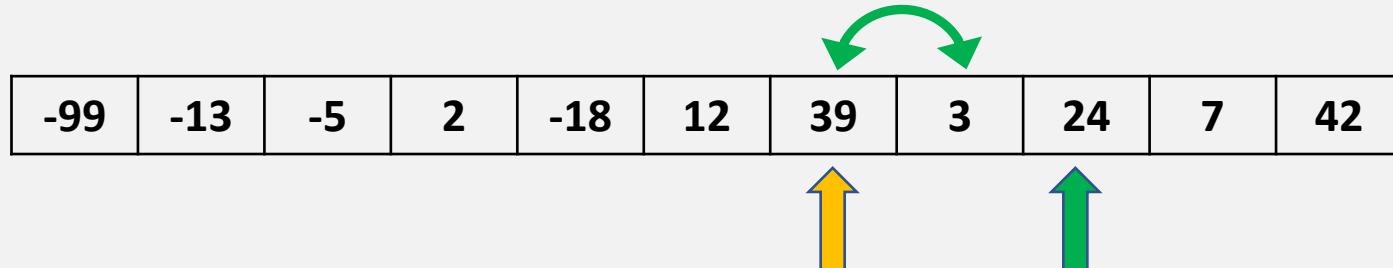
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 3 \Rightarrow \text{vrai}$



↑ indiceCourant

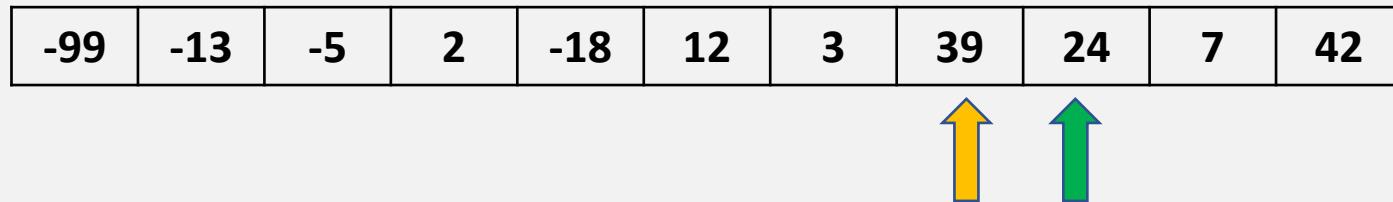
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 24 \Rightarrow \text{vrai}$

-99	-13	-5	2	-18	12	3	39	24	7	42
-----	-----	----	---	-----	----	---	-----------	-----------	---	----



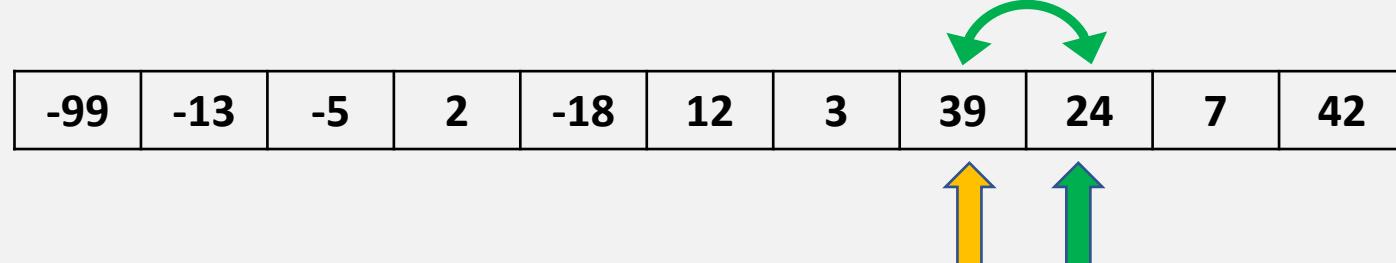
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 24 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 7 \Rightarrow \text{vrai}$

-99	-13	-5	2	-18	12	3	24	39	7	42
-----	-----	----	---	-----	----	---	----	----	---	----



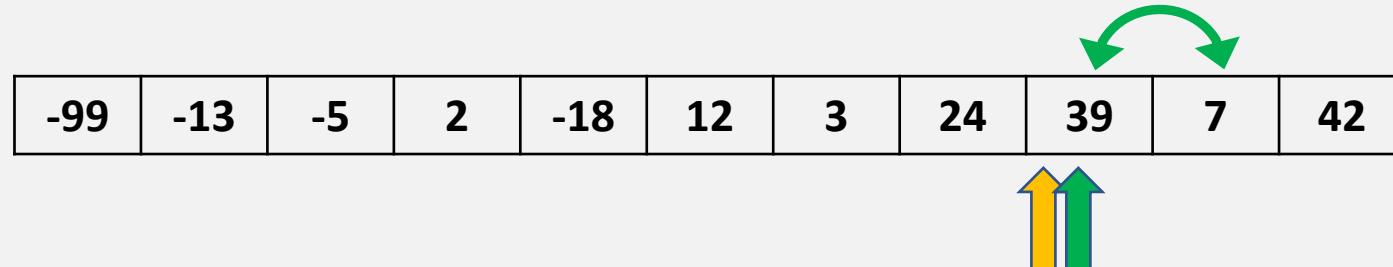
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 7 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 7 \Rightarrow \text{vrai}$

-99	-13	-5	2	-18	12	3	24	7	39	42
-----	-----	----	---	-----	----	---	----	---	----	----



↑ indiceCourant

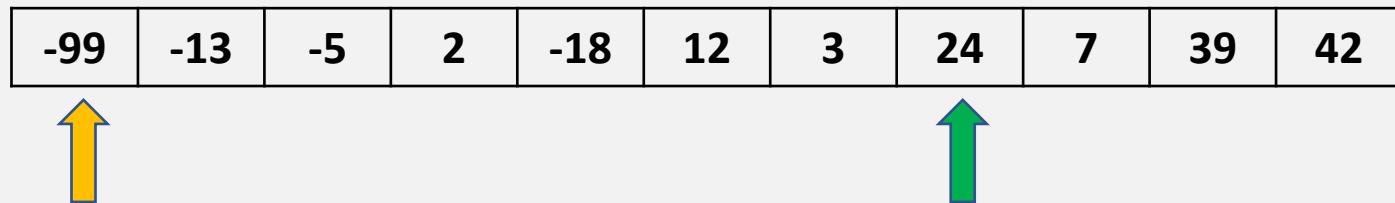
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -13 \Rightarrow \text{faux}$

-99	-13	-5	2	-18	12	3	24	7	39	42
-----	-----	----	---	-----	----	---	----	---	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -5 \Rightarrow \text{faux}$

-99	-13	-5	2	-18	12	3	24	7	39	42
-----	-----	----	---	-----	----	---	----	---	----	----



↑ indiceCourant

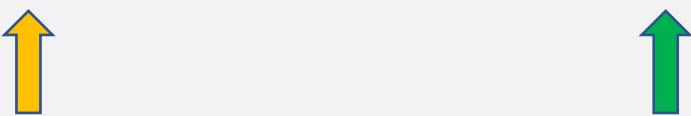
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-5 > 2 \Rightarrow \text{faux}$

-99	-13	-5	2	-18	12	3	24	7	39	42
-----	-----	----	---	-----	----	---	----	---	----	----



↑ indiceCourant

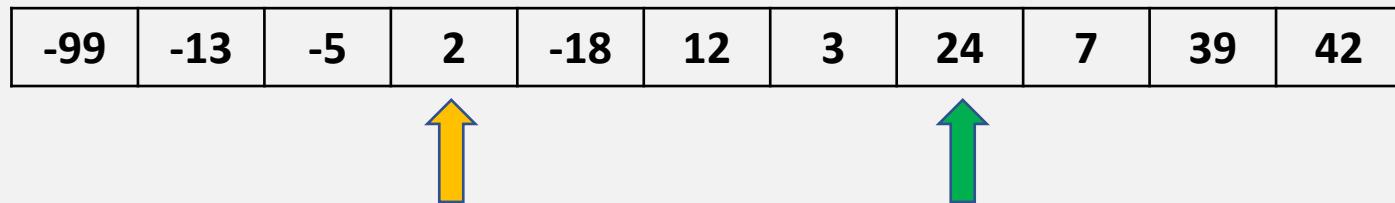
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > -18 \Rightarrow$ **vrai**

-99	-13	-5	2	-18	12	3	24	7	39	42
-----	-----	----	---	-----	----	---	----	---	----	----



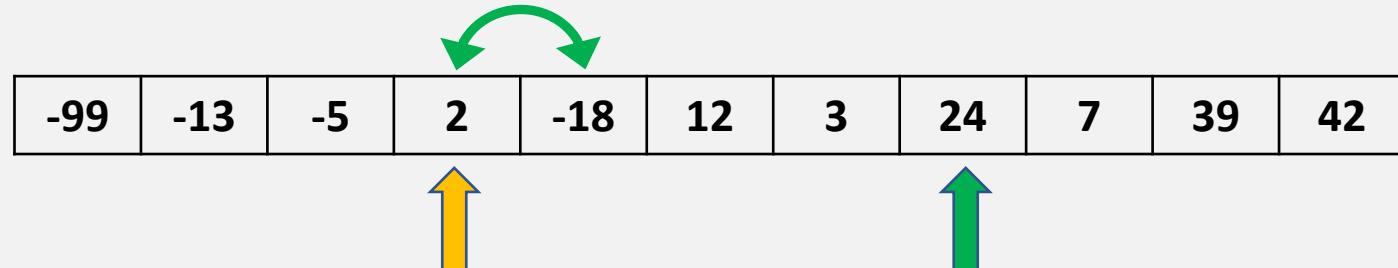
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > -18 \Rightarrow$ **vrai**



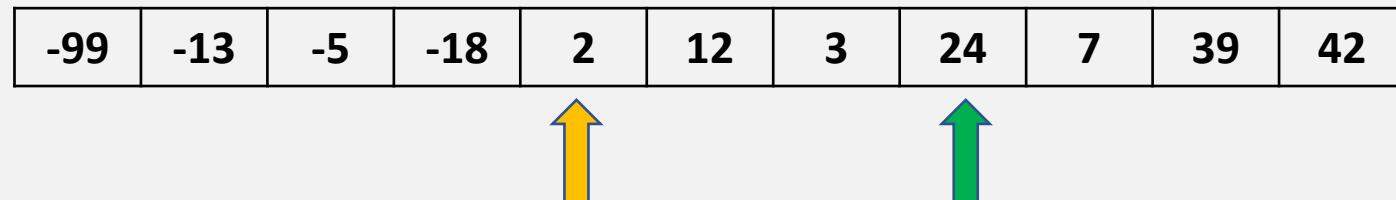
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 12 \Rightarrow \text{faux}$



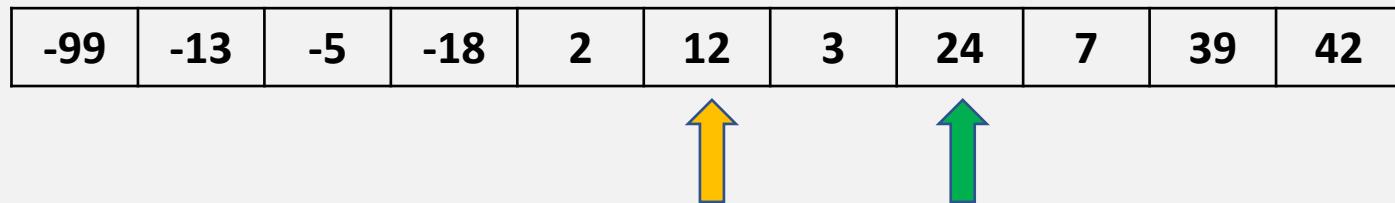
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 3 \Rightarrow \text{vrai}$



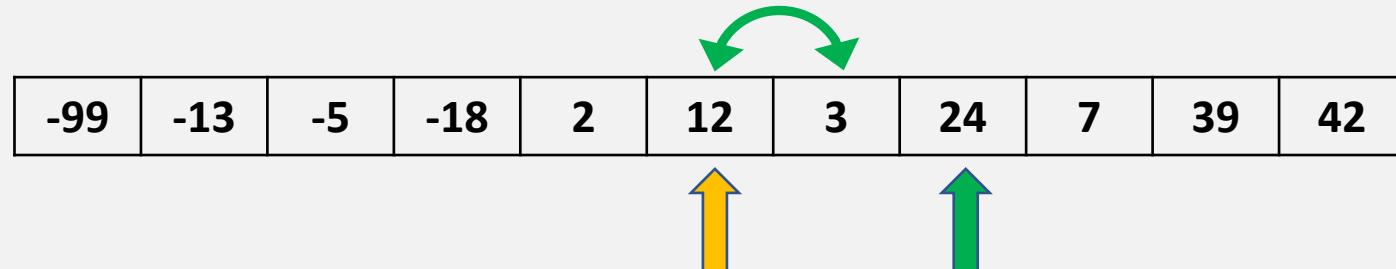
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 3 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 24 \Rightarrow \text{faux}$

-99	-13	-5	-18	2	3	12	24	7	39	42
-----	-----	----	-----	---	---	----	----	---	----	----



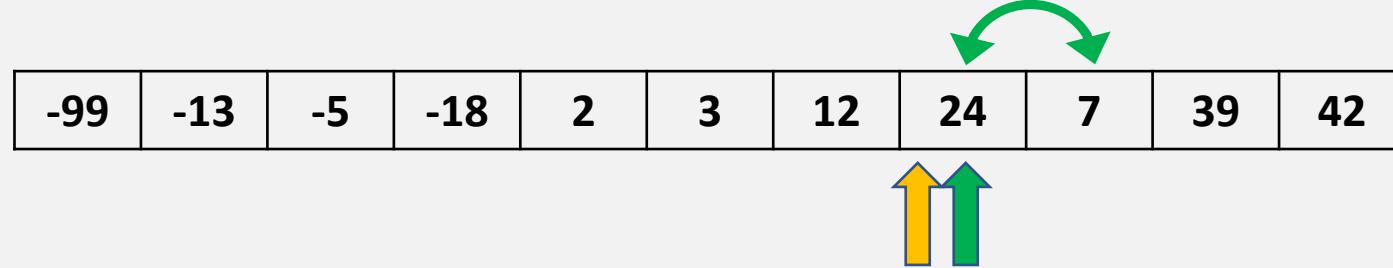
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$24 > 7 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$24 > 7 \Rightarrow \text{vrai}$

-99	-13	-5	-18	2	3	12	7	24	39	42
-----	-----	----	-----	---	---	----	---	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -13 \Rightarrow \text{faux}$

-99	-13	-5	-18	2	3	12	7	24	39	42
-----	-----	----	-----	---	---	----	---	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -5 \Rightarrow \text{faux}$

-99	-13	-5	-18	2	3	12	7	24	39	42
-----	-----	----	-----	---	---	----	---	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

-5 > -18 => **vrai**

-99	-13	-5	-18	2	3	12	7	24	39	42
-----	-----	----	-----	---	---	----	---	----	----	----



indiceCourant

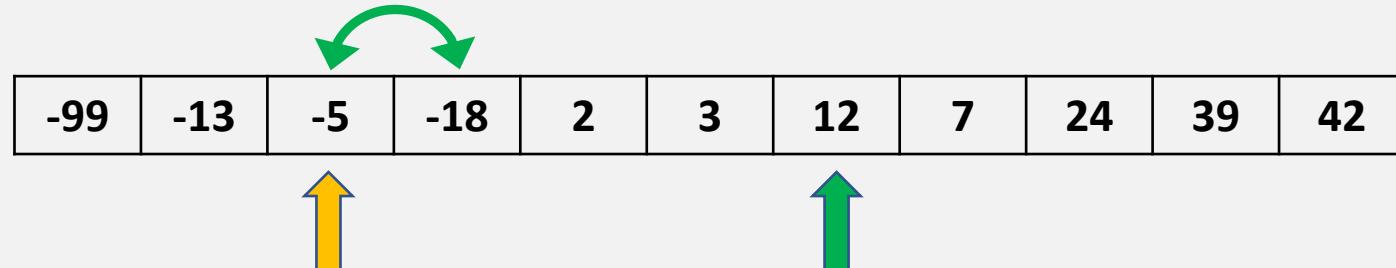


indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

-5 > -18 => **vrai**



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$-5 > 2 \Rightarrow \text{faux}$

-99	-13	-18	-5	2	3	12	7	24	39	42
-----	-----	-----	----	---	---	----	---	----	----	----



↑ indiceCourant

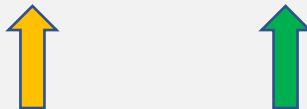
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 3 \Rightarrow \text{faux}$

-99	-13	-18	-5	2	3	12	7	24	39	42
-----	-----	-----	----	---	---	----	---	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$3 > 12 \Rightarrow \text{faux}$

-99	-13	-18	-5	2	3	12	7	24	39	42
-----	-----	-----	----	---	---	----	---	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 7 \Rightarrow \text{vrai}$

-99	-13	-18	-5	2	3	12	7	24	39	42
-----	-----	-----	----	---	---	----	---	----	----	----



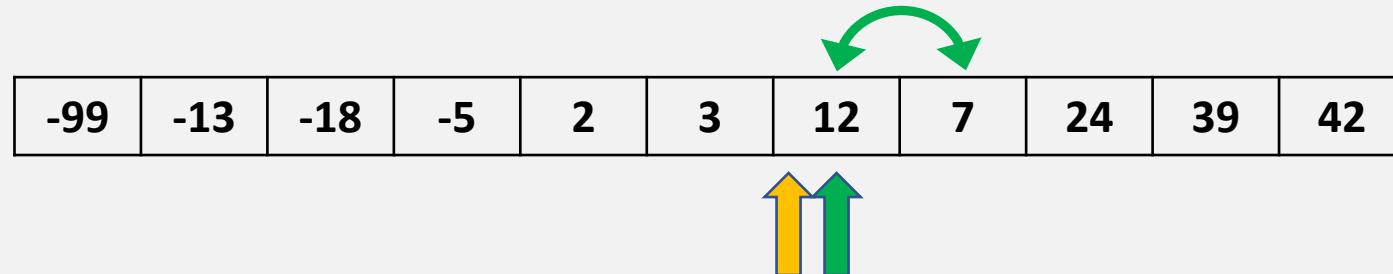
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 7 \Rightarrow \text{vrai}$



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

-99	-13	-18	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -13 \Rightarrow \text{faux}$

-99	-13	-18	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -18 \Rightarrow \text{vrai}$

-99	-13	-18	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



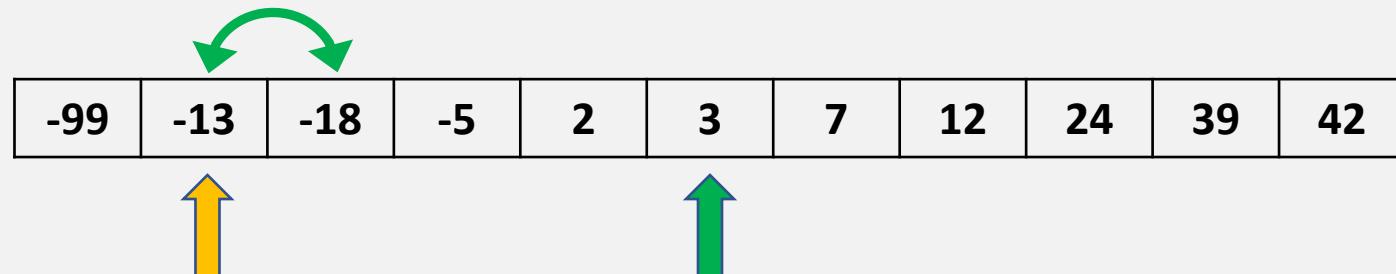
↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -18 \Rightarrow$ **vrai**



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$-13 > -5 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

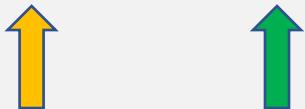
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$-5 > 2 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 3 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$3 > 7 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -18 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

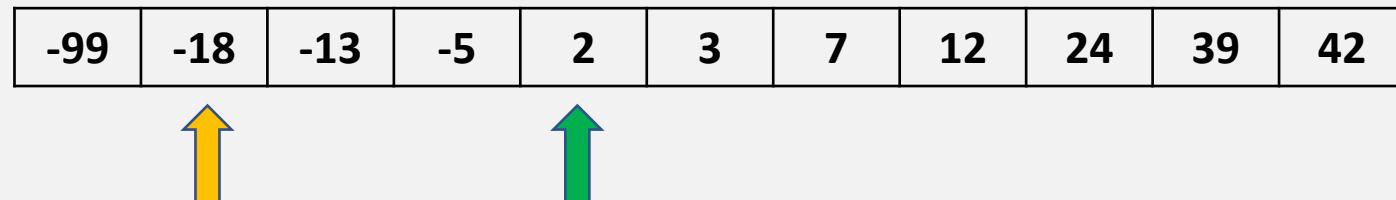
↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

-18 > -13 => **faux**

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -5 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-5 > 2 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > 3 \Rightarrow \text{faux}$

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

STOP

-99	-18	-13	-5	2	3	7	12	24	39	42
-----	-----	-----	----	---	---	---	----	----	----	----



↑ indiceCourant

↑ indiceMax

Tri à bulles

```
entier[] TriBulles(entier[] p_valeurs) {
    entier ancienneValeur = 0;
    booleen permutationAuDernierTour = vrai;
    entier indiceMax = p_valeurs.Capacité - 1;
    entier[] valeursCopiees = CopierTableau(p_valeurs);

    tant que (permutationAuDernierTour) {
        permutationAuDernierTour = faux;
        pour entier indiceCourant de 0 à indiceMax - 1 faire {
            si (valeursCopiees[indiceCourant + 1] < valeursCopiees[indiceCourant]) alors {
                ancienneValeur = valeursCopiees[indiceCourant + 1];
                valeursCopiees[indiceCourant + 1] = valeursCopiees[indiceCourant];
                valeursCopiees[indiceCourant] = ancienneValeur;
                permutationAuDernierTour = vrai;
            }
        }
        indiceMax = indiceMax - 1;
    }

    renvoyer valeursCopiees;
}
```

Tri rapide : diviser pour régner

- But : diviser un problème complexe en sous-problèmes moins complexes
- Méthode :
 - Diviser : diviser un problème initial en sous-problèmes
 - Régner : résoudre les sous-problèmes
 - Combiner : à partir des résolutions des sous-problèmes produire la solution du problème initial

Diviser pour régner – Tri rapide – Diviser

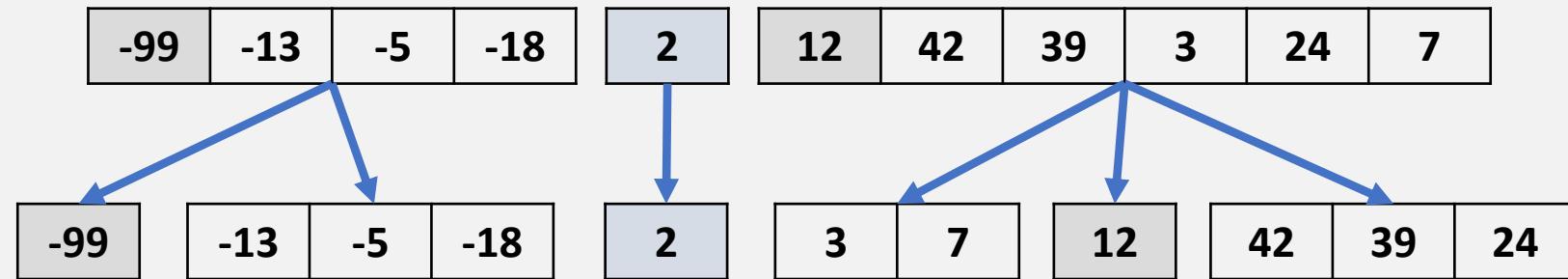
- Comment diviser un tableau ?
 - Idée : déterminer une valeur dite pivot et créer deux tableaux tels que :
 - $\forall \text{ élément} \in \text{partition1}, \text{élément} \leq \text{valeurPivot}$
 - $\forall \text{ élément} \in \text{partition2}, \text{élément} > \text{valeurPivot}$
 - Exemple :

2	12	-99	-13	-5	42	-18	39	3	24	7
---	----	-----	-----	----	----	-----	----	---	----	---

⇒

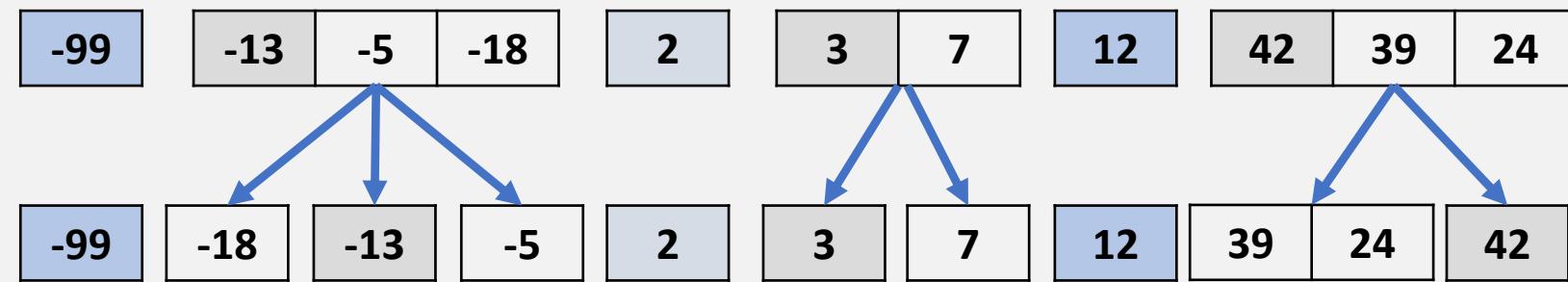
-99	-13	-5	-18	2	12	42	39	3	24	7
-----	-----	----	-----	---	----	----	----	---	----	---

Diviser pour régner – Tri rapide

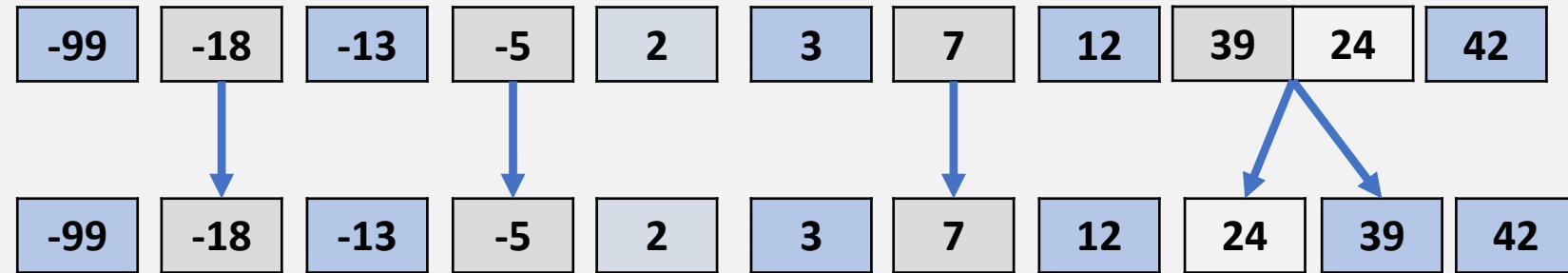


Diviser pour régner – Tri rapide

-99	-13	-5	-18	2	12	42	39	3	24	7
-----	-----	----	-----	---	----	----	----	---	----	---



Diviser pour régner – Tri rapide



Diviser pour régner – Tri rapide



Diviser pour régner – Tri rapide



Pseudocode pour les tableaux

```
entier[] TriRapide(entier[] p_valeurs) {
    entier[] valeursCopiees = CopierTableau(p_valeurs);
    TriRapide_rec(valeursCopiees, 0, valeursCopiees.Capacité - 1);

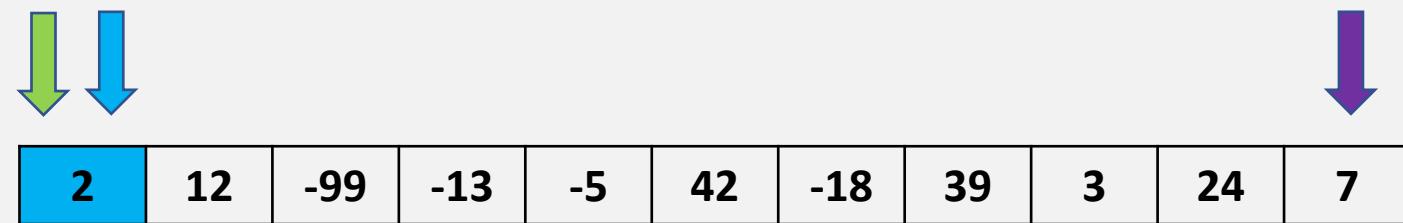
    renvoyer valeursCopiees;
}

aucun TriRapide_rec(entier[] p_valeurs, entier p_indicePremier, entier p_indiceDernier) {
    entier indicePivot = 0;
    si (p_indicePremier < p_indiceDernier) alors {
        indicePivot = ChoixPivot(p_valeurs, p_indicePremier, p_indiceDernier);
        indicePivot = Partitionner(p_valeurs, p_indicePremier, p_indiceDernier, indicePivot);
        TriRapide_rec(p_valeurs, p_indicePremier, indicePivot - 1);
        TriRapide_rec(p_valeurs, indicePivot + 1, p_indiceDernier);
    }
}
```

Pseudocode pour les tableaux

```
entier ChoixPivot(entier[] p_valeurs, entier p_indicePremier, entier p_indiceDernier) {
    renvoyer p_indicePremier;
}
```

Partitionnement optimisé



↓ indicePivot

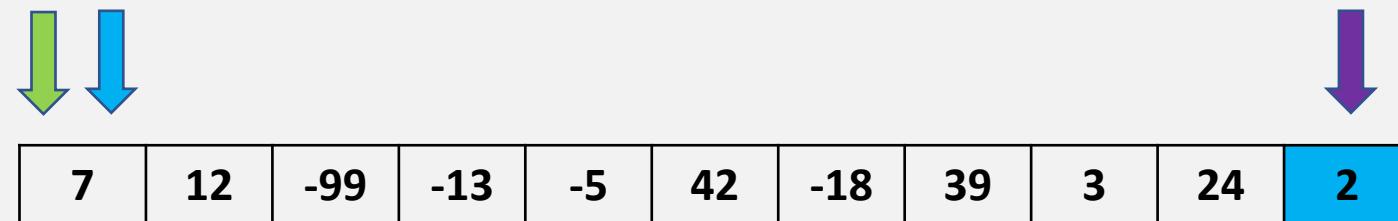
↓ indicePremier

↓ indiceDernier

↑ indiceValeurARanger

Partitionnement optimisé

$7 \leq 2 \Rightarrow \text{faux}$



↓ indicePivot

↓ indicePremier

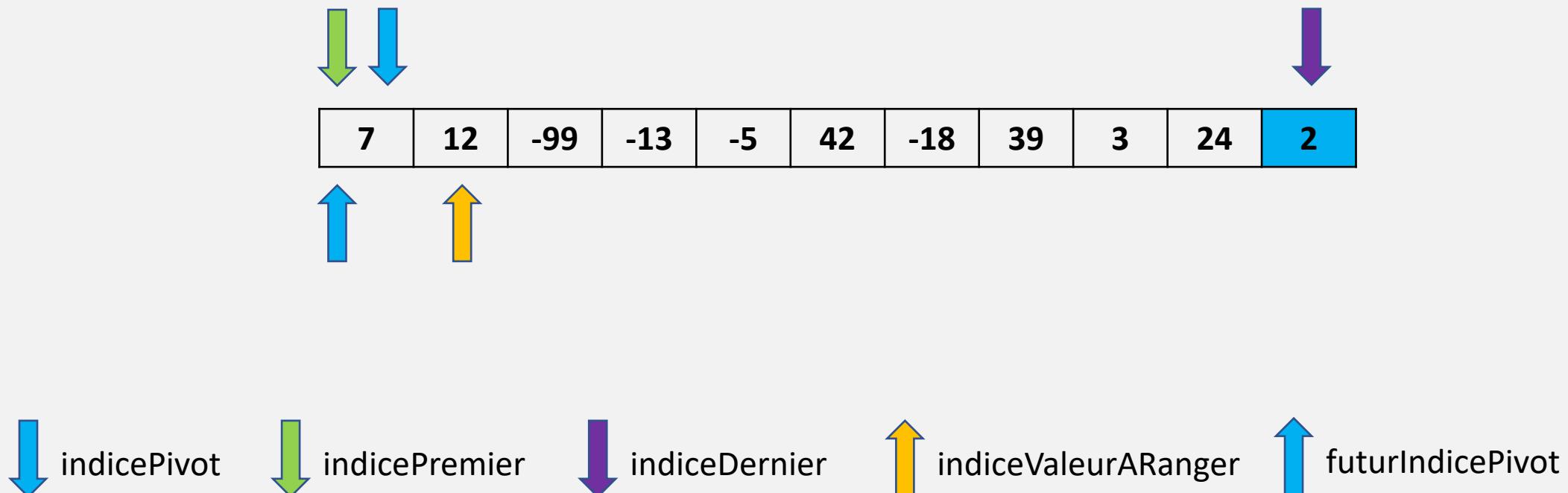
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

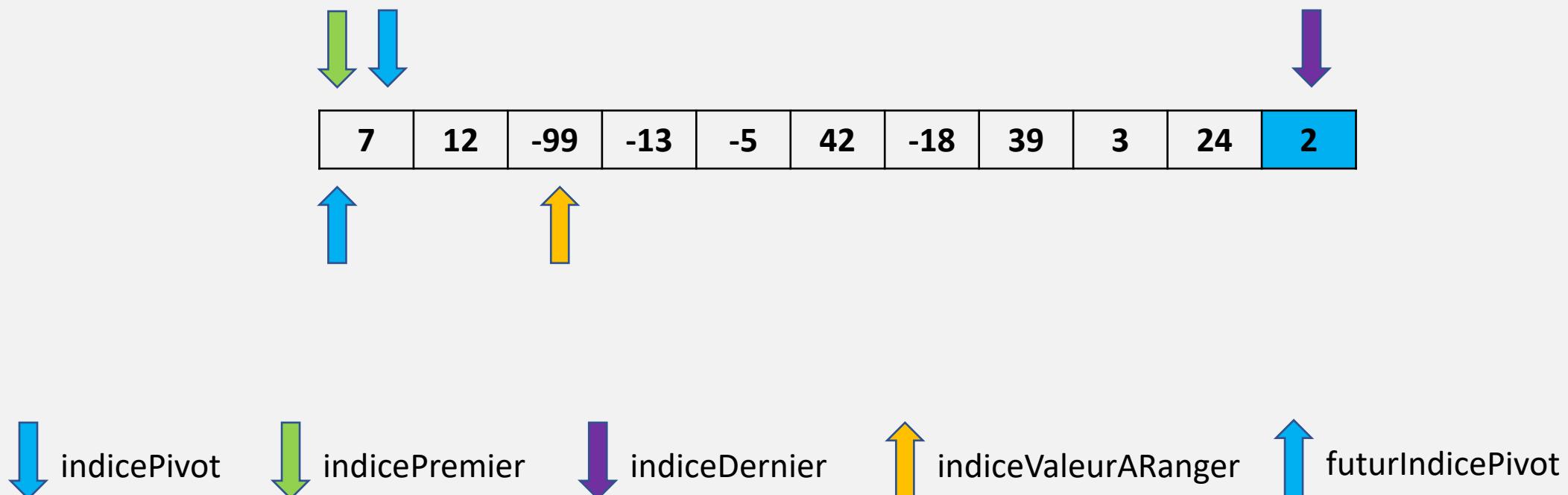
Partitionnement optimisé

$12 \leq 2 \Rightarrow \text{faux}$



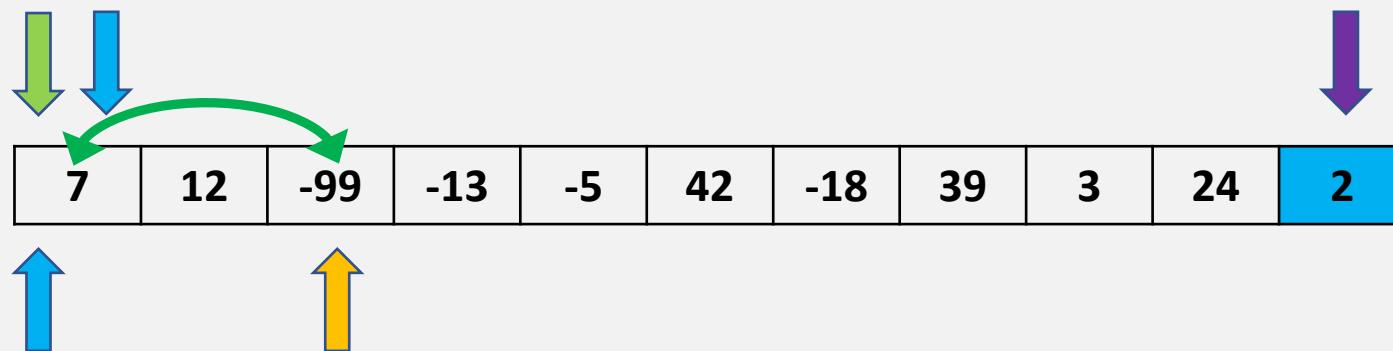
Partitionnement optimisé

$-99 \leq 2 \Rightarrow \text{vrai}$



Partitionnement optimisé

Échanger les valeurs



↓ indicePivot

↓ indicePremier

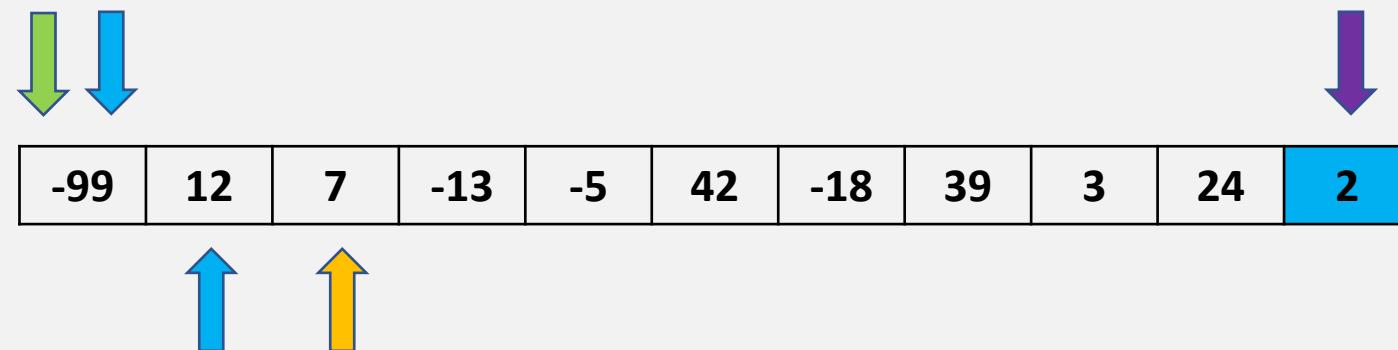
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Avancer futurIndicePivot



↓ indicePivot

↓ indicePremier

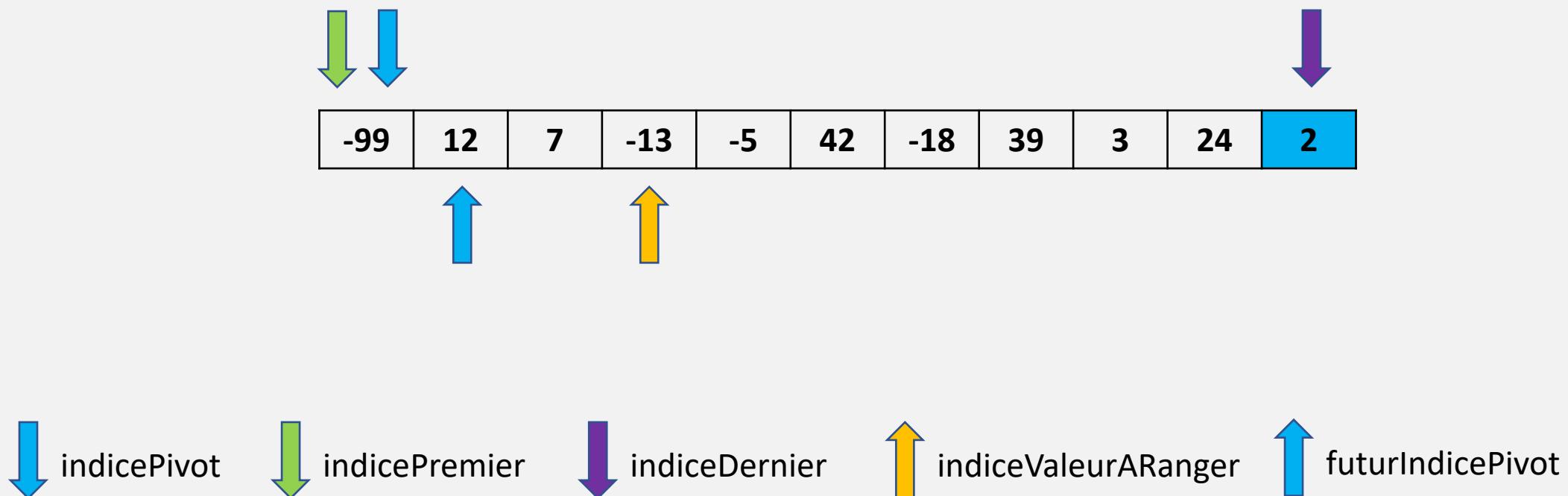
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

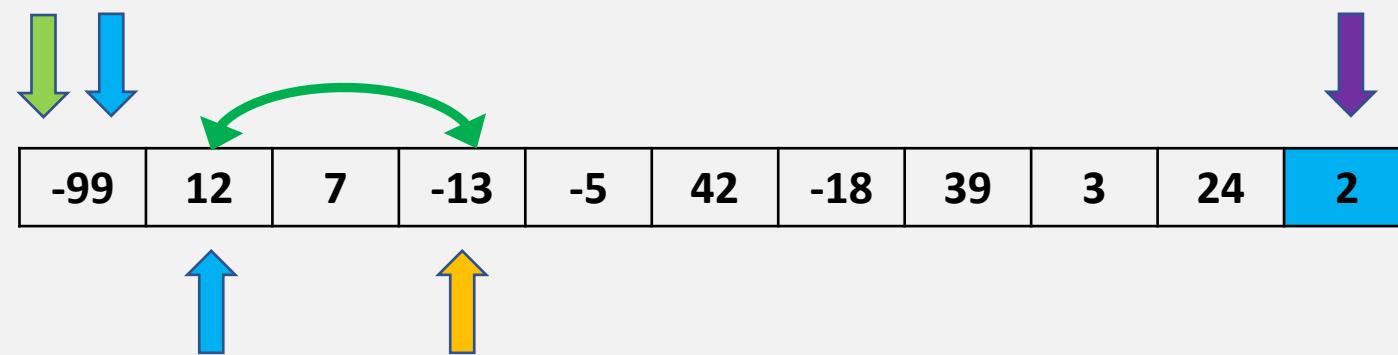
Partitionnement optimisé

$-13 \leq 2 \Rightarrow \text{vrai}$



Partitionnement optimisé

Échanger les valeurs



↓ indicePivot

↓ indicePremier

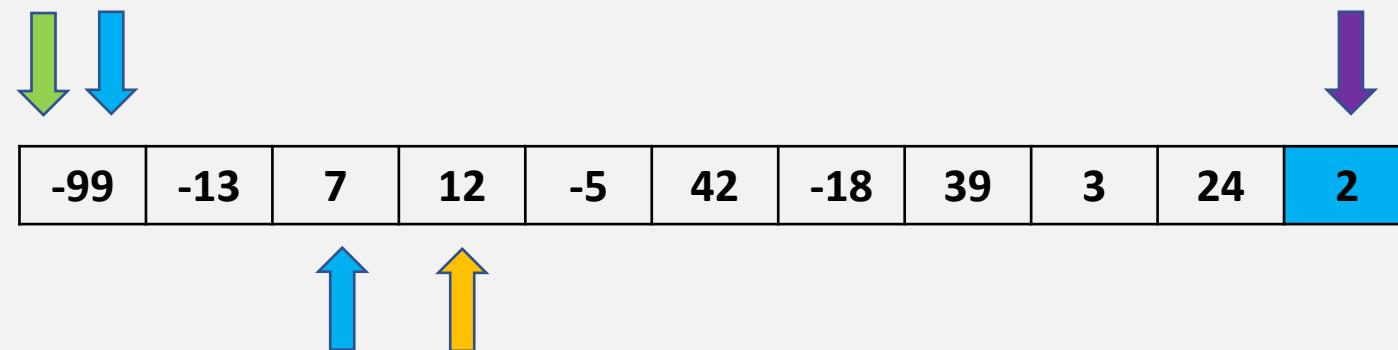
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Avancer futurIndicePivot



↓ indicePivot

↓ indicePremier

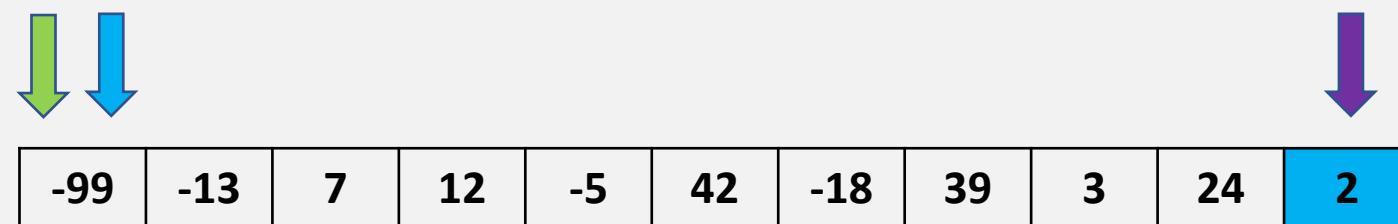
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

$-5 \leq 2 \Rightarrow \text{vrai}$



↓ indicePivot

↓ indicePremier

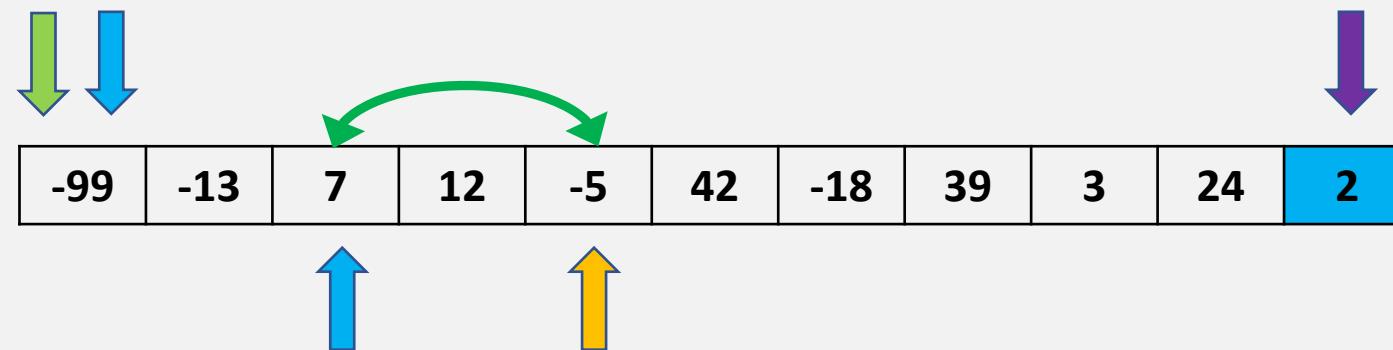
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Échanger les valeurs



↓ indicePivot

↓ indicePremier

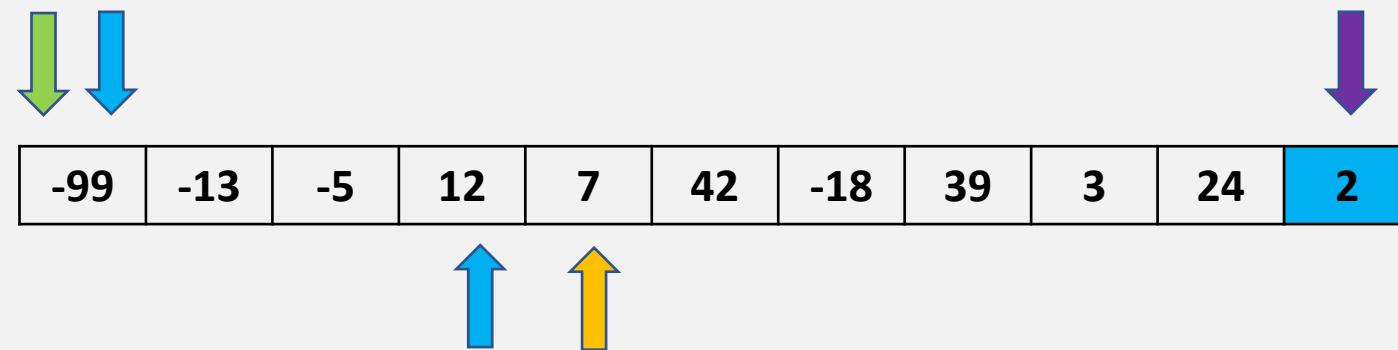
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Avancer futurIndicePivot



↓ indicePivot

↓ indicePremier

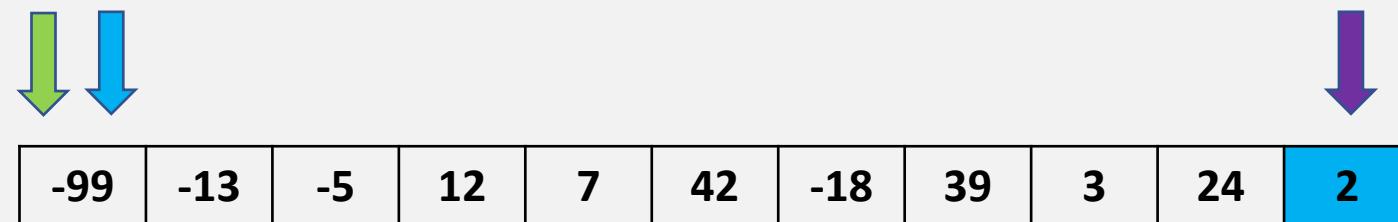
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

$7 \leq 2 \Rightarrow \text{faux}$



↓ indicePivot

↓ indicePremier

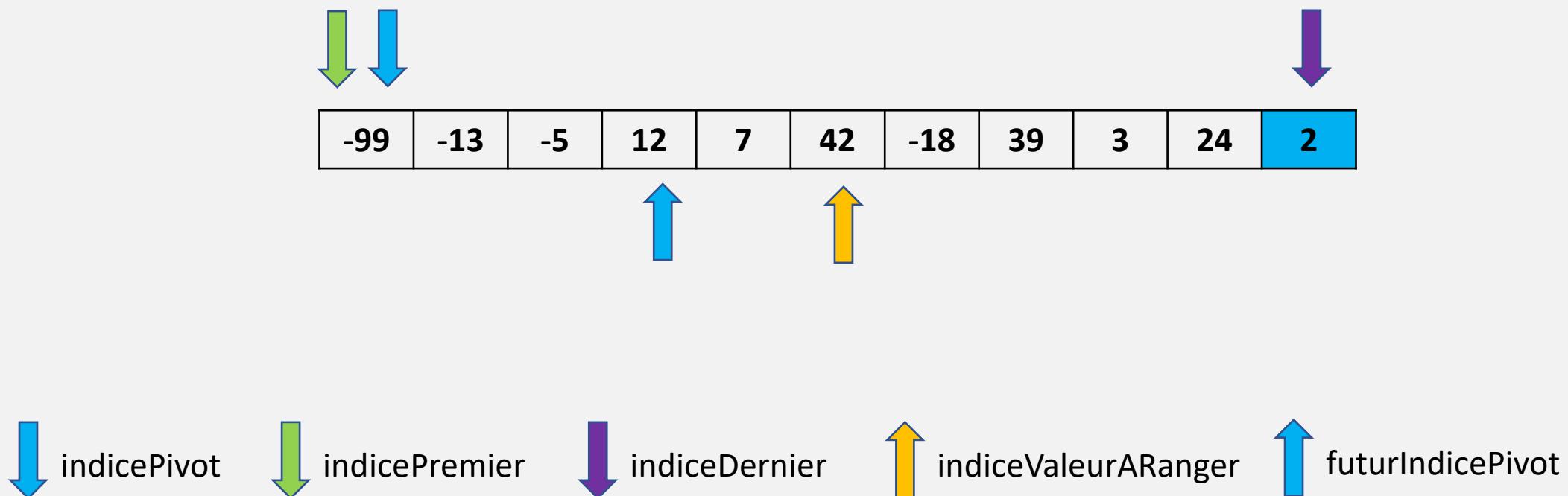
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

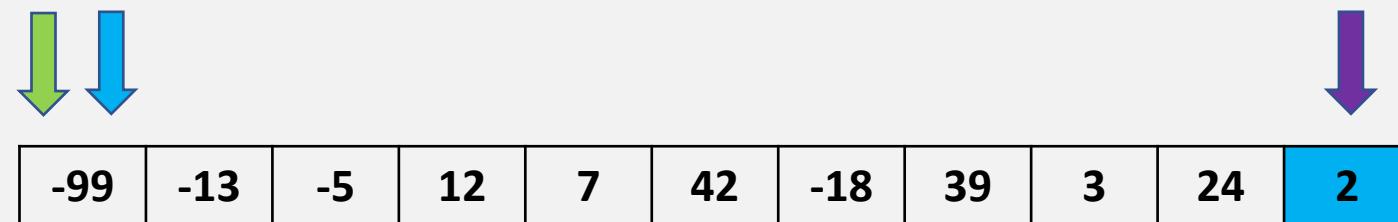
Partitionnement optimisé

$42 \leq 2 \Rightarrow \text{faux}$



Partitionnement optimisé

$-18 \leq 2 \Rightarrow \text{vrai}$



↓ indicePivot

↓ indicePremier

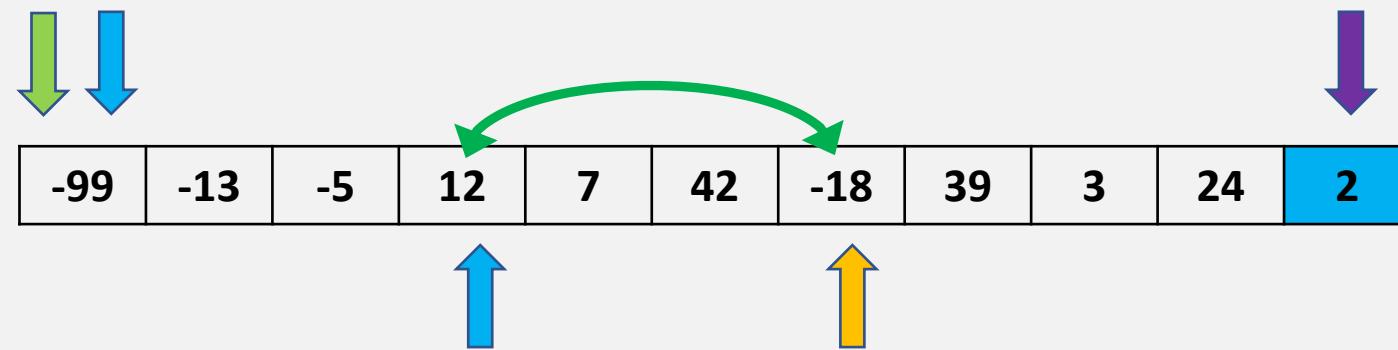
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Échanger les valeurs



↓ indicePivot

↓ indicePremier

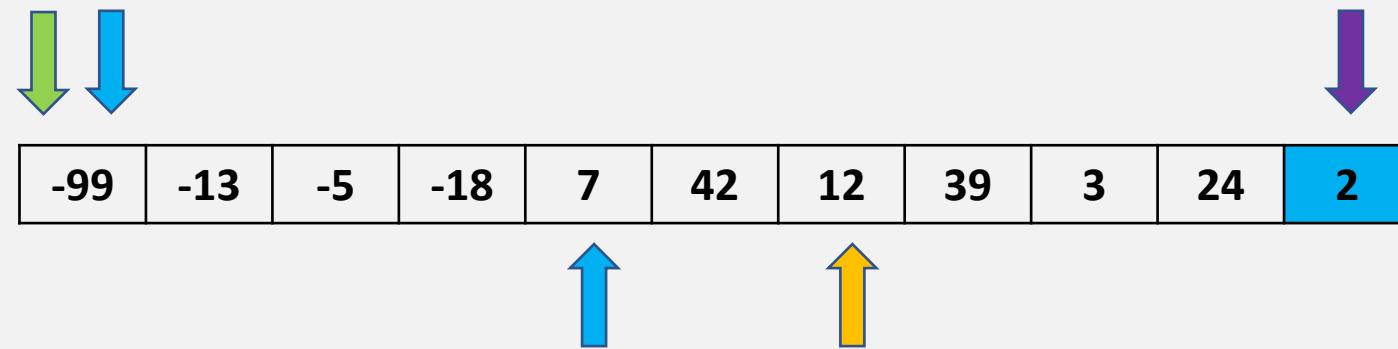
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Avancer futurIndicePivot



↓ indicePivot

↓ indicePremier

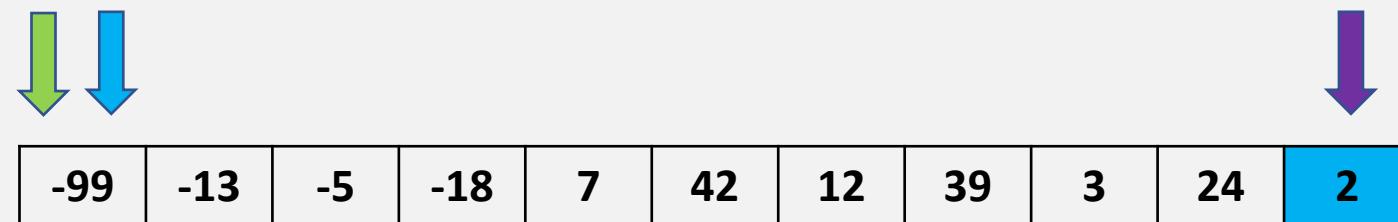
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

$39 \leq 2 \Rightarrow \text{faux}$



↓ indicePivot

↓ indicePremier

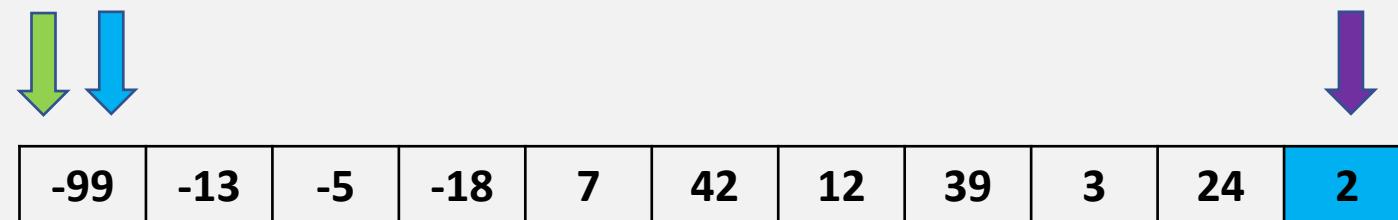
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

$3 \leq 2 \Rightarrow \text{faux}$



↓ indicePivot

↓ indicePremier

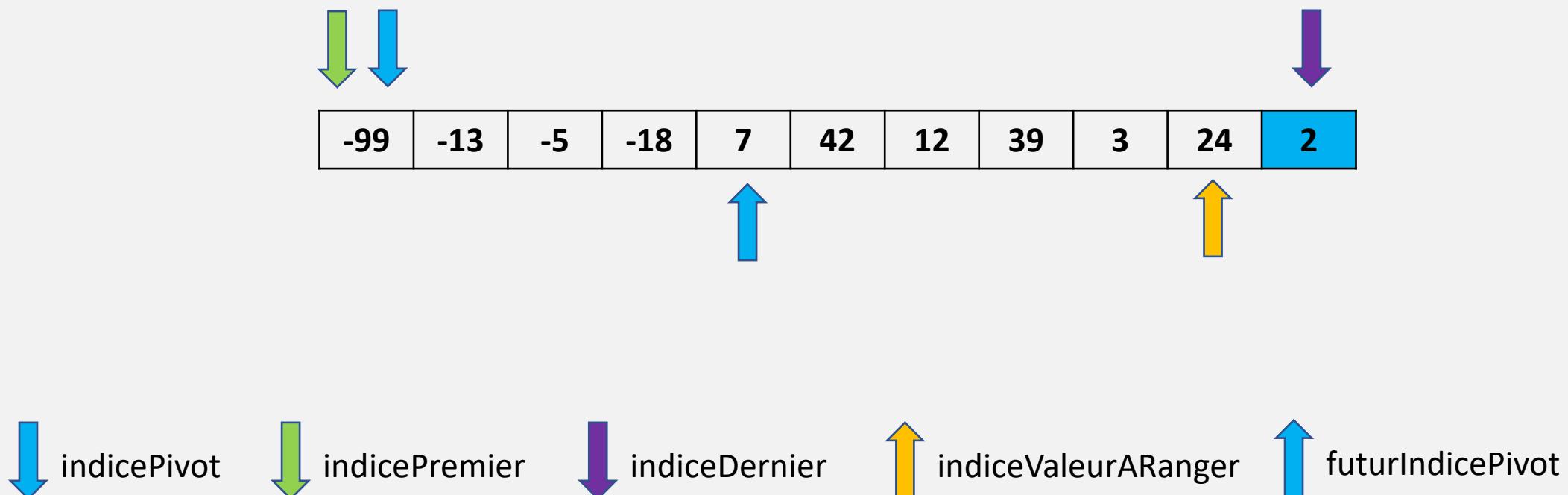
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

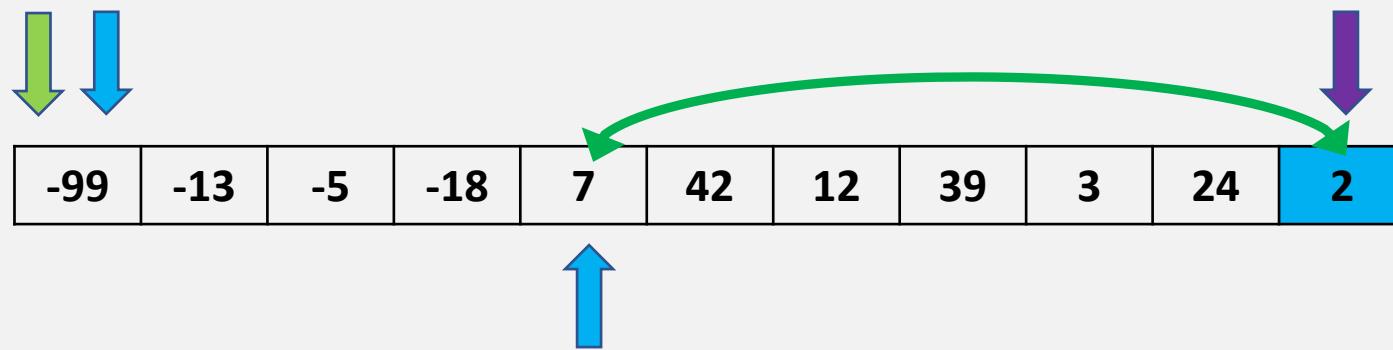
Partitionnement optimisé

$24 \leq 2 \Rightarrow \text{faux}$



Partitionnement optimisé

Échanger la valeur du pivot avec celle du futurPivot



↓ indicePivot

↓ indicePremier

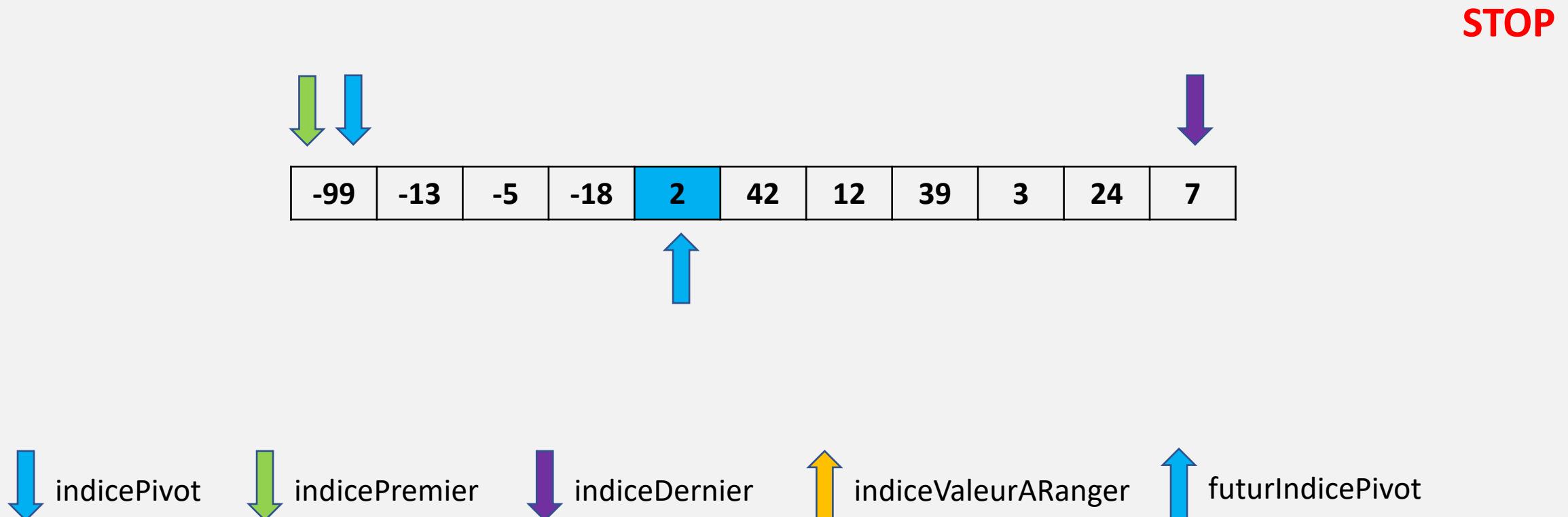
↓ indiceDernier

↑ indiceValeurARanger

↑ futurIndicePivot

Partitionnement optimisé

Échanger la valeur du pivot avec celle du futurPivot



Version optimisée pour les tableaux

```
entier Partitionner(entier[] p_valeurs, entier p_indicePremier, entier p_indiceDernier, entier p_indicePivot) {
    entier ancienneValeur = 0;
    entier futurIndicePivot = p_indicePremier;

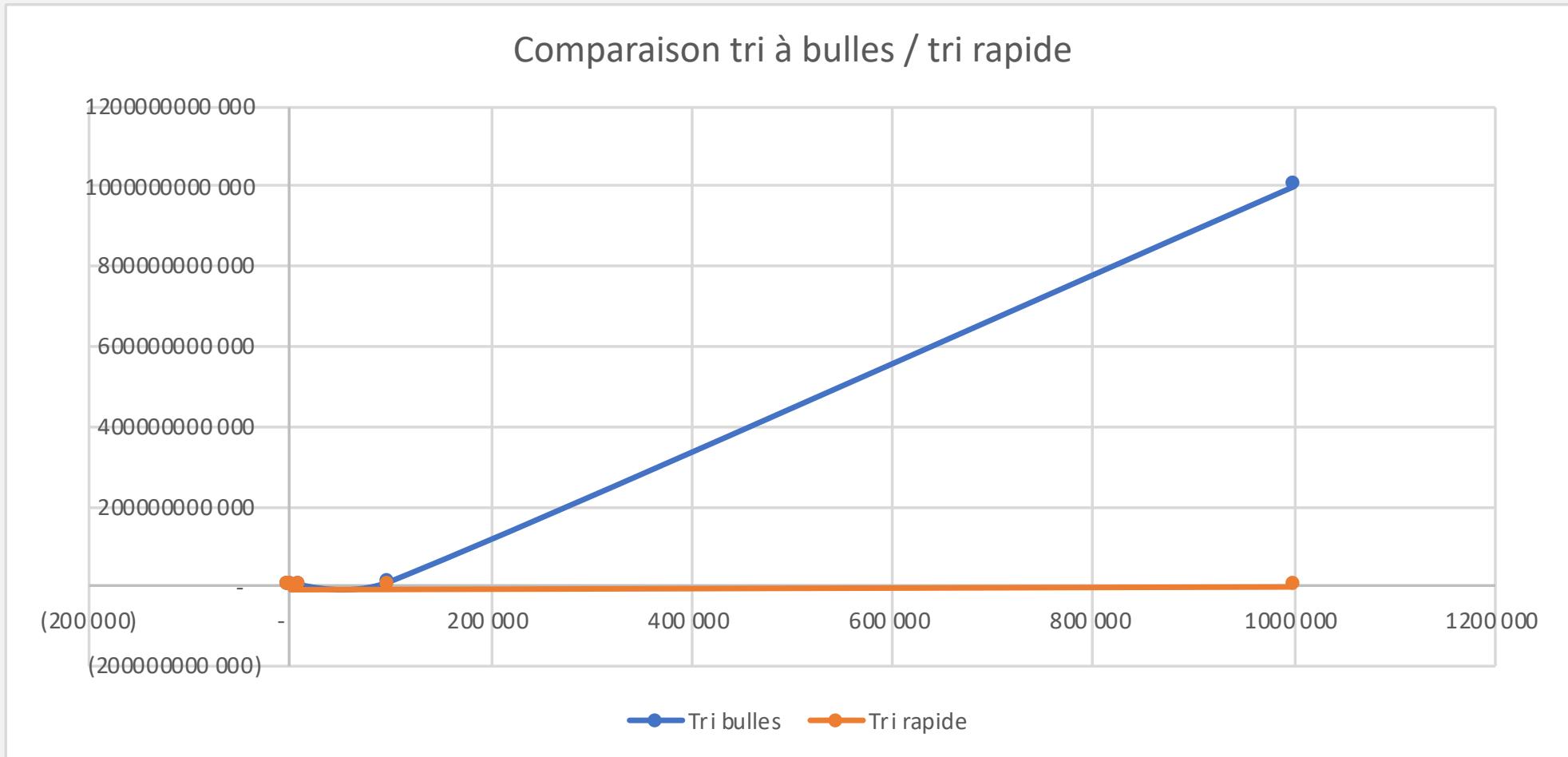
    ancienneValeur = p_valeurs[p_indicePivot];
    p_valeurs[p_indicePivot] = p_valeurs[p_indiceDernier];
    p_valeurs[p_indiceDernier] = ancienneValeur;

    pour entier indiceValeurARanger de p_indicePremier à p_indiceDernier - 1 {
        si p_valeurs[indiceValeurARanger] <= p_valeurs[p_indiceDernier] alors {
            ancienneValeur = p_valeurs[futurIndicePivot];
            p_valeurs[futurIndicePivot] = p_valeurs[indiceValeurARanger];
            p_valeurs[indiceValeurARanger] = ancienneValeur;
            futurIndicePivot = futurIndicePivot + 1;
        }
    }

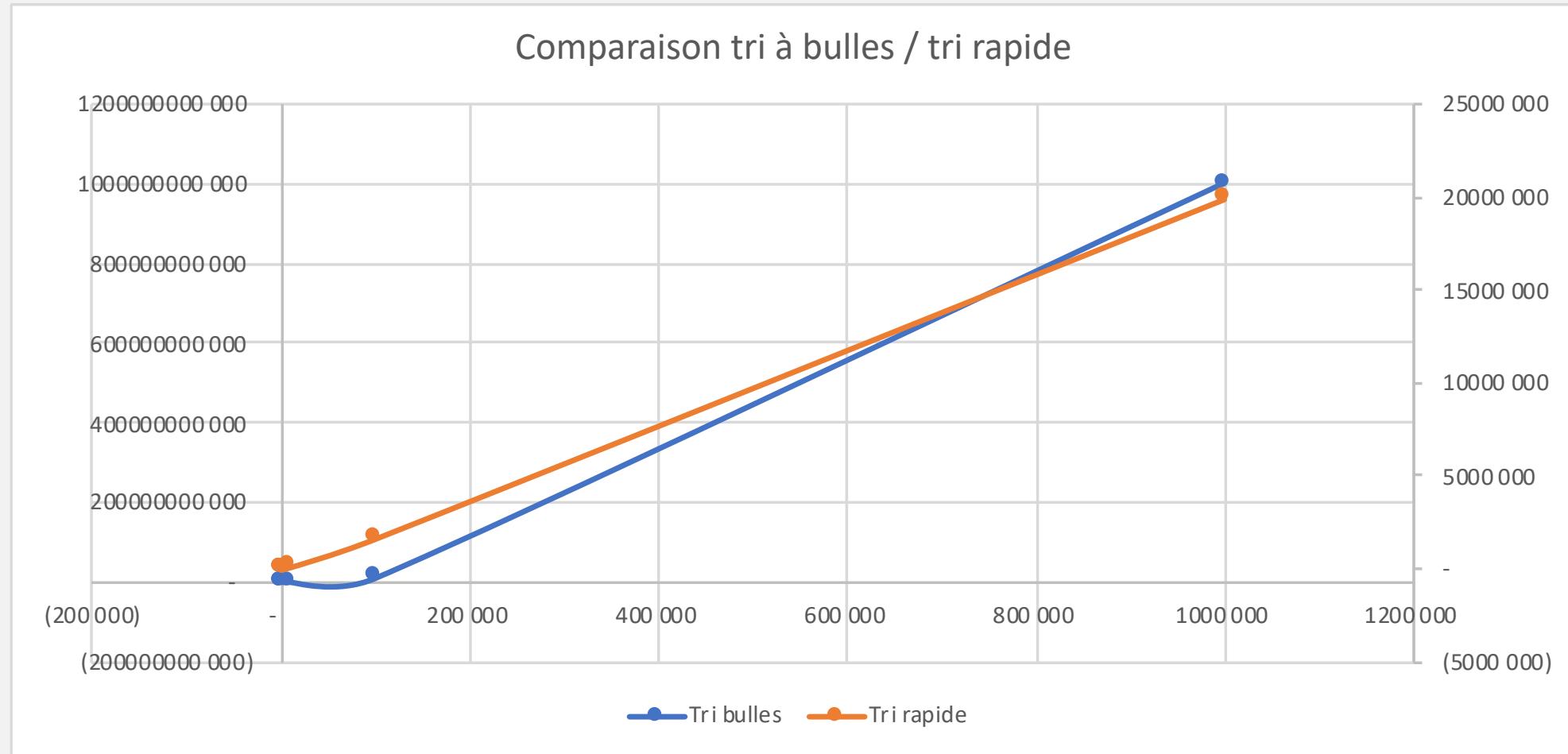
    ancienneValeur = p_valeurs[futurIndicePivot];
    p_valeurs[futurIndicePivot] = p_valeurs[p_indiceDernier];
    p_valeurs[p_indiceDernier] = ancienneValeur;

    renvoyer futurIndicePivot;
}
```

Intuition sur ces deux algorithmes



Intuition sur ces deux algorithmes – Avec deux axes



Algorithme intuitif V. 1 – Rappels

```
booleen RechercherValeur(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    pour entier indiceValeurCourante de 0 à p_collection.Capacite - 1 faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
    }
    renvoyer estTrouvee;
}
```

Algorithme intuitif V. 1 – Rappels

```
boolean RechercherValeur(int[] p_collection, int p_valeurAChercher) {
    boolean estTrouvee = faux;
    pour entier indiceValeurCourante de 0 à p_collection.Capacite - 1 faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
    }
    renvoyer estTrouvee;
}
```

Au maximum 22 comparaisons pour un tableau de 11 éléments

Dans le pire des cas on fera 22 comparaisons donc $2 * n$ comparaisons, n étant le nombre d'éléments.

En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * n$, k étant une constante.

On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(n)**.

Algorithme intuitif V. 2 – Rappels

```
booleen RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    entier indiceValeurCourante = 0;
    tant que (non estTrouvee et indiceValeurCourante < p_collection.Capacité) faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
        ++indiceValeurCourante;
    }

    renvoyer estTrouvee;
}
```

Algorithme intuitif V. 2 – Rappels

```
booleen RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {
    booleen estTrouvee = faux;
    entier indiceValeurCourante = 0;
    tant que (non estTrouvee et indiceValeurCourante < p_collection.Capacité) faire {
        si (p_collection[indiceValeurCourante] == p_valeurAChercher) {
            estTrouvee = vrai;
        }
        ++indiceValeurCourante;
    }

    renvoyer estTrouvee;
}
```

Au maximum 33 comparaisons pour un tableau de 11 éléments

Dans le pire des cas on fera 33 comparaisons donc $3 * n$ comparaisons, n étant le nombre d'éléments.

En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * n$, k étant une constante.

On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(n)**.

Principes de la recherche dichotomique

- La recherche dichotomique, ou recherche par dichotomie (en anglais : binary search), est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié.
- Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.

Principes de la recherche dichotomique

Soit le tableau trié suivant :

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -5

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -5

$3 == -5 \Rightarrow \text{faux}$

$3 < -5 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 10) / 2 \Rightarrow 5$$

 indicePremier

 indiceDernier

 indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -5

$3 == -5 \Rightarrow \text{faux}$

$3 < -5 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 5 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -5

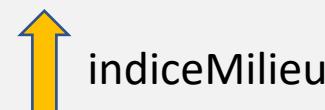
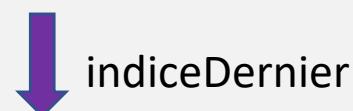
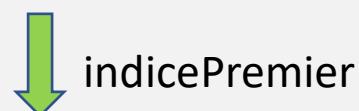
$-13 == -5 \Rightarrow \text{faux}$

$-13 < -5 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 4) / 2 \Rightarrow 2$$



Principes de la recherche dichotomique

Recherche de -5

$-13 == -5 \Rightarrow \text{faux}$

$-13 < -5 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 2 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -5

$-5 == -5 \Rightarrow \text{vrai}$

valeur trouvée !

$4 * 2 + 2 = 10$ comparaisons

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (3 + 4) / 2 \Rightarrow 3$$

indicePremier

indiceDernier

indiceMilieu

2 comparaisons

Principes de la recherche dichotomique

Recherche de -3

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Principes de la recherche dichotomique

Recherche de -3

$3 == -3 \Rightarrow \text{faux}$

$3 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 10) / 2 \Rightarrow 5$$

 indicePremier

 indiceDernier

 indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$3 == -3 \Rightarrow \text{faux}$

$3 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



Recherche réduite à un sous-tableau de 5 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

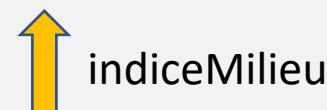
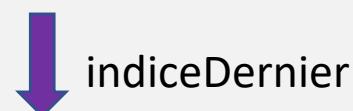
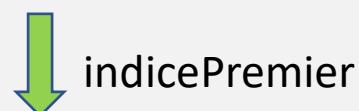
$-13 == -3 \Rightarrow \text{faux}$

$-13 < -3 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



$$\text{indiceMilieu} = (0 + 4) / 2 \Rightarrow 2$$



Principes de la recherche dichotomique

Recherche de -3

$-13 == -3 \Rightarrow \text{faux}$

$-13 < -3 \Rightarrow \text{vrai}$

The diagram illustrates a binary search algorithm. At the top, there are two green arrows pointing downwards, one above the other. Below them is a 2x11 grid representing an array of 11 elements. The first row contains indices from 0 to 10. The second row contains the corresponding values: -99, -18, -13, -5, 2, 3, 7, 12, 24, 39, and 42. The value -13 is located at index 4. A yellow arrow points upwards from the bottom of the array grid towards the middle of the array.

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Recherche réduite à un sous-tableau de 2 éléments



indicePremier



indiceDernier



indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$-5 == -3 \Rightarrow \text{faux}$

$-5 < -3 \Rightarrow \text{vrai}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

Diagram showing the search space: a 2x11 grid where the first row contains indices 0 to 10 and the second row contains values at those indices. A green arrow points down to index 3, a purple arrow points down to index 4, and a yellow arrow points up to index 3.5.

$$\text{indiceMilieu} = (3 + 4) / 2 \Rightarrow 3$$

↓ indicePremier ↓ indiceDernier ↑ indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$-5 == -3 \Rightarrow \text{faux}$

$-5 < -3 \Rightarrow \text{vrai}$

The diagram illustrates a binary search algorithm. At the top, two green arrows point downwards from the search value '-3' towards the array. Below the array, a yellow arrow points upwards to the midpoint element '2'. A green box at the bottom contains the text 'Recherche réduite à un sous-tableau de 1 élément'.

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

indicePremier

indiceDernier

indiceMilieu

Principes de la recherche dichotomique

Recherche de -3

$2 == -3 \Rightarrow \text{faux}$

$2 < -3 \Rightarrow \text{faux}$

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42

$$\text{indiceMilieu} = (4 + 4) / 2 \Rightarrow 4$$

indicePremier

indiceDernier

indiceMilieu

4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

indicePremier <= indiceDernier => **faux**

STOP => Non trouvée !

4 * 4 comparaisons

0	1	2	3	4	5	6	7	8	9	10
-99	-18	-13	-5	2	3	7	12	24	39	42



indicePremier

indiceDernier

indiceMilieu

Algorithme de recherche dichotomique

```
boolean RechercherValeurDichotomie(int p_collection, int p_valeurAChercher) {
    boolean estTrouvee = faux;
    entier indicePremier = 0;
    entier indiceDernier = p_collection.Capacité - 1;
    entier indiceMilieu = 0;

    tant que (non estTrouvee et indicePremier <= indiceDernier) {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        si (p_collection[indiceMilieu] == p_valeurAChercher) {
            estTrouvee = vrai;
        } sinon si (p_collection[indiceMilieu] < p_valeurAChercher) {
            indicePremier = indiceMilieu + 1;
        } sinon {
            indiceDernier = indiceMilieu - 1;
        }
    }

    renvoyer estTrouvee;
}
```

Algorithme de recherche dichotomique

- À chaque tour de boucle, on divise la taille de notre problème par 2
- Si n est le nombre d'éléments et k représente la $k^{\text{ième}}$ division, le problème est successivement de taille :
 - n ($n / 2^0$, tour 1)
 - $n/2$ ($n / 2^1$, tour 2)
 - $n/4$ ($n / 2^2$, tour 3)
 - $n/8$ ($n / 2^3$, tour 4)
 - $n/16$ ($n / 2^4$, tour 5)
 - ...
 - $n/2^k$ (tour $k + 1$)

Algorithme de recherche dichotomique

- Dans le pire des cas on va arriver à un tableau d'un élément donc :
 - $1 \leq \frac{n}{2^k}$
 - $2^k \leq n$
 - $\log(2^k) \leq \log(n)$ ($\log(a^k) = k * \log(a)$)
 - $k * \log(2) \leq \log(n)$
 - $k \leq \frac{\log(n)}{\log(2)}$
 - $k \leq \log_2(n)$

Algorithme de recherche dichotomique

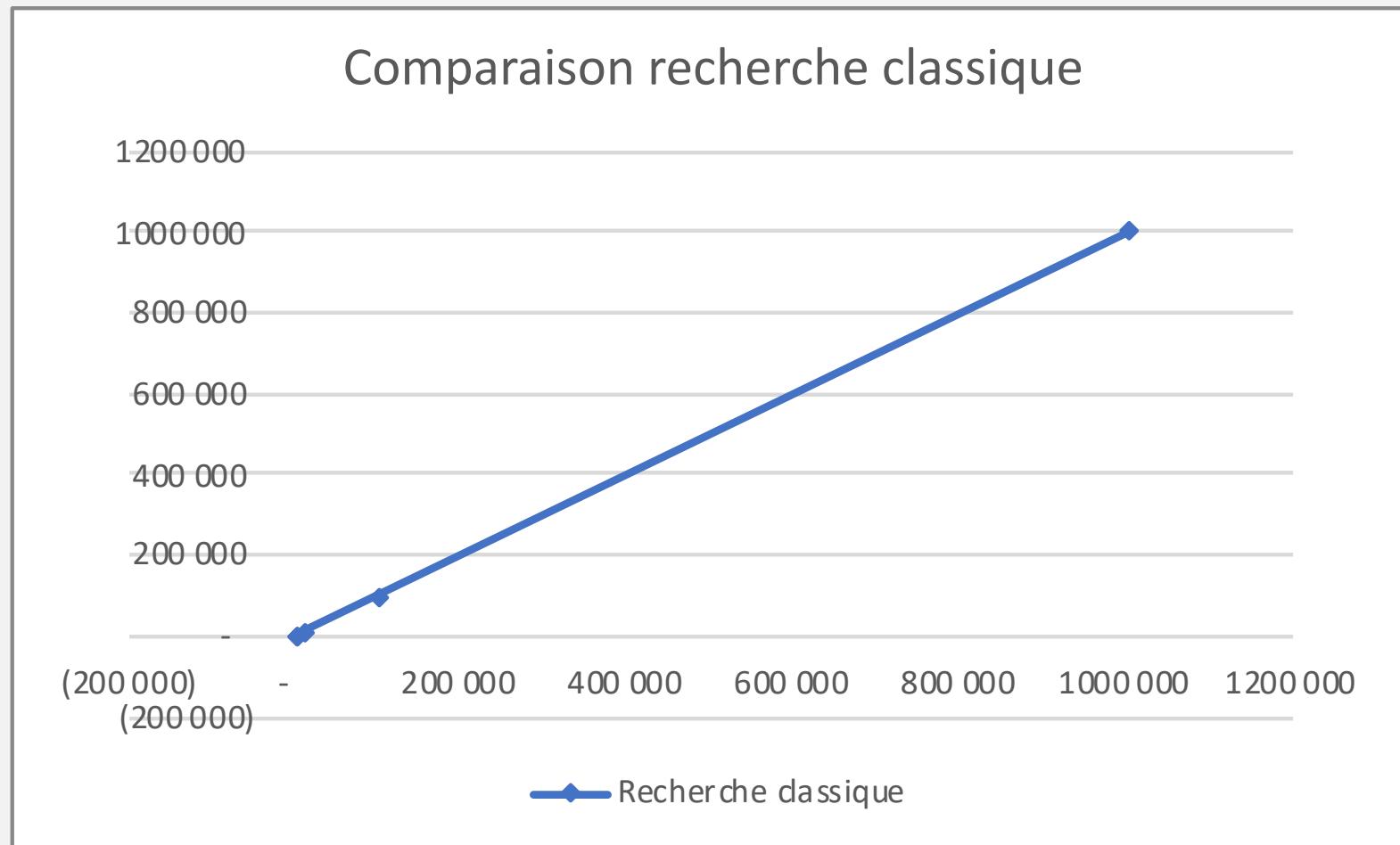
```
booléen RechercherValeurDichomie(int p_collection, int p_valeurAChercher) {
    booléen estTrouvée = faux;
    entier indicePremier = 0;
    entier indiceDernier = p_collection.Capacité - 1;
    entier indiceMilieu = 0;

    tant que (non estTrouvée et indicePremier <= indiceDernier) {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        si (p_collection[indiceMilieu] == p_valeurAChercher) {
            estTrouvée = vrai;
        } sinon si (p_collection[indiceMilieu] < p_valeurAChercher) {
            indicePremier = indiceMilieu + 1;
        } sinon {
            indiceDernier = indiceMilieu - 1;
        }
    }

    renvoyer estTrouvée;
}
```

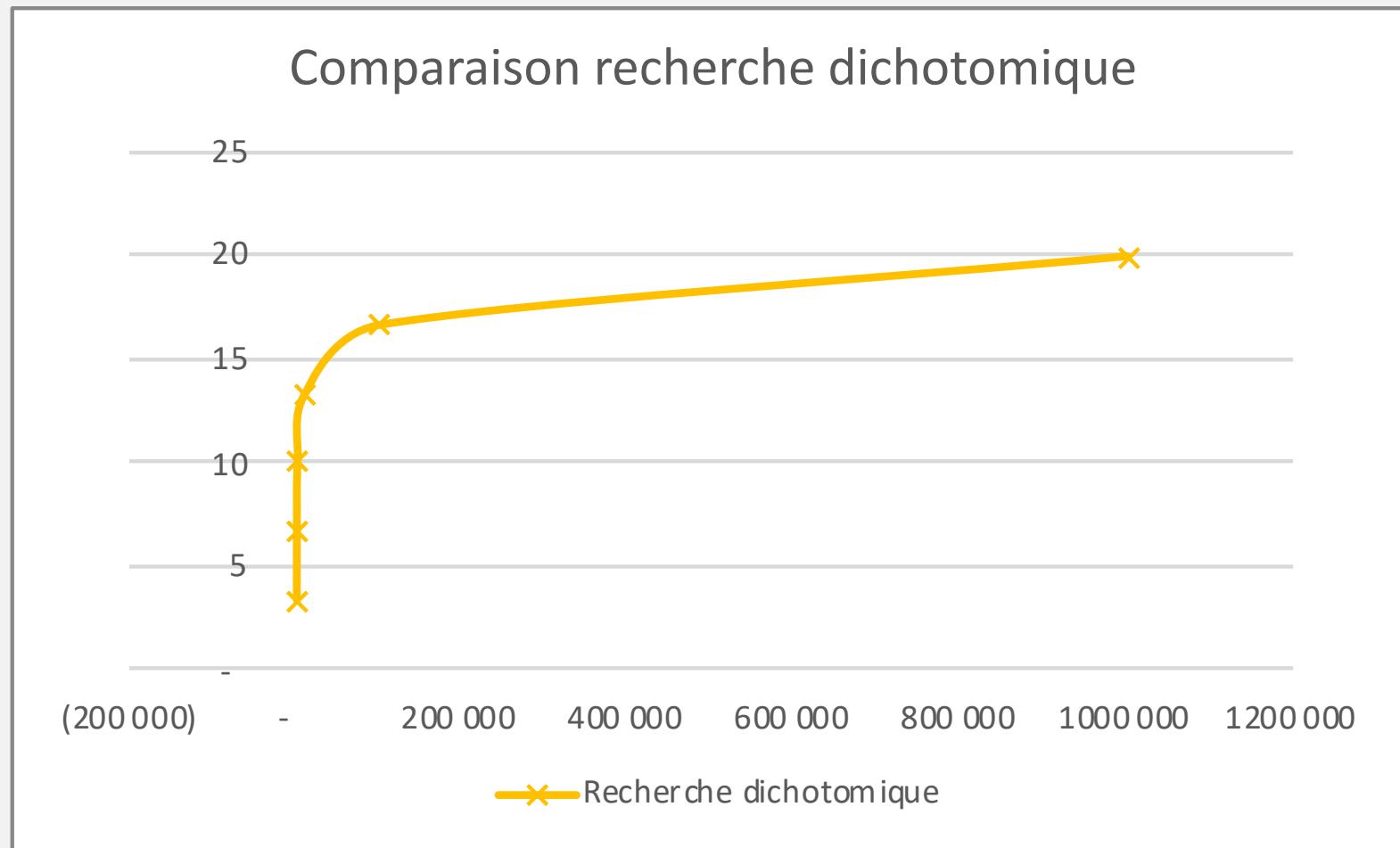
Dans le pire des cas on fera 16 comparaisons donc $4 * \log_2(n)$ comparaisons, n étant le nombre d'éléments. En **complexité algorithmique**, on généralise cela en disant que l'on a ici $k * \log_2(n)$, k étant une constante. On utilise ensuite la **notation O de Landau** et on dit que la complexité de l'algorithme est en **O(log₂(n))**.

Intuition sur la complexité des algorithmes



x : quantité de données
y : nombre de comparaisons

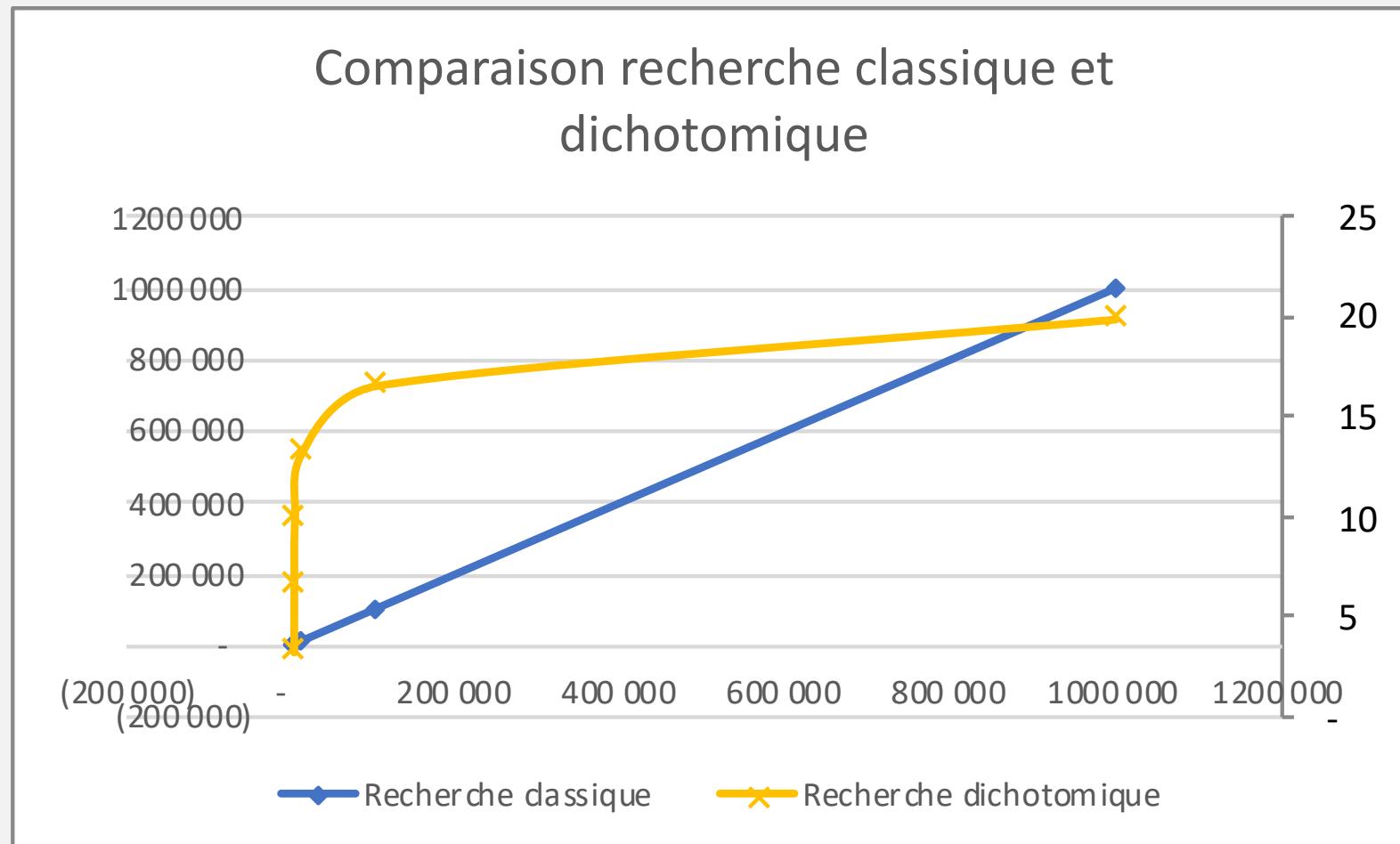
Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons

Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons