

The background of the slide is a close-up photograph of various colorful 3D mathematical symbols and letters scattered on a surface with a pink and light blue geometric pattern. The symbols include plus, minus, multiplication, division, and percentage signs in yellow, red, green, and blue, as well as letters like 'a', 'n', and 'x' in red, yellow, and blue. A semi-transparent white rectangular box is centered over the image, containing the title text.

Module02 – Langage C++ Impératif

Objectifs

- Découvrir les fonctionnalités de base de C++
 - Alternatives / répétitions
 - Types de données
 - Fonctions/lambda
 - Organisation des fichiers
- Être capable de faire une partie de ce que vous savez faire dans d'autres langages en C++
- Pratiquer sur des exemples simples (Recherche / Tri / etc.)

Syntaxe – Instructions et blocs

- Les **instructions** sont séparées par des « ; »
 - `int` unNombre = 0;
- Un **bloc** permet de grouper des **plusieurs instructions** qui s'exécutent dans un même **contexte**.
- Pour définir un bloc, on l'encadre par une pair d'accolades « { *instruction; ...; instruction;* } »

```
{  
    int unNombre = 0;  
    int unNombrePlus1 = unNombre + 1;  
}
```

Syntaxe – Alternatives – SI ... (if)

```
if (<predicat>
    <instruction> / <bloc>
[else
    <instruction> / <bloc>]
```

Donc formes possibles :

```
if (<predicat>
    <instruction> / <bloc>
```

```
if (<predicat>
    <instruction> / <bloc>
else
    <instruction> / <bloc>
```

Syntaxe – Alternatives – SI ... (if) – Exemples

```
bool estNombreNegatif = false;  
if (nombre < 0)  
{  
    estNombreNegatif = true;  
}
```

```
bool estNombreNegatif = nombre < 0;
```

```
bool estNombrePair;  
if (nombre % 2 == 0)  
{  
    estNombrePair = true;  
}  
else  
{  
    estNombrePair = false;  
}
```

```
bool estNombrePair = nombre % 2 == 0;
```

Normes : toujours utiliser des blocs pour les alternatives et les répétitions

Syntaxe – Alternatives – SELON (switch)

```
switch (expression)
{
    case <valeur constante>:
        [<instructions> | <bloc>]*
        [break;]
    [case <valeur constante>:
        [<instructions> | <bloc>]*
        [break;]
    [default:
        [<instructions> | <bloc>]*
        [break;]]
}
```

break : cette instruction permet de sortir du bloc courant, sinon le programme continue à exécuter les instructions suivantes même si elles ne sont pas dans le même cas

Normes : toujours mettre un cas par défaut comme dernier cas, même s'il est vide

Syntaxe – Alternatives – SELON (switch) – Exemple

```
switch (c)
{
    case 'A':
        maja = maja + 1;
        break;
    case 'a':
        mina = mina + 1;
        break;
    default:
        nona = nona + 1;
}
```


Exemple d'interprétation :

```
if (c == 'A')
{
    maja = maja + 1;
}
else if (c == 'a')
{ // *
    mina = mina + 1;
}
else
{
    nona = nona + 1;
}
```

Normes : cours pas de début de bloc pour le if après le sinon, mais obligatoire pour le alors

Syntaxe – Alternatives – SELON (switch) – Exemple sans break

```
switch (c)
{
    case 'A':
        maja = maja + 1;
    case 'a':
        mina = mina + 1;
    default:
        nona = nona + 1;
}
```



Exemple d'interprétation :

```
if (c == 'A')
{
    maja = maja + 1;
    mina = mina + 1;
    nona = nona + 1;
}
else if (c == 'a')
{
    mina = mina + 1;
    nona = nona + 1;
}
else
{
    nona = nona + 1;
}
```


Syntaxe – Répétition – TANT QUE ... (WHILE)

But : Exécuter les instructions **tant que** le prédicat est vrai

```
while (<predicat>)  
    <instruction> | <bloc>
```

```
int n = 1;  
int fact = 1;  
  
while (n < 10) {  
    fact *= n;  
    n = n + 1;  
}
```

Syntaxe – Répétition – FAIRE ... TANT QUE (DO ... While)

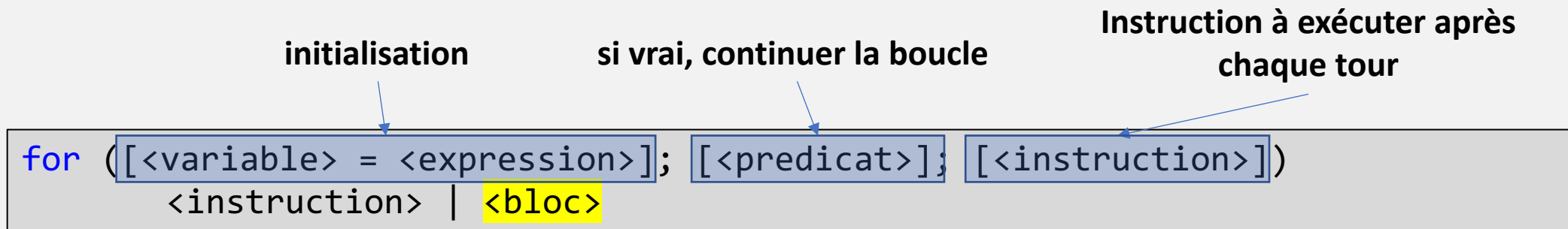
But : Exécuter les instructions **une fois et le refaire** tant que le prédicat est vrai

```
do  
    <instruction> | <bloc>  
while (<predicat>);
```

```
int n = 1;  
int fact = 1;  
  
do {  
    fact *= n;  
    n = n + 1;  
} while (n < 10);
```

Syntaxe – Répétition – POUR ... (FOR)

But : Exécuter les instructions **tant que** le prédicat est vrai avec une instruction de déclaration / initialisation et instruction d'incrément



```
int fact = 1;

for (int n = 1; n < 10; ++n)
{
    fact *= n;
}
```

Types de base – Visual studio 2022

Type Name	Bytes	Other Names	Range of Values
bool	1	none	false or true
char	1	none	-128 to 127 by default
unsigned char	1	none	0 to 255
short	2	short int, signed short int	-32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
int	4	signed	-2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
long	4	long int, signed long int	-2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
float	4	none	3.4E +/- 38 (7 digits)
long long	8	none (but equivalent to __int64)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	none (but equivalent to unsigned __int64)	0 to 18,446,744,073,709,551,615
double	8	none	1.7E +/- 308 (15 digits)
long double	same as double	none	Same as double
enum	varies	none	

unsigned <type> : le signe n'est pas représenté (i.e. c'est au développeur de le savoir) ce qui permet d'avoir un bit de plus.
sizeof(<type|variable>) : renvoie le nombre d'octets nécessaire pour une variable du type donné

Types de base – Entiers

- Pour simplifier certains codes, on peut utiliser des constantes entières écrites dans les bases 2 (binaire), 8 (octale) et 16 (hexadécimale)

```
int valeurAPartirDecimal = 254;  
int valeurAPartirBinaire = 0b11111110;  
int valeurAPartirOctal = 0376;  
int valeurAPartirHexadecimal = 0xFE;
```

- Vous pouvez utiliser un séparateur de chiffres. En C++, on utilise l'apostrophe « ' ». Exemple : **int** a = 12'345;

Types de base – Réels

- Par défaut, les valeurs réels sont considérées comme des « double »
- Pour spécifier le type « float », il faut suffixer la valeur par f

```
float valeurFloat = 254.0f;  
double valeurDouble = 3.1415;
```

Opérateurs de comparaisons

- Syntaxe : `<expression> <opérateur> <expression> -> booléen`
- Où `<opérateur>` :
 - `==` | `!=` : égale, différent
 - `<` | `<=` : inférieur, inférieur ou égale
 - `>` | `>=` : supérieur, supérieur ou égale
- Exemple : `42 > 13 ⇔ true`

Opérateurs booléens

- Syntaxe : `<expressionBool> <opérateur> <expressionBool> -> booléen`
- Où `<opérateur>` :
 - `||` : ou
 - `&&` : et
 - `!` : not
- Exemple : `a == 1 && a > 13 ⇔ false`
- L'évaluation du `||` et `&&` utilise la notion de court-circuit : la deuxième expression n'est évaluée que si on ne peut pas conclure avec la première

Opérateurs arithmétiques sur les entiers / réels

- Syntaxe : `<expression> <opérateur> <expression> -> entier / réel`
- Où `<opérateur>` :
 - `*` | `/` : multiplication
 - `+` | `-` : addition, soustraction
- Exemple : `42.0 / 10.0 \Leftrightarrow 4.2`
- La multiplication, la division sont prioritaires sur l'addition et la soustraction
- Dans le cas où les types ne sont pas les mêmes, le compilateur va utiliser le type de plus grande capacité
- Les opérations se passent minimalement en int
- En cas de dépassement de capacité, le comportement va dépendre du compilateur et de l'architecture cible. Pour les entiers, on ignore souvent les bits supplémentaires. Pour les réels, on peut trouver la valeur "infinity" et "NaN"

Manipulation bit à bit

Opérateur	Symbole C++	Utilisation	Opération réel
décalage à gauche	<<	$x \ll y$	Tous les bits de x sont décalés de y positions vers la gauche
décalage à droite	>>	$x \gg y$	Tous les bits de x sont décalés de y positions vers la droite
NON bit à bit	~	$\sim x$	Tous les bits de x sont inversés
ET bit à bit (AND)	&	$x \& y$	Chaque bit de x AND chaque bit de y
OU bit à bit (OR)		$x y$	Chaque bit de x OR chaque bit de y

Priorité des opérateurs

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
2	a++ a-- type() type{} a() a[] . ->	Suffix/postfix increment and decrement Functional cast Function call Subscript Member access	
3	++a --a +a -a ! ~ (type) *a &a sizeof new new[] delete delete[]	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT C-style cast Indirection (dereference) Address-of Size-of ^[note 1] Dynamic memory allocation Dynamic memory deallocation	Right-to-left
4	.* ->*	Pointer-to-member	Left-to-right
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	<=>	Three-way comparison operator (since C++20)	
9	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
10	== !=	For relational operators = and ≠ respectively	
11	&	Bitwise AND	
12	^	Bitwise XOR (exclusive or)	
13		Bitwise OR (inclusive or)	
14	&&	Logical AND	
15		Logical OR	
16	a?b:c throw = += -= *= /= %= <<= >>= &= ^= =	Ternary conditional ^[note 2] throw operator Direct assignment (provided by default for C++ classes) Compound assignment by sum and difference Compound assignment by product, quotient, and remainder Compound assignment by bitwise left shift and right shift Compound assignment by bitwise AND, XOR, and OR	Right-to-left
17	,	Comma	Left-to-right

Tableaux

- Syntaxe : `<type> <nomVariable>[<capacité>]`
- La **capacité** doit être **connue** à la **compilation**
- Exemple :
 - `int mesDonnees[5] = { 13, 42, 23, 17, 5};`
 - `int mesDonnees[] = { 13, 42, 23, 17, 5};`
- Attention : `sizeof(mesDonnees) -> 20`

Entrées/sorties

- En C++, on manipule des variables globales de l'espace de nommage `std` : `cout/cerr` (flux de sortie) et `cin` (flux d'entrée)
- Ces variables déclarent les opérateurs « `<<` » et « `>>` »
- Exemples :
 - `std::cout << "Bonjour à toutes et à tous !" << std::endl;`
 - `std::string s;`
`std::cin >> s;`
- Remarque : "cin" surcharge son opérateur « `>>` » pour les types de base. Il ne lit que ce qu'il a besoin dans le flux. Si le flux contient déjà ce dont il a besoin, il n'attendra pas d'entrée clavier

Entrées/sorties

- Pour les entrées, après une lecture, vous pouvez valider les erreurs avec la méthode « good »

```
if (std::cin.good()) { ... }
```

- Pour réinitialiser le statut d'erreur, vous pouvez utiliser la méthode « clear »

Fonctions

- Syntaxe :

```
<type retour> <nom de la fonction>([<type paramètre> <nomParamètre>]*) {  
    // [...]  
    [return <expression du type de retour>;]  
}
```

- Exemple :

```
bool estPair(int p_valeur) {  
    return (p_valeur & 1) == 0;  
}
```

```
<typeRetour> <nomFonction>([<typeParam1> <nomParam1>  
[ = <constante1>]][, <typeParamN> <nomParamN> [ =  
<constanteN>]]*) { <bloc code>; [return  
<expression>; ] }
```

Fonctions – Passage de paramètres

- Les paramètres peuvent être passés par :
 - **Valeur** : les valeurs sont copiées, on manipule donc une copie de ce qui est passé en paramètre
 - **Par référence (&)** : le paramètre référence la même case mémoire que celle utilisée par l'appelant. Si on modifie la valeur dans la fonction, elle est modifiée dans l'appelant
- Une **référence agit comme un synonyme**
- Les tableaux passés en paramètres perdent leur dimension (sizeof) et sont alors assimilable à des pointeurs (vu plus loin). Ils sont passés par copie de pointeur, donc le tableau n'est pas copié, c'est simplement son pointeur qui l'est.


Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	


Fonctions – Exemples – Passage par valeur



```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```

Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	
p_v1(@0FFFAB08)	p_v2 (@0FFFAB0C)	
42	13	
temp (@0FFFAB10)		
42		


Fonctions – Exemples – Passage par valeur



```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```

Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	
p_v1(@0FFFAB08)	p_v2 (@0FFFAB0C)	
13	13	
temp (@0FFFAB10)		
42		


Fonctions – Exemples – Passage par valeur



```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```

Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	
p_v1(@0FFFAB08)	p_v2 (@0FFFAB0C)	
13	42	
temp (@0FFFAB10)		
42		

Fonctions – Exemples – Passage par valeur




```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```

Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	
p_v1 (@0FFFAB08)	p_v2 (@0FFFAB0C)	
13	42	
temp (@0FFFAB10)		
42		

Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



Pile		Tas
v1 (@0FFFAB00)	v2 (@0FFFAB04)	
42	13	


v1 et v2 n'ont pas été modifié !

Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
→ echangerReferences(v1, v2);  
// ...
```

Pile		Tas
v1 (@0FFFAB00) 42	v2 (@0FFFAB04) 13	


Fonctions – Exemples – Passage par référence



```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```

Pile	Tas
<div>v1, p_v1(@0FFFFAB00)</div> <div>42</div> <div>v2, p_v2(@0FFFFAB04)</div> <div>13</div> <div>temp(@0FFFFAB08)</div> <div>42</div>	


Fonctions – Exemples – Passage par référence



```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```

Pile	Tas
<div>v1, p_v1 (@0FFFAB00)</div> <div>13</div> <div>v2, p_v2 (@0FFFAB04)</div> <div>13</div> <div>temp (@0FFFAB08)</div> <div>42</div>	


Fonctions – Exemples – Passage par référence



```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```

Pile	Tas
<div>v1, p_v1 (@0FFFAB00)</div> <div>13</div> <div>v2, p_v2 (@0FFFAB04)</div> <div>42</div> <div>temp (@0FFFAB08)</div> <div>42</div>	

Fonctions – Exemples – Passage par référence



```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```

Pile	Tas
<div>v1, p_v1 (@0FFFAB00)</div> <div>13</div> <div>v2, p_v2 (@0FFFAB04)</div> <div>42</div> <div>temp (@0FFFAB08)</div> <div>42</div>	

Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



Pile		Tas
v1 (@0FFFAB00) 13	v2 (@0FFFAB04) 42	

Fonctions anonymes

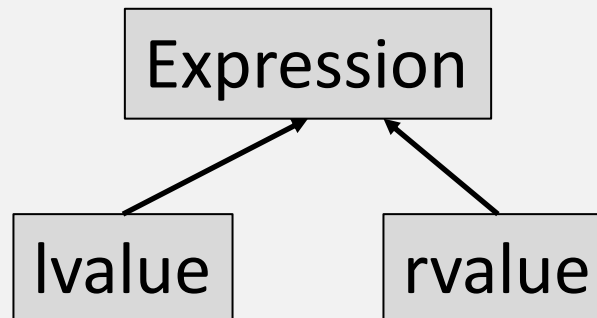
- Fonction sans nom
- Syntaxe : [<variableACapturer>*] (<parametre>*) { <bloc code> }

```
#include <algorithm>
#include <cmath>

void abssort(float* x, unsigned n) {
    std::sort(x, x + n,
        // Lambda expression begins
        [](float a, float b) {
            return (std::abs(a) < std::abs(b));
        } // end of lambda expression
    );
}
```

lvalue / rvalue (Version simplifiée)

- lvalue (left value) : une expression qui a **une case mémoire adressable** (ex. variable, tableau + index, etc.)
- rvalue (right value) : une expression simple ou contenant aussi des lvalue qui n'a pas (conceptuellement) de case mémoire
- Idée : permettre au compilateur d'avoir plus de liberté pour traiter ce genre d'expression avec le rvalue



lvalue / rvalue (origines)

- Cas de l'affectation :
 - Syntaxe : `<lvalue> = <expression>;`
- Ex. :
 - `n = 10;`
 - `a = n;` // Conversion de lvalue vers rvalue pour `n`
 - ~~`42 = a;`~~ // 42 n'est pas une lvalue on ne peut pas lui affecter la valeur de `a`
 - ~~`m + 1 = n;`~~ // `m` est une lvalue, `1` une rvalue, `m + 1` est une rvalue : on ne peut pas lui assigner une valeur
- Autres exemples
 - `i++`, `++i` : `i` doit être une lvalue, le résultat est une rvalue
 - `&a` : rvalue (Opérateur d'adresse vu plus tard dans le cours, ici `a` doit être une lvalue)
 - `*ptr` : lvalue (Opérateur de déréférencement vu plus tard dans le cours)

Fichiers d'en-tête

- Fichiers de déclarations ou d'entêtes (.h)
 - Ces fichiers servent à déclarer des fonctions ou des classes
 - Il ne contient donc généralement pas de définition (c.à.d.. de corps de fonction / méthode)
- La première ligne doit contenir la directive de précompilation « #pragma once » (Version simple)
- Des fichiers d'entête peuvent être inclus dans un autre fichier d'entête ou un fichier de définition avec la directive de pré-compilation « #include <nomFichier> »
- <nomFichier> :
 - Si le nom de fichier est encadré par « < » et « > » : le fichier est cherché dans les répertoires configurés au moment de la compilation
 - Si le nom de fichier est encadré par « " » : le fichier est cherché à partir du fichier courant (chemin relatif)

Déclaration / Définition – Exemple

util.h

```
#pragma once

int contrainteValeur(int p_valeur, int p_min, int p_max);
```

util.cpp

```
#include "util.h"

int contrainteValeur(int p_valeur, int p_min, int p_max) {
    int resultat = p_valeur;

    if (p_valeur < p_min) {
        p_valeur = p_min;
    }
    if (p_valeur > p_max) {
        p_valeur = p_max;
    }

    return p_valeur;
}
```

Déclaration / Définition – Exemple

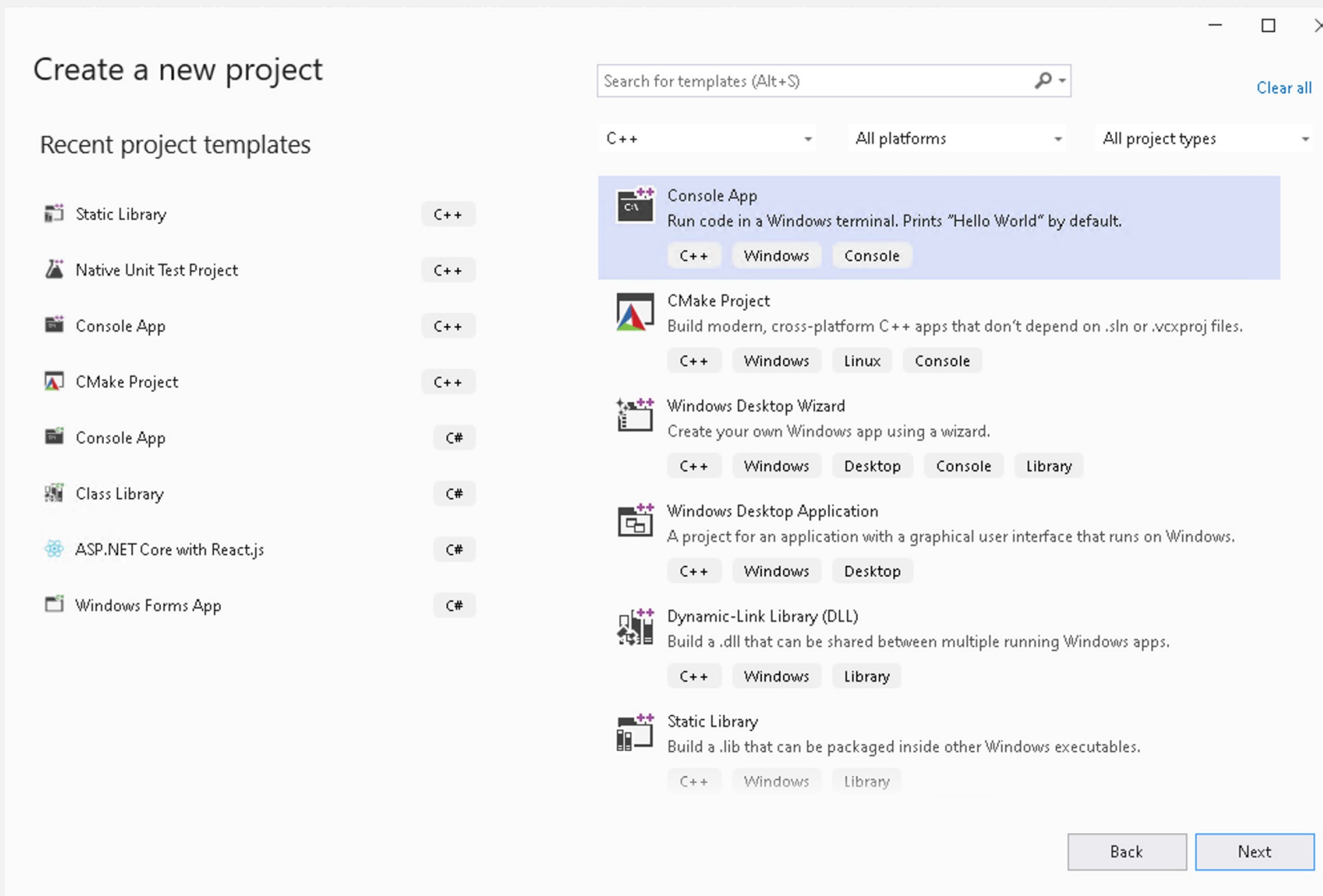
main.cpp

```
#include <iostream>
#include "util.h"

int main() {
    int min = 0;
    int max = 100;

    std::cout << contrainteValeur(110, min, max) << std::endl;
    std::cout << contrainteValeur(-3, min, max) << std::endl;
}
```

Visual Studio – HelloWorld



Visual Studio – HelloWorld

□ ×

Configure your new project

Console App C++ Windows Console

Project name

Location

 ...

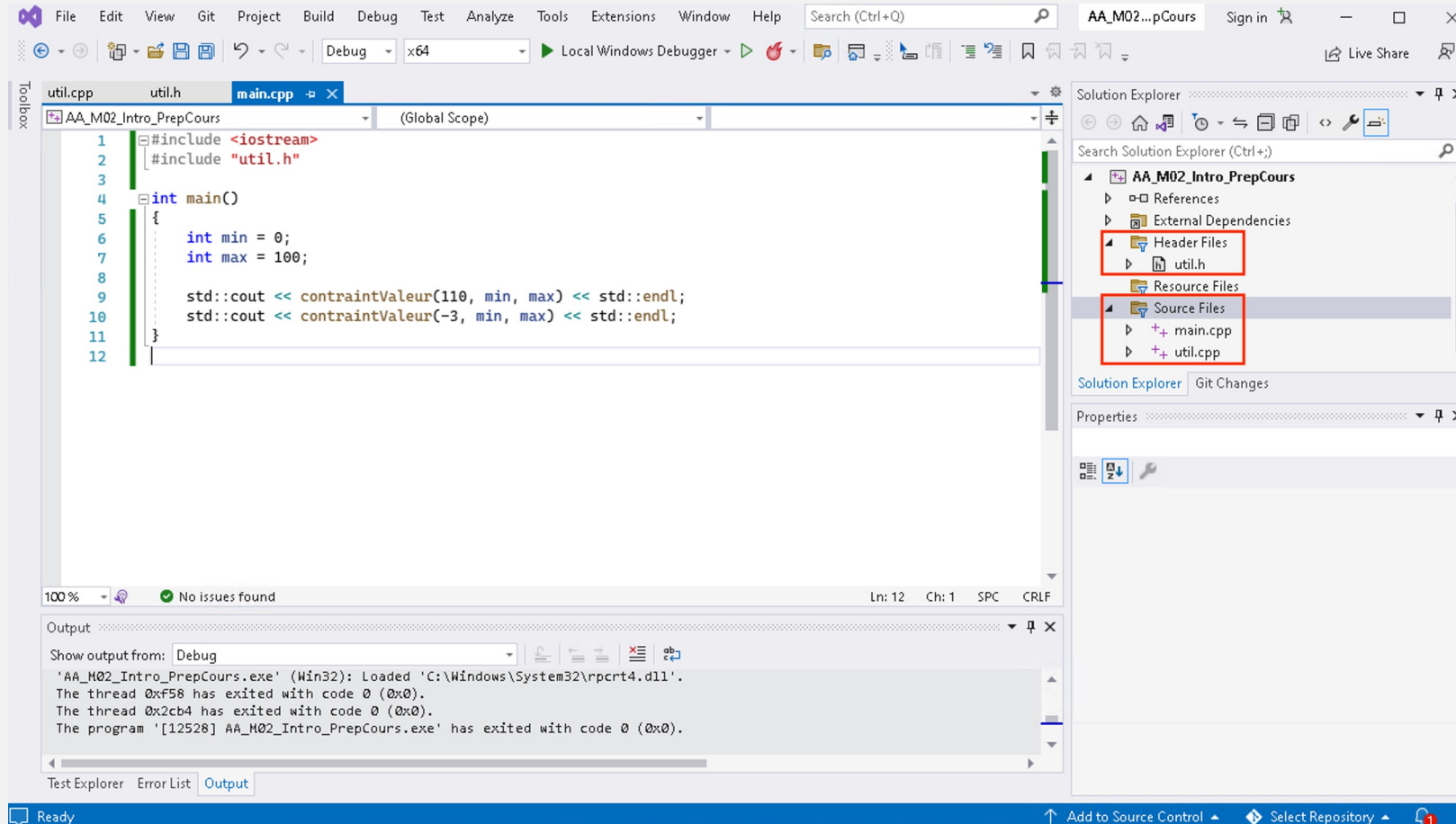
Solution

Solution name ⓘ

☐ Place solution and project in the same directory

Back Create

Visual Studio – HelloWorld



Visual Studio – HelloWorld

Show output from: Build

Build started...

1>----- Build started: Project: AA_M02_Intro_PrepCours, Configuration: Debug x64 -----

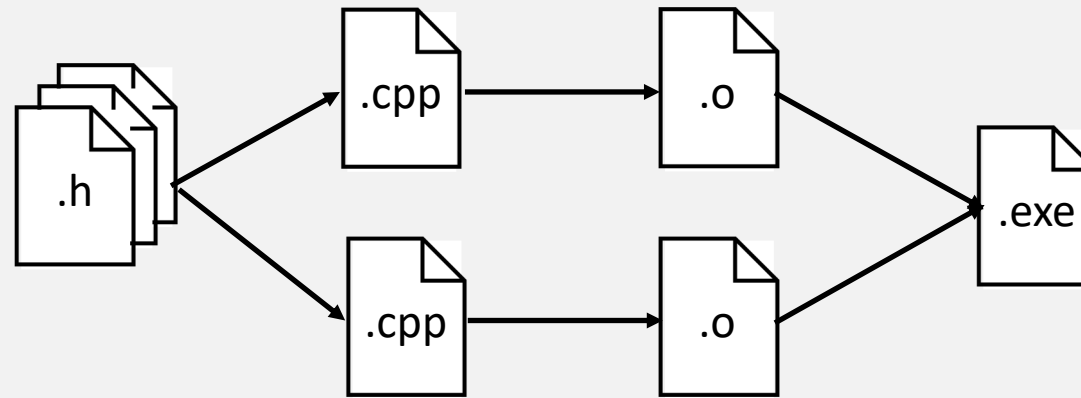
1>main.cpp

1>util.cpp

1>Generating Code...

1>AA_M02_Intro_PrepCours.vcxproj -> C:\Users\pfleon\source\repos\AA_M02_Intro_PrepCours\x64\Debug\AA_M02_Intro_PrepCours.exe

===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====



<< AA_M02_Intro_PrepCours > x64 > Debug >

Name

- AA_M02_I.8cec5213.tlog
- AA_M02_Intro_PrepCours.Build.CppClean.log
- AA_M02_Intro_PrepCours.exe.recipe
- AA_M02_Intro_PrepCours.ilc
- AA_M02_Intro_PrepCours.log
- AA_M02_Intro_PrepCours.vcxproj.FileListAbsolute.
- main.obj
- util.obj

<< source > repos > AA_M02_Intro_PrepCours > x64 > Debug

Name

Date modified

Type

AA_M02_Intro_PrepCours.exe

2023-05-21 22:53

Application

Références

- <https://learn.microsoft.com/en-us/cpp/cpp/data-type-ranges> :
domaine de valeurs des types de base
- https://en.cppreference.com/w/cpp/language/operator_precedence :
priorité des opérateurs
- <https://learn.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp> : lambda
- https://en.cppreference.com/w/cpp/language/value_category :
lvalue/rvalue

Annexes

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

Tri à bulles

- Le nom provient du principe de ce tri : on parcourt 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
---	----	-----	-----	----	----	-----	----	---	----	---



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > 12$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

12 > -99 => **vrai**

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



indiceCourant

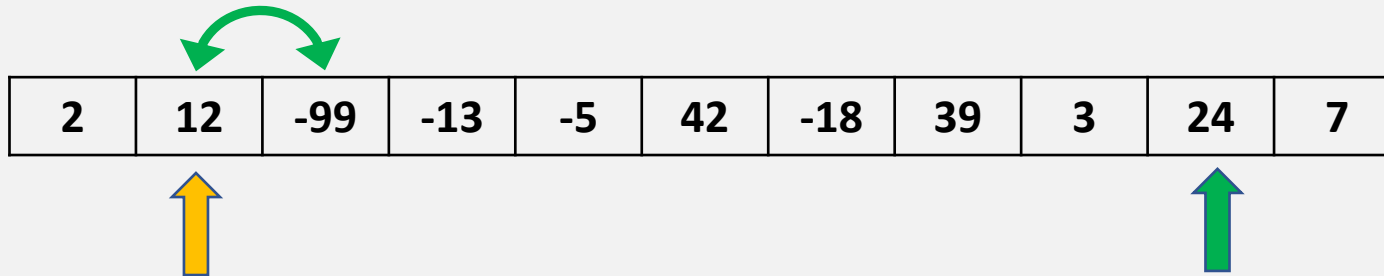


indiceMax


Tri à bulles

permutationAuDernierTour => **faux**

12 > -99 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -13$ => **vrai**

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



indiceCourant

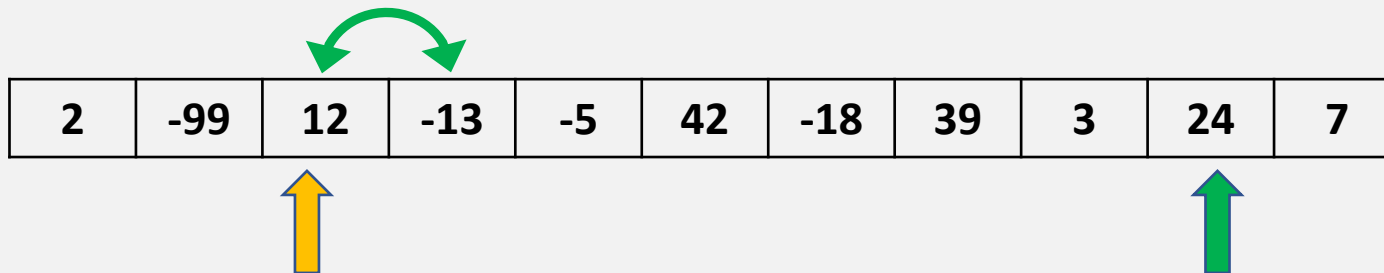



indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

12 > -13 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

12 > -5 => **vrai**

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



indiceCourant

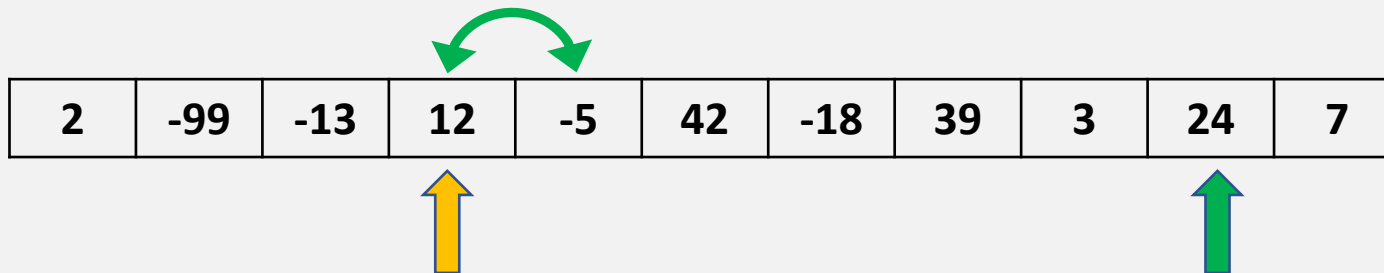


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

12 > -5 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

12 > 42 => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

42 > -18 => **vrai**

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



indiceCourant

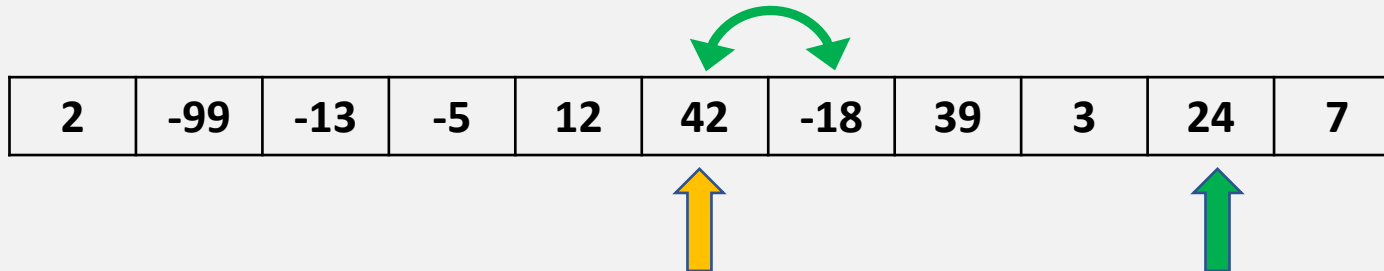



indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

$42 > -18$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

42 > 39 => **vrai**

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



indiceCourant

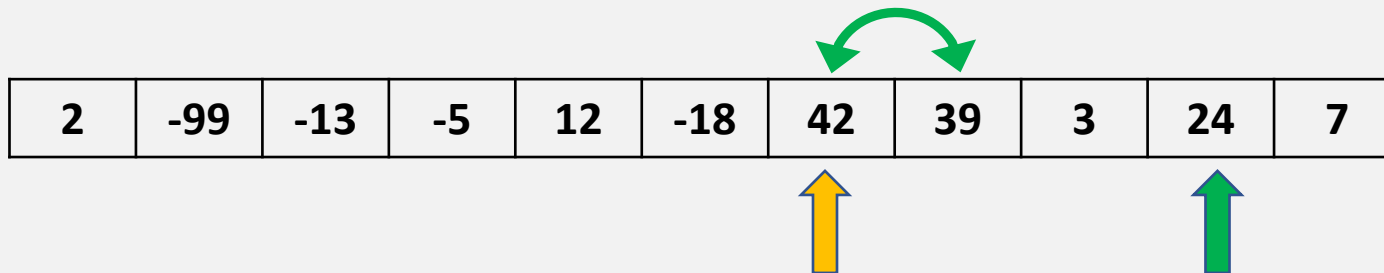



indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

42 > 39 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

42 > 3 => **vrai**

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



indiceCourant

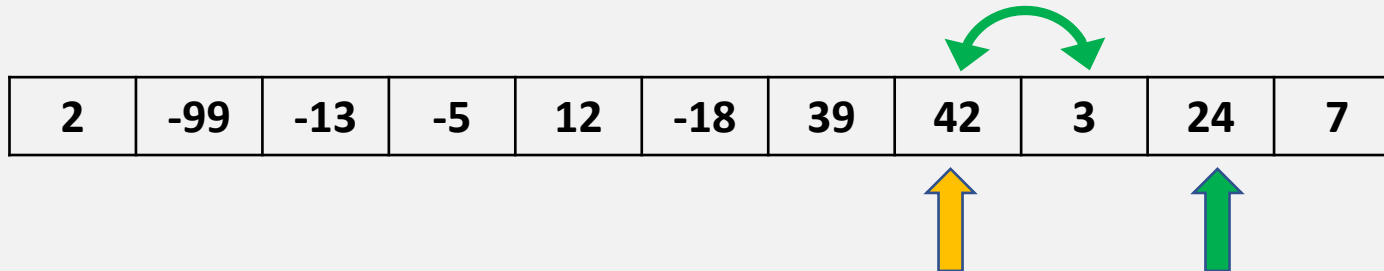


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

42 > 3 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

42 > 24 => **vrai**

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



indiceCourant

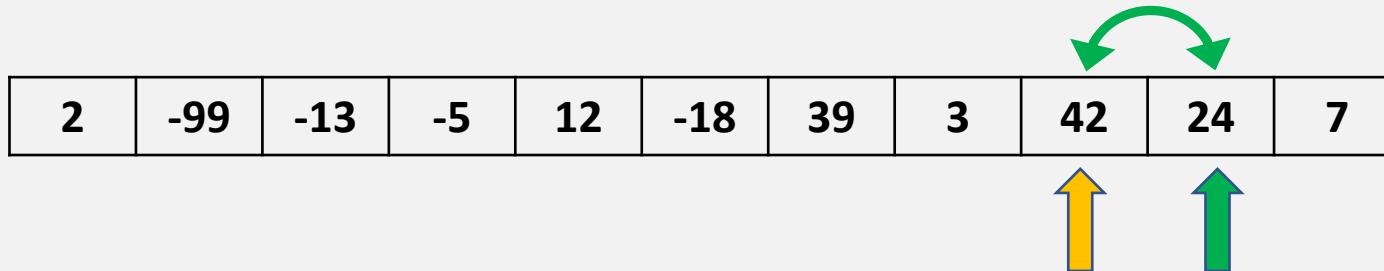



indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

42 > 24 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles


permutationAuDernierTour => **vrai**

42 > 7 => **vrai**

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



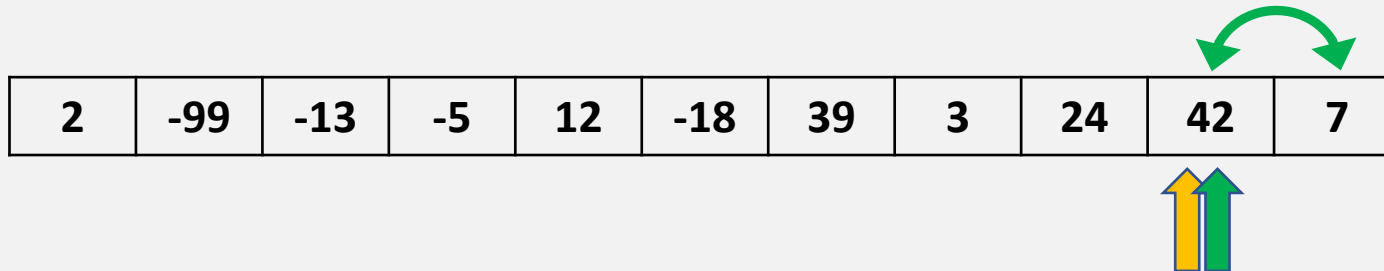
 indiceCourant


 indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

42 > 7 => **vrai**



 indiceCourant

 indiceMax


Tri à bulles


permutationAuDernierTour => **vrai**

42 > 7 => **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**

$2 > -99$ => **vrai**

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



indiceCourant

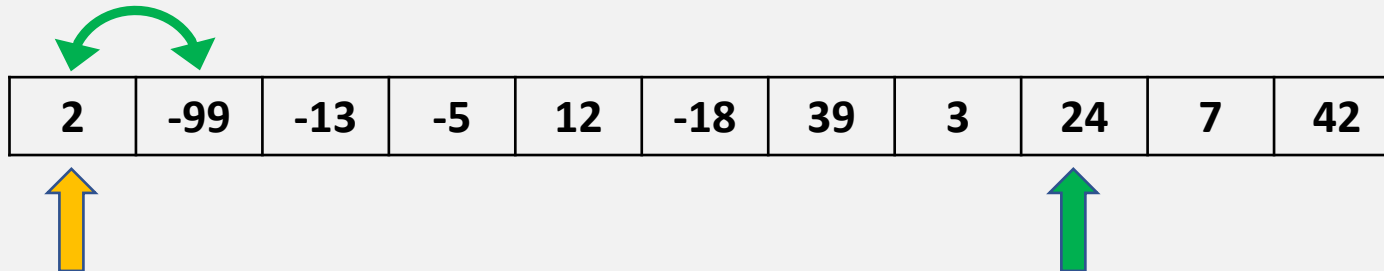


indiceMax


Tri à bulles

permutationAuDernierTour => **faux**

$2 > -99$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

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

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -13$ => **vrai**

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



indiceCourant

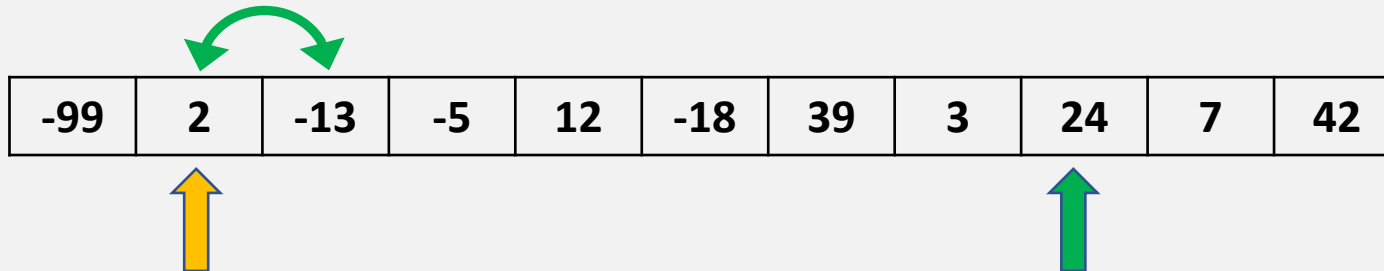



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -13$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -5$ => **vrai**

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



indiceCourant

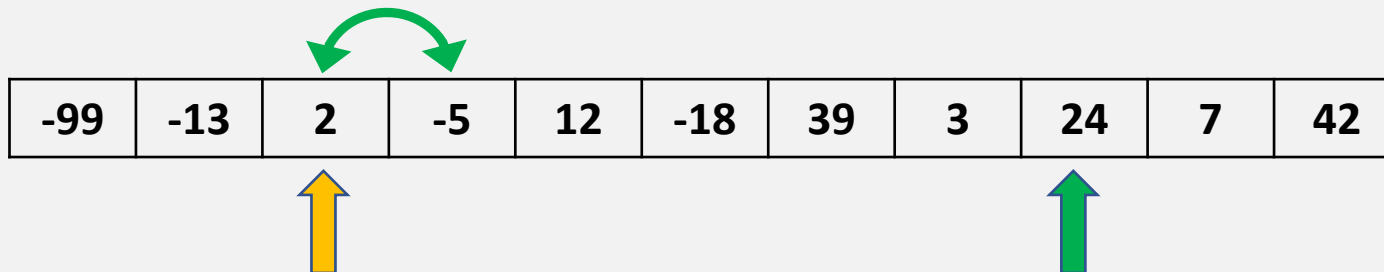


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

$2 > -5 \Rightarrow$ **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 12$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -18$ => **vrai**

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



indiceCourant

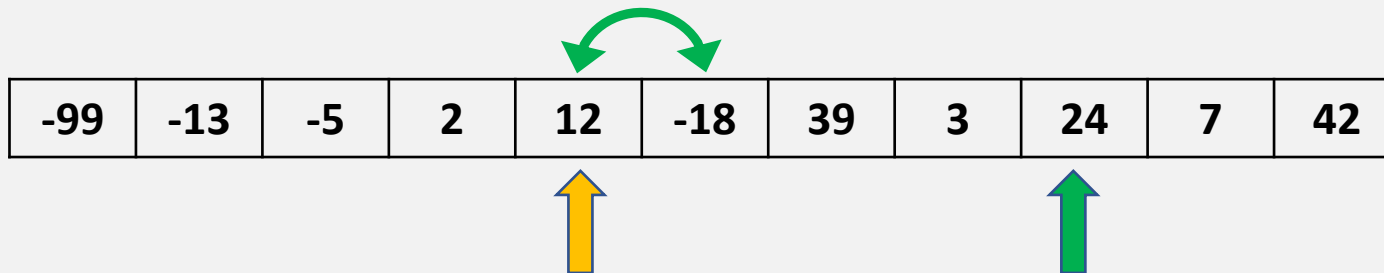


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

$12 > -18$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

12 > 39 => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

39 > 3 => **vrai**

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



indiceCourant

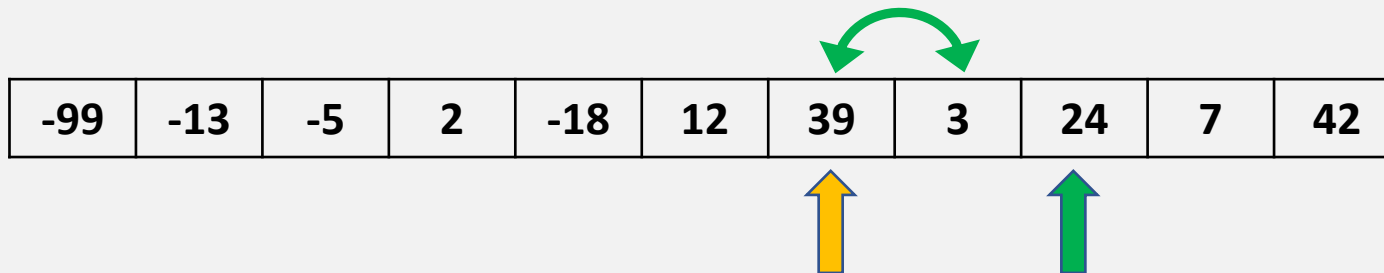



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$39 > 3$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

39 > 24 => **vrai**

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



indiceCourant

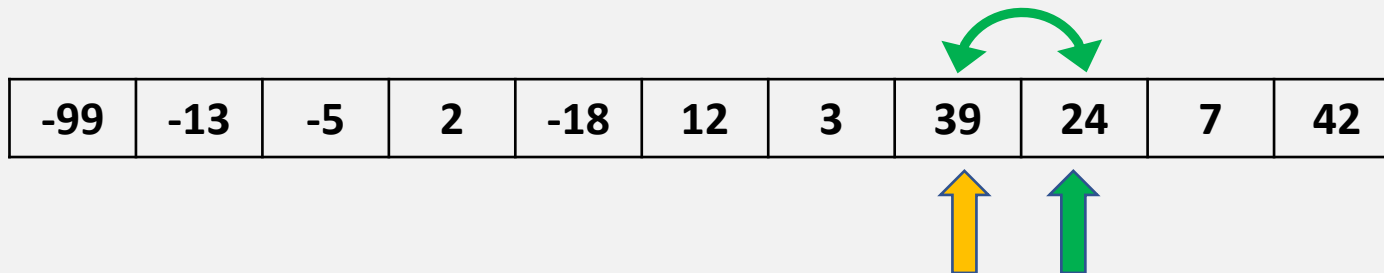


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

39 > 24 => **vrai**



 indiceCourant

 indiceMax


Tri à bulles


permutationAuDernierTour => **vrai**

39 > 7 => **vrai**

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



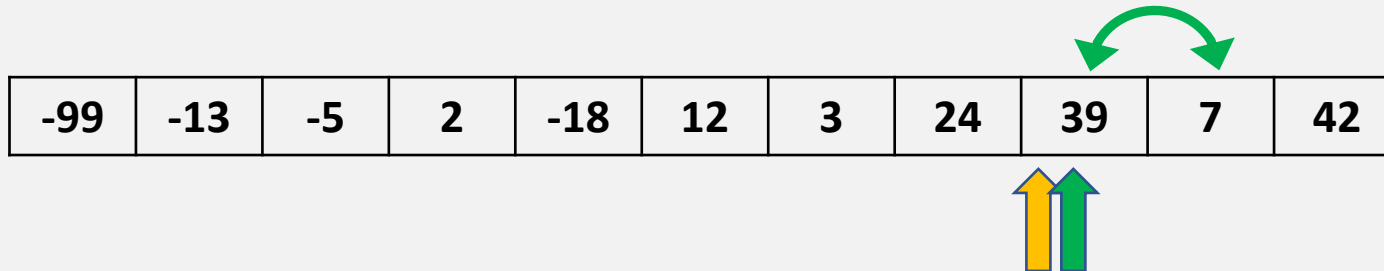
 indiceCourant


 indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

39 > 7 => **vrai**



 indiceCourant

 indiceMax


Tri à bulles


permutationAuDernierTour => **vrai**

39 > 7 => **vrai**

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



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -13$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -5$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-5 > 2$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$2 > -18$ => **vrai**

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



indiceCourant

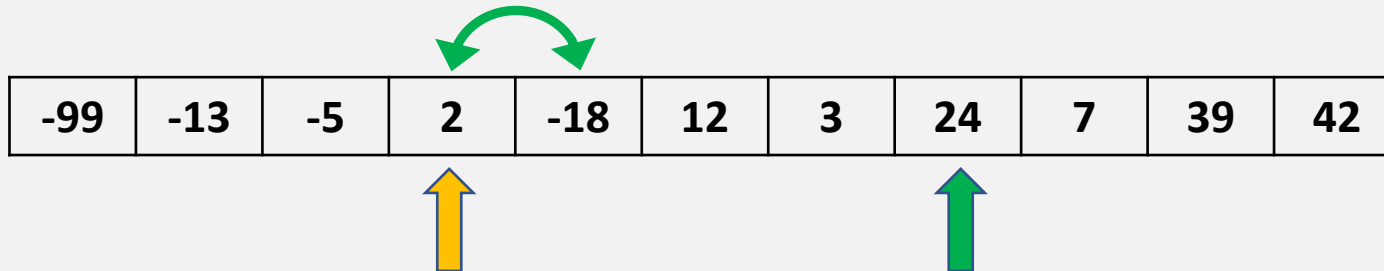


indiceMax


Tri à bulles

permutationAuDernierTour => **faux**

$2 > -18$ => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 12$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

12 > 3 => **vrai**

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



indiceCourant

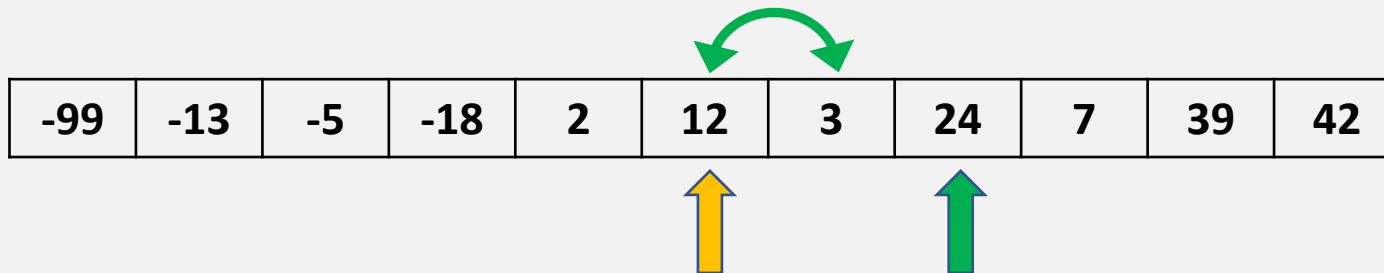


indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

12 > 3 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

12 > 24 => **faux**

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



indiceCourant

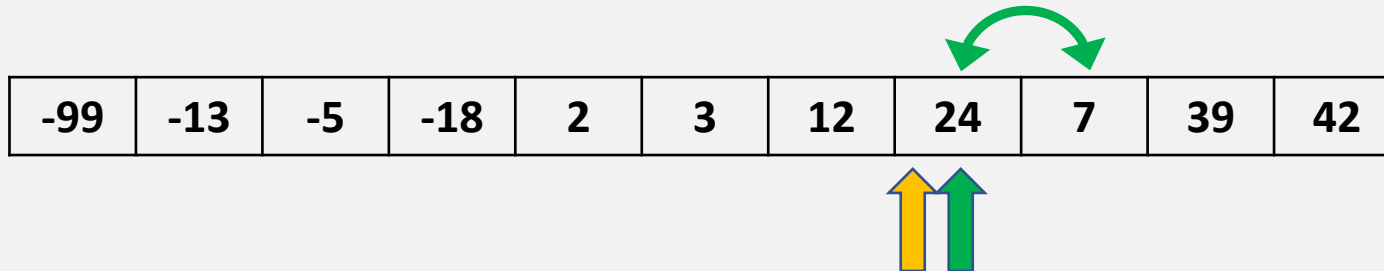



indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

24 > 7 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles


permutationAuDernierTour => **vrai**

24 > 7 => **vrai**

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



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -13$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-13 > -5$ => **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

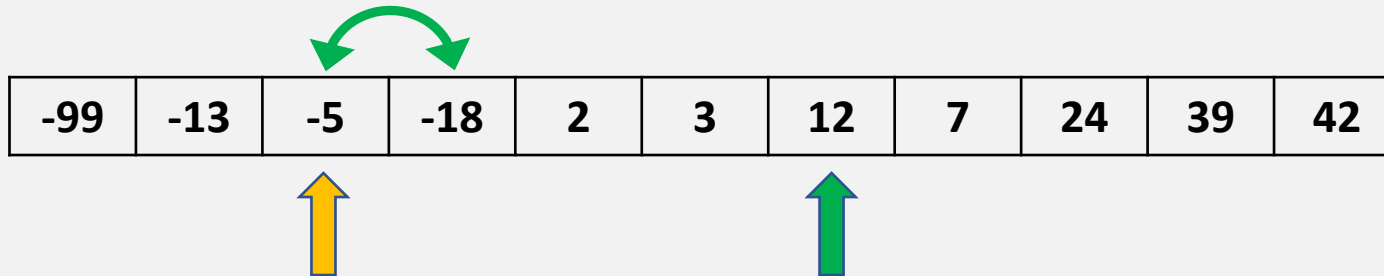



indiceMax


Tri à bulles

permutationAuDernierTour => **faux**

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



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$-5 > 2$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$2 > 3$ => **faux**

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



indiceCourant



indiceMax


Tri à bulles


permutationAuDernierTour => **vrai**

$3 > 12$ => **faux**

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



 indiceCourant

 indiceMax

Tri à bulles


permutationAuDernierTour => **vrai**

12 > 7 => **vrai**

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



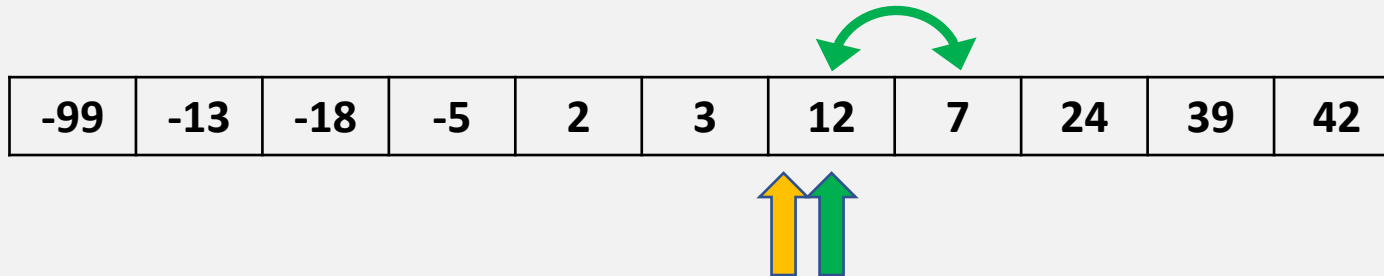
 indiceCourant


 indiceMax


Tri à bulles

permutationAuDernierTour => **vrai**

$12 > 7$ => **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 => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

-13 > -18 => **vrai**

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



indiceCourant

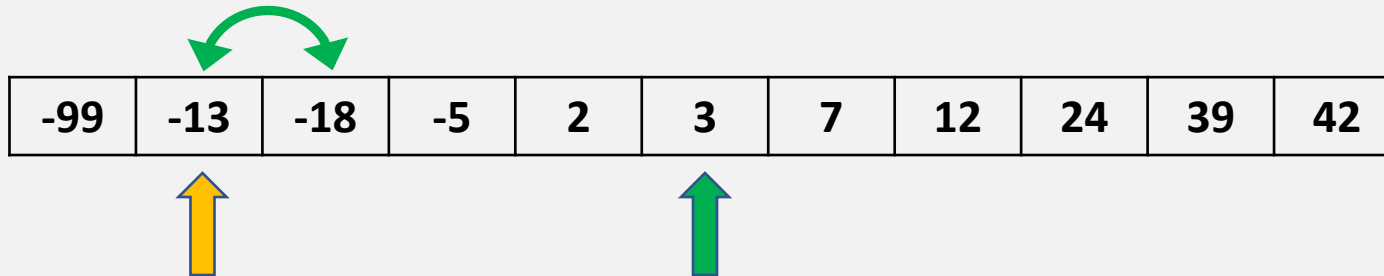


indiceMax


Tri à bulles

permutationAuDernierTour => **faux**

-13 > -18 => **vrai**



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

$-13 > -5$ => **faux**

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



indiceCourant



indiceMax

Tri à bulles

permutationAuDernierTour => **vrai**

-5 > 2 => **faux**

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



indiceCourant



indiceMax

Tri à bulles


permutationAuDernierTour => **vrai**

$2 > 3$ => **faux**

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



 indiceCourant

 indiceMax


Tri à bulles


permutationAuDernierTour => **vrai**

3 > 7 => **faux**

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



 indiceCourant

 indiceMax

Tri à bulles

permutationAuDernierTour => **faux**

$-99 > -18$ => **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**

$-13 > -5$ => **faux**

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



indiceCourant



indiceMax


Tri à bulles


permutationAuDernierTour => **faux**

-5 > 2 => **faux**

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



 indiceCourant

 indiceMax


Tri à bulles


permutationAuDernierTour => **faux**

$2 > 3$ => **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;  
    boolean 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

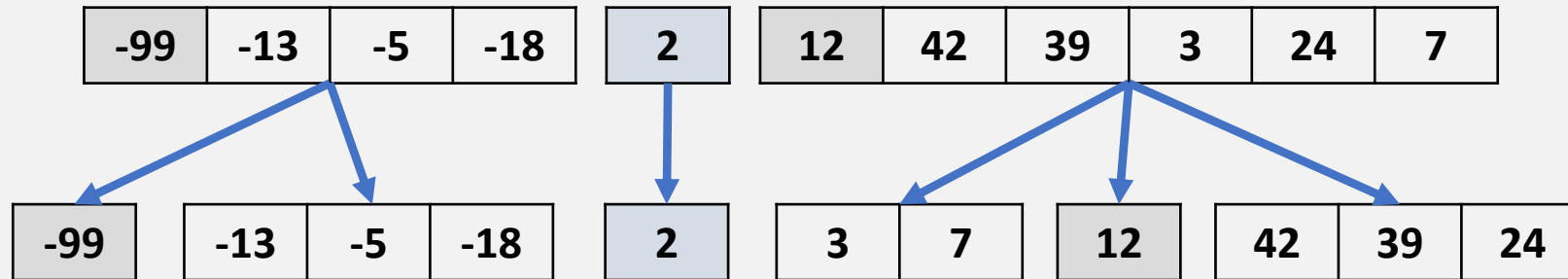
- Comment diviser un tableau ?
 - Idée : déterminer une valeur dite pivot et créer deux tableaux tels que :
 - $\forall \text{élément} \in \text{partition1}, \text{élément} \leq \text{valeurPivot}$
 - $\forall \text{élément} \in \text{partition2}, \text{élément} > \text{valeurPivot}$
 - Exemple :

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

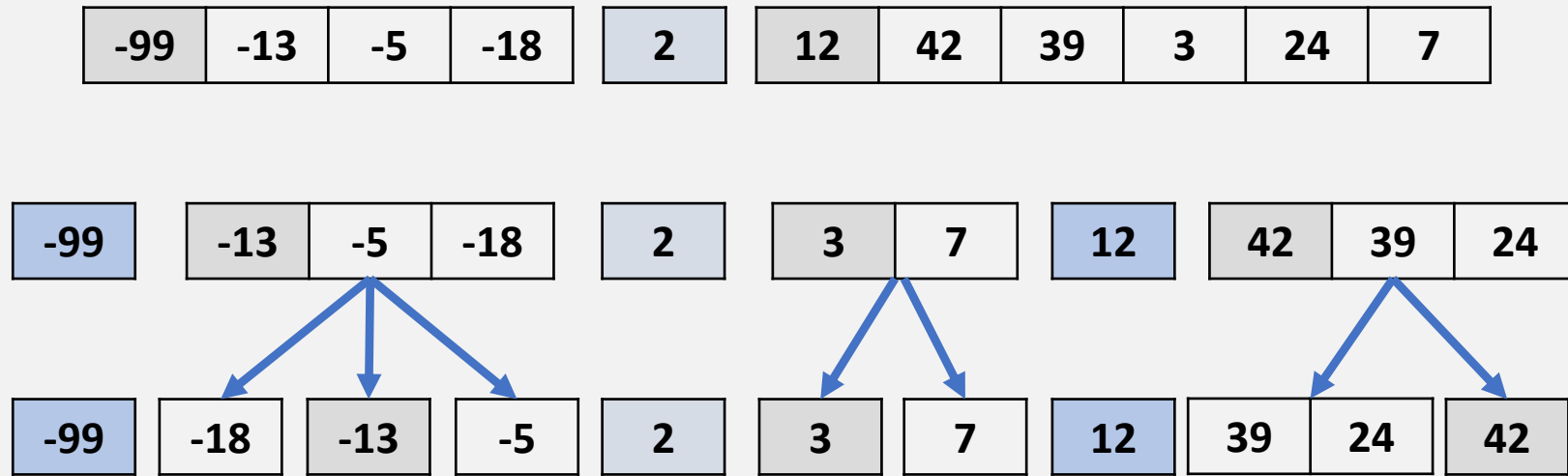
⇒

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

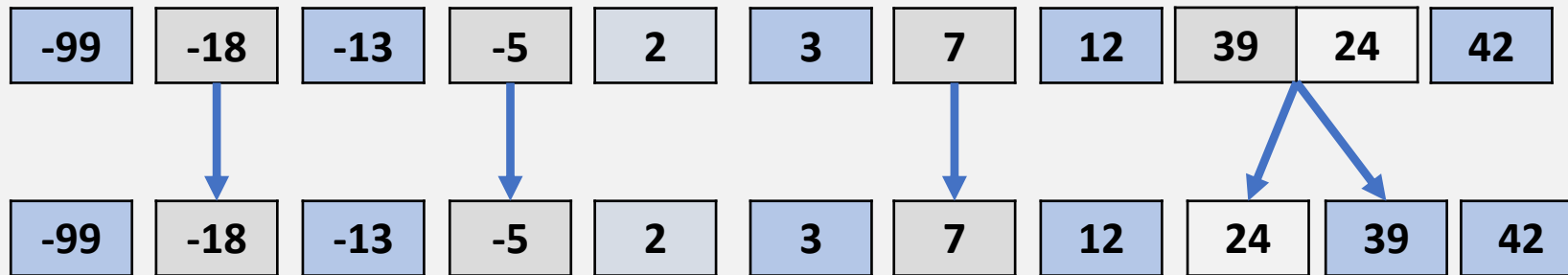
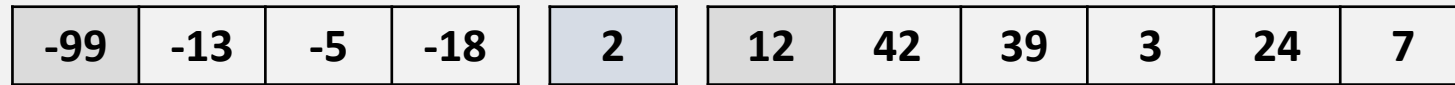
Diviser pour régner – Tri rapide



Diviser pour régner – Tri rapide



Diviser pour régner – Tri rapide



Diviser pour régner – Tri rapide

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

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

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

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

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

Diviser pour régner – Tri rapide

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

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

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

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

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

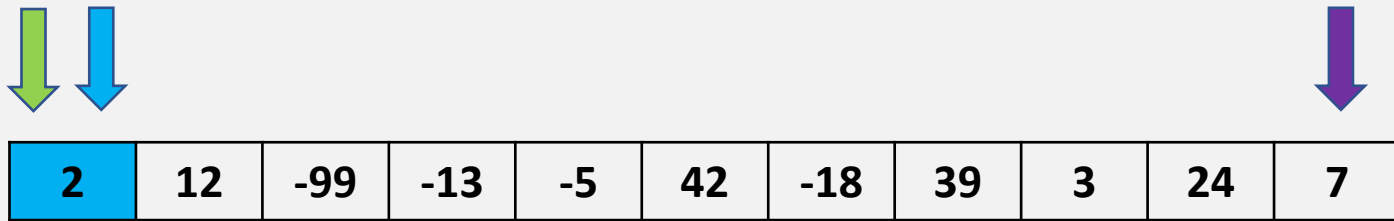
Pseudocode pour les tableaux

```
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




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

 indicePivot

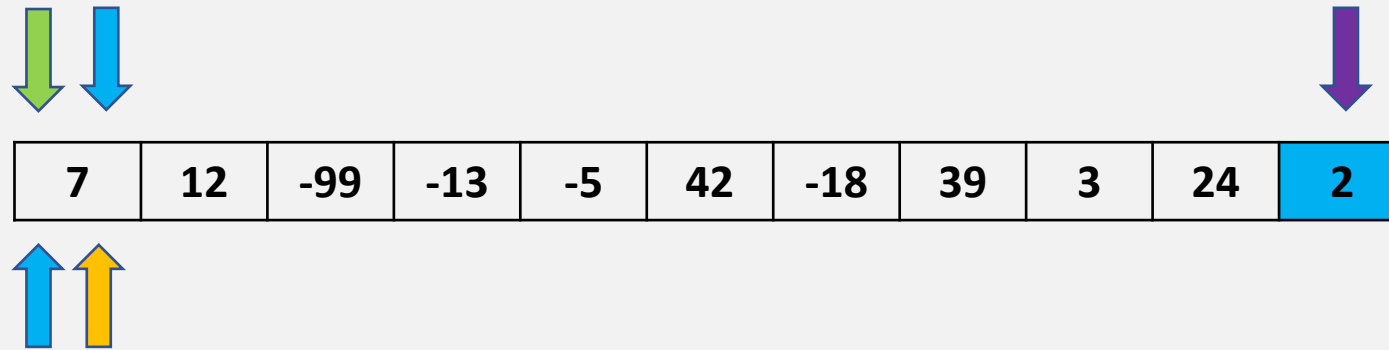
 indicePremier






 indiceDernier

 indiceValeurARanger

Partitionnement optimisé

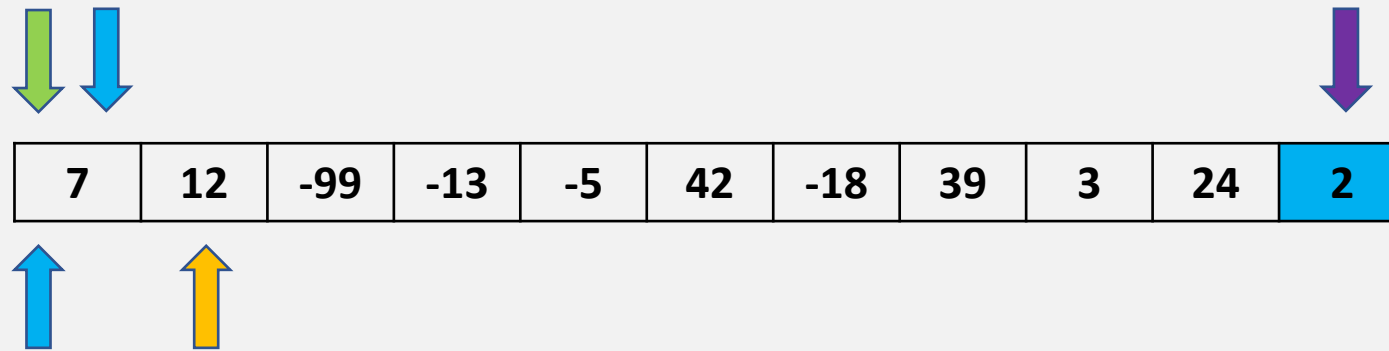
$7 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

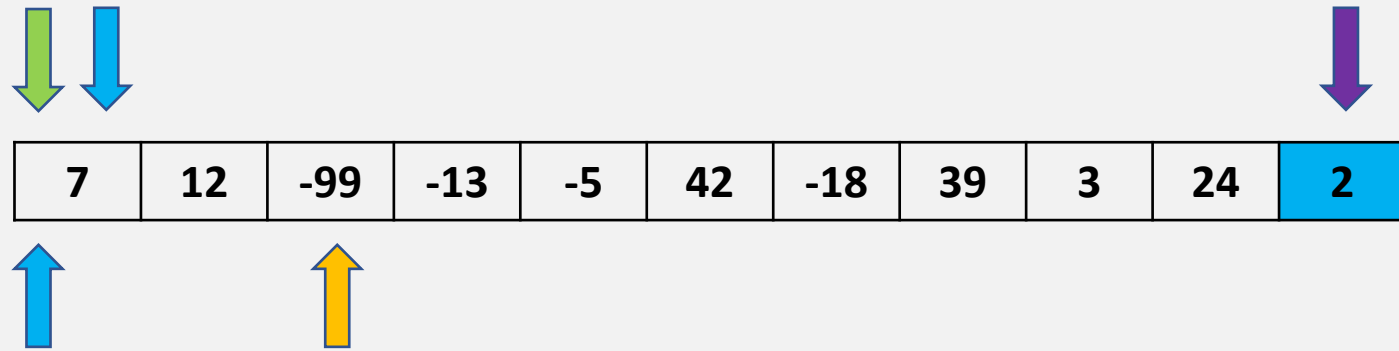
$12 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

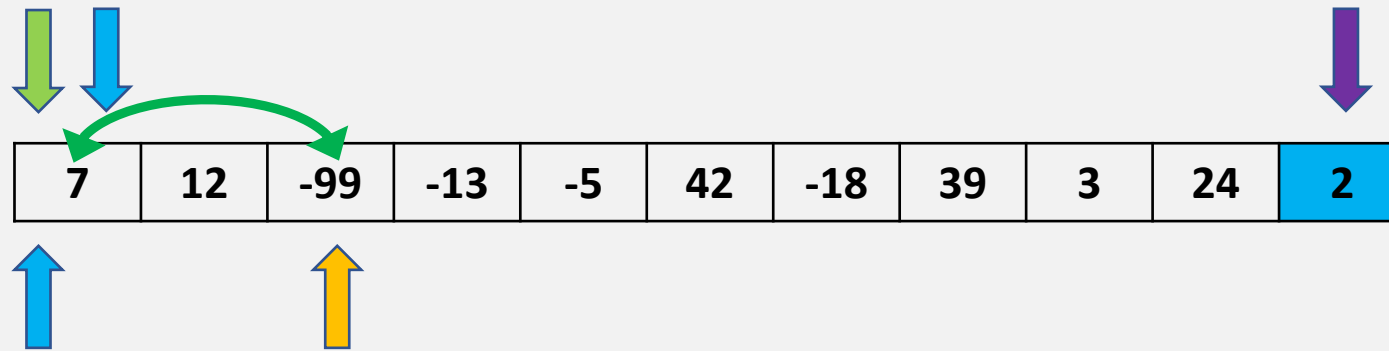
$-99 \leq 2 \Rightarrow$ vrai



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

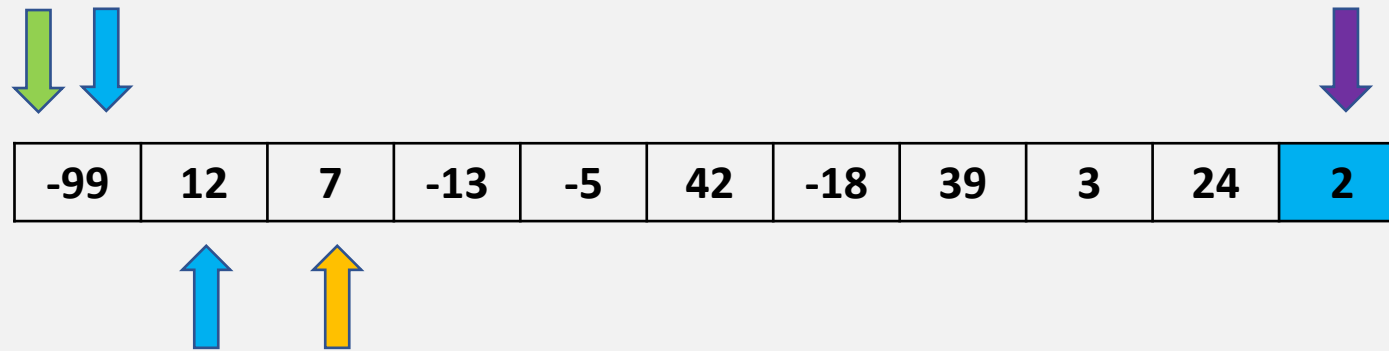
Échanger les valeurs








↓ indicePivot ↓ indicePremier ↓ indiceDernier ↑ indiceValeurARanger ↑ futurIndicePivot

Partitionnement optimisé

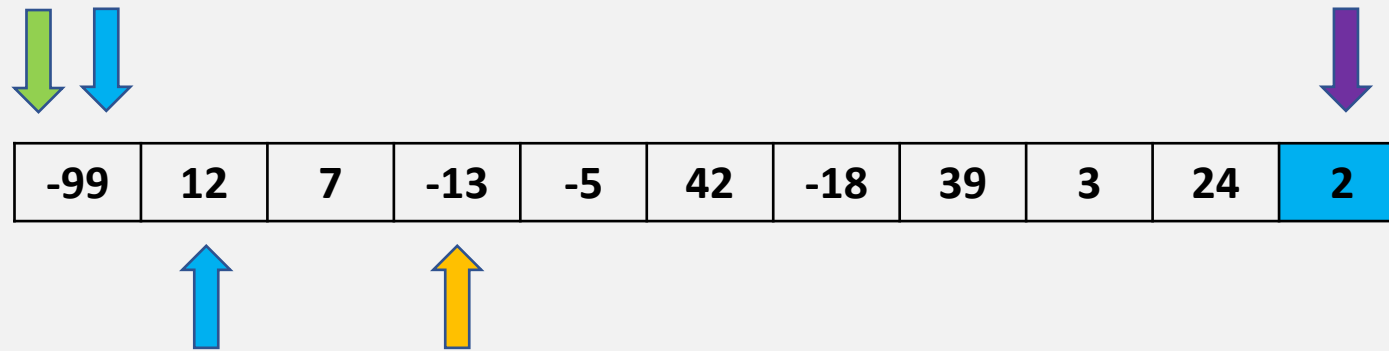
Avancer futurIndicePivot








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

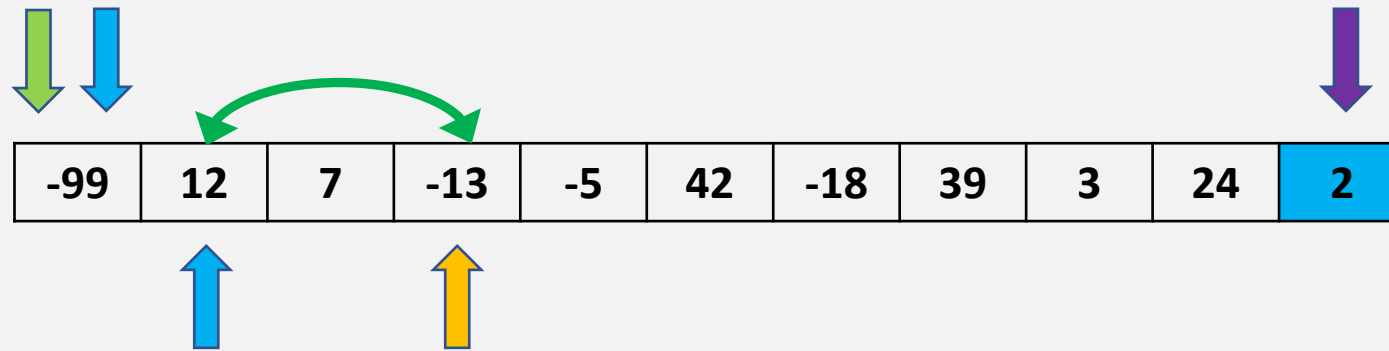
$-13 \leq 2 \Rightarrow$ vrai



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

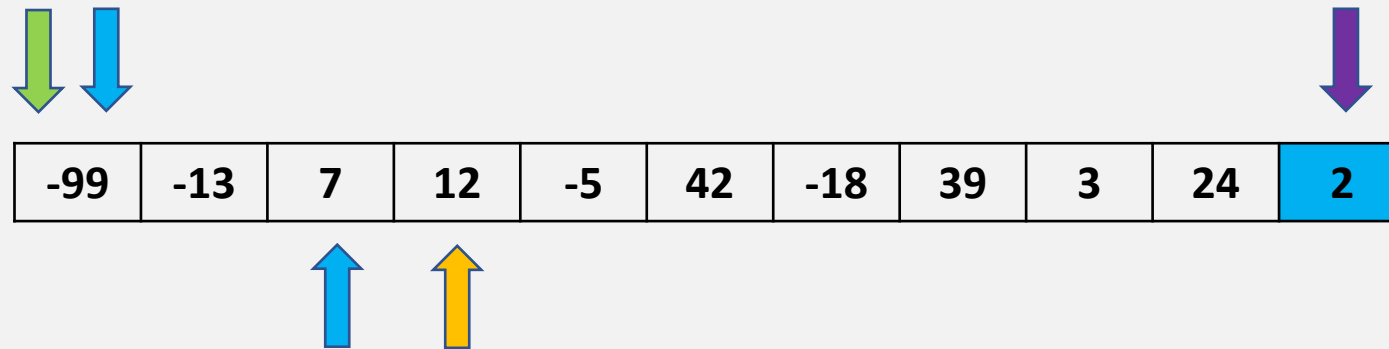
Échanger les valeurs








↓ indicePivot ↓ indicePremier ↓ indiceDernier ↑ indiceValeurARanger ↑ futurIndicePivot

Partitionnement optimisé

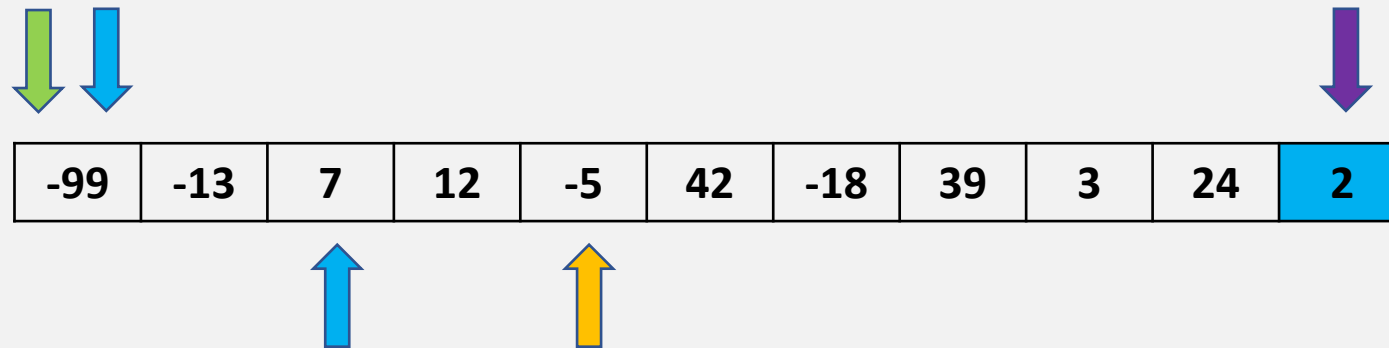
Avancer futurIndicePivot








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

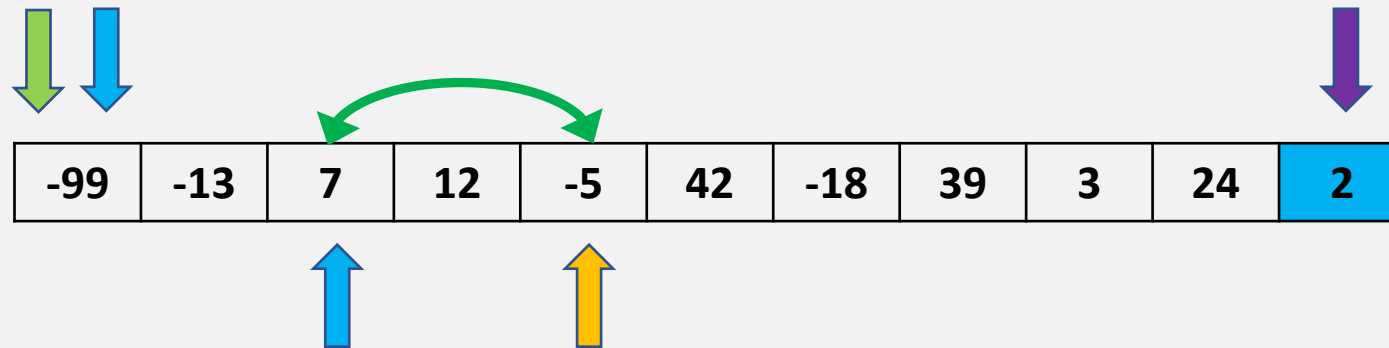
$-5 \leq 2 \Rightarrow$ vrai



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

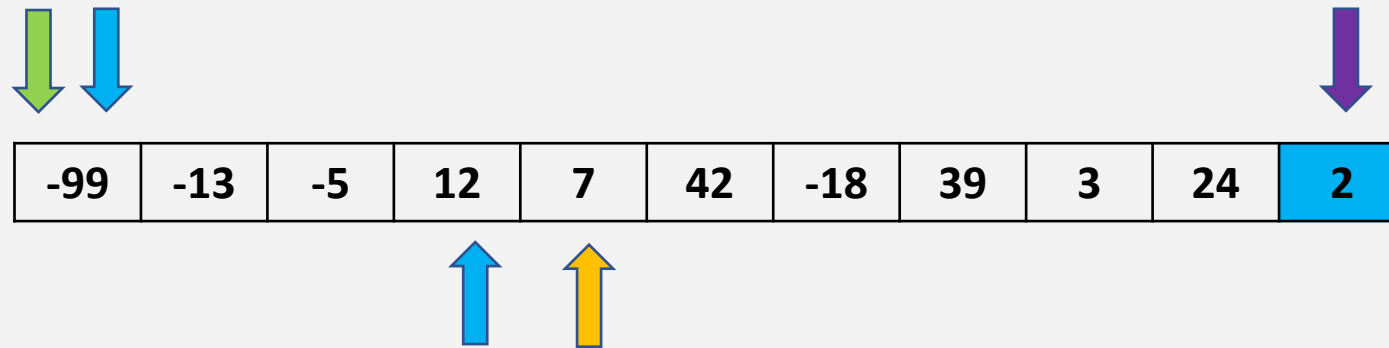
Échanger les valeurs








↓ indicePivot ↓ indicePremier ↓ indiceDernier ↑ indiceValeurARanger ↑ futurIndicePivot

Partitionnement optimisé

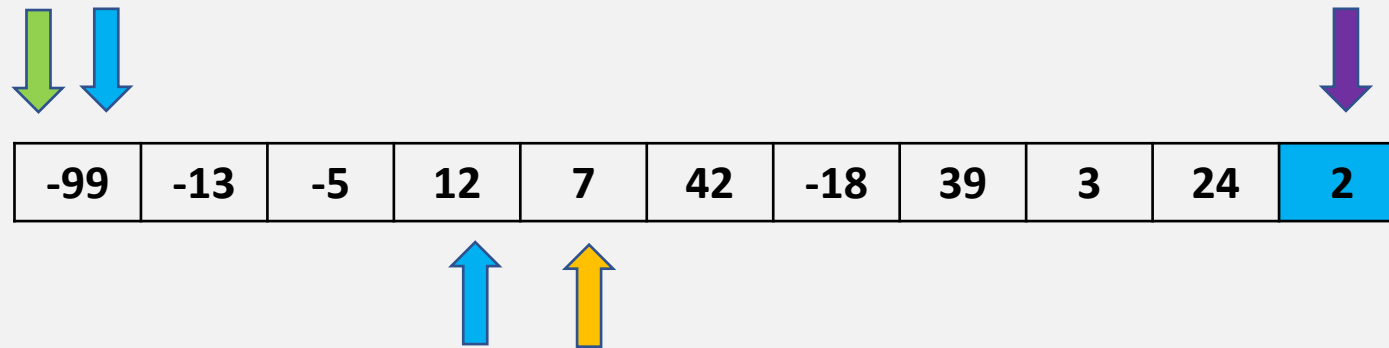
Avancer futurIndicePivot








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

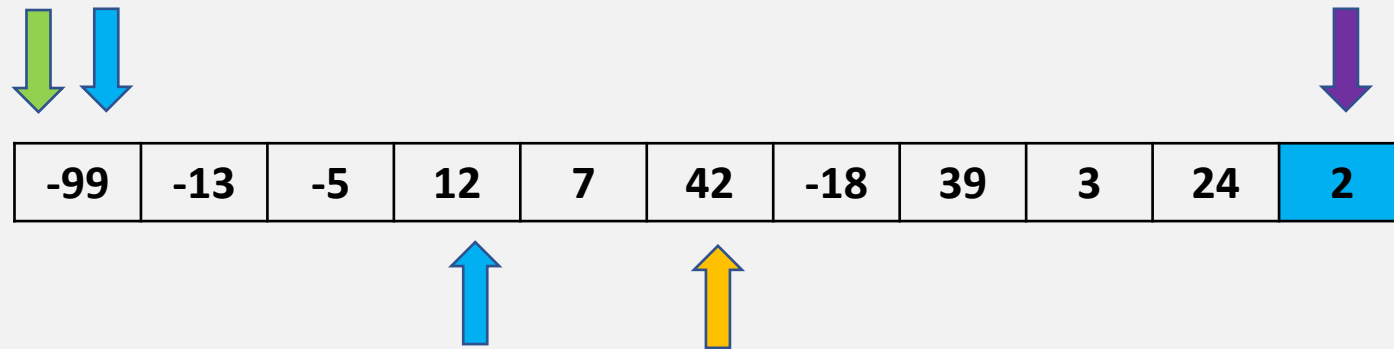
$7 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

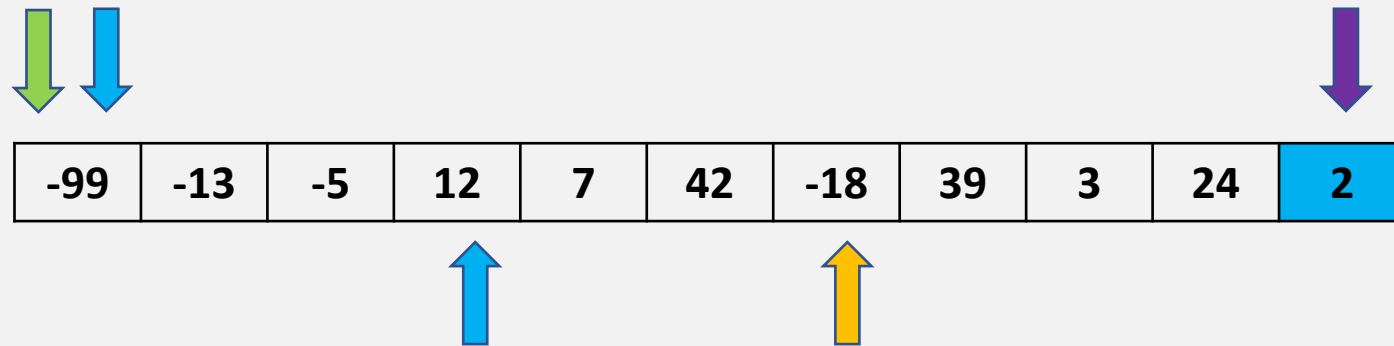
$42 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

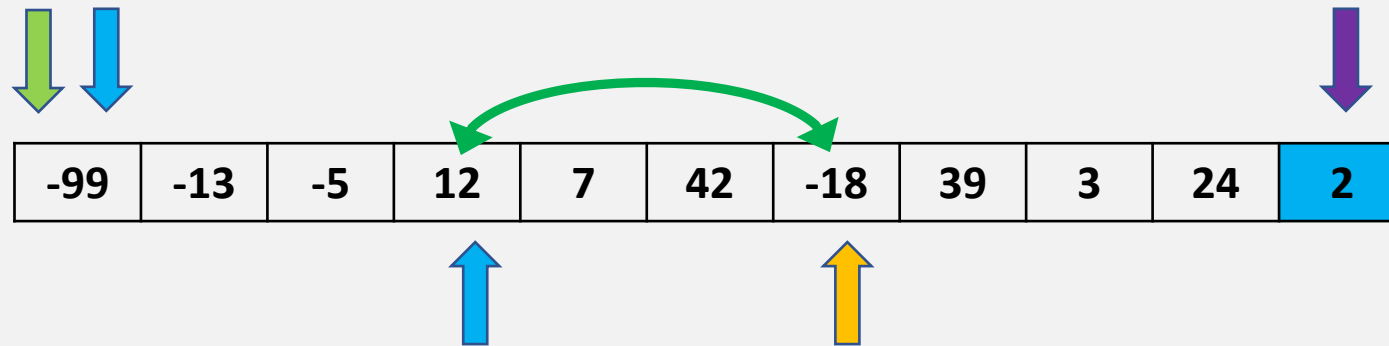
$-18 \leq 2 \Rightarrow$ vrai



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

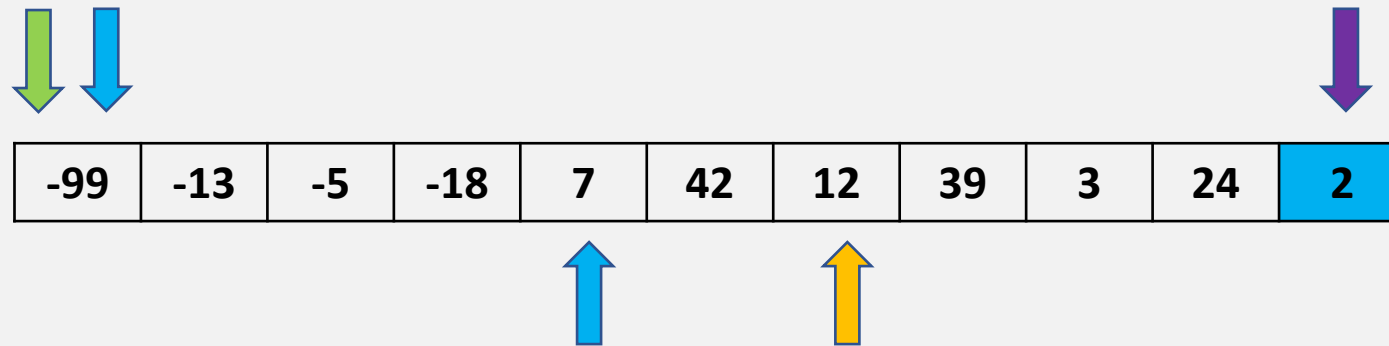
Échanger les valeurs








↓ indicePivot ↓ indicePremier ↓ indiceDernier ↑ indiceValeurARanger ↑ futurIndicePivot

Partitionnement optimisé

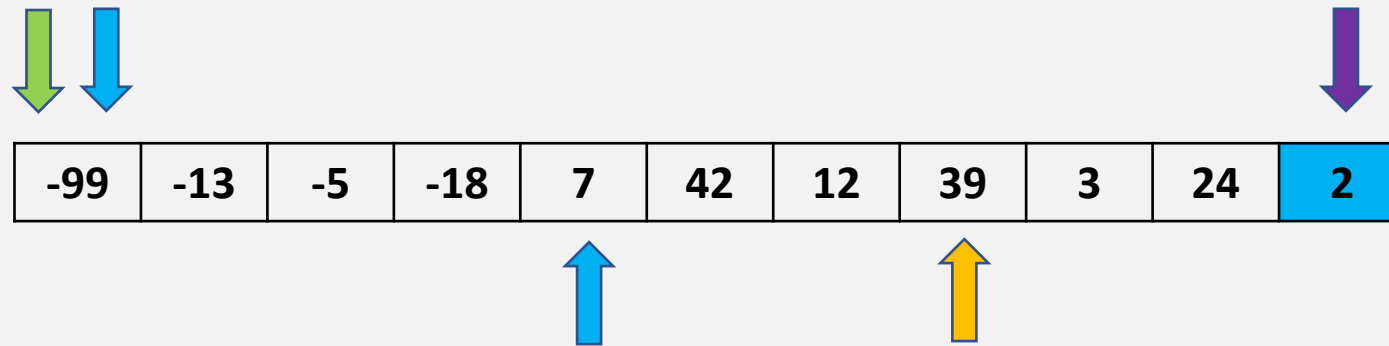
Avancer futurIndicePivot








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

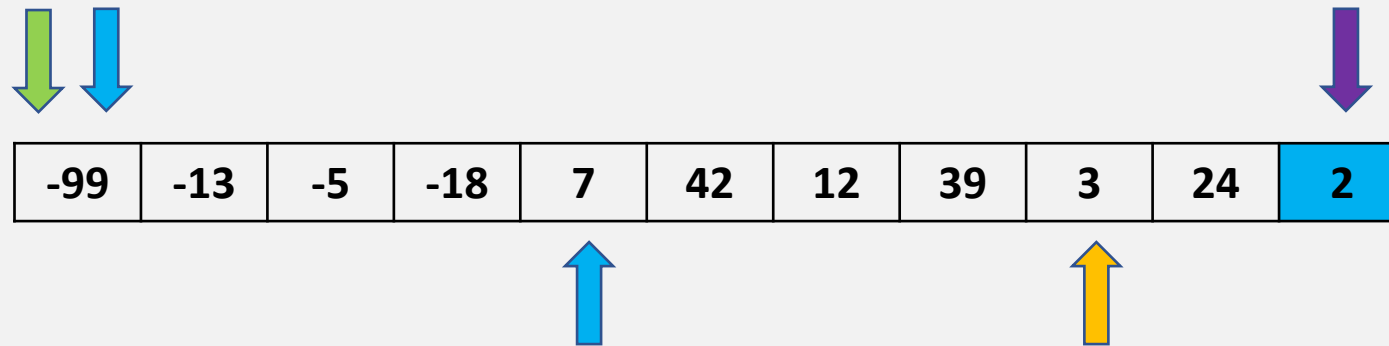
$39 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

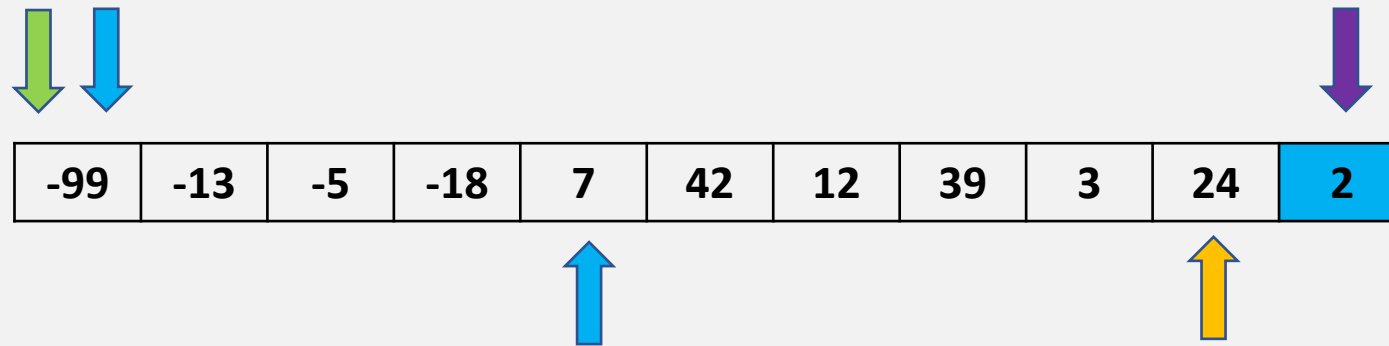
$3 \leq 2 \Rightarrow$ faux








 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

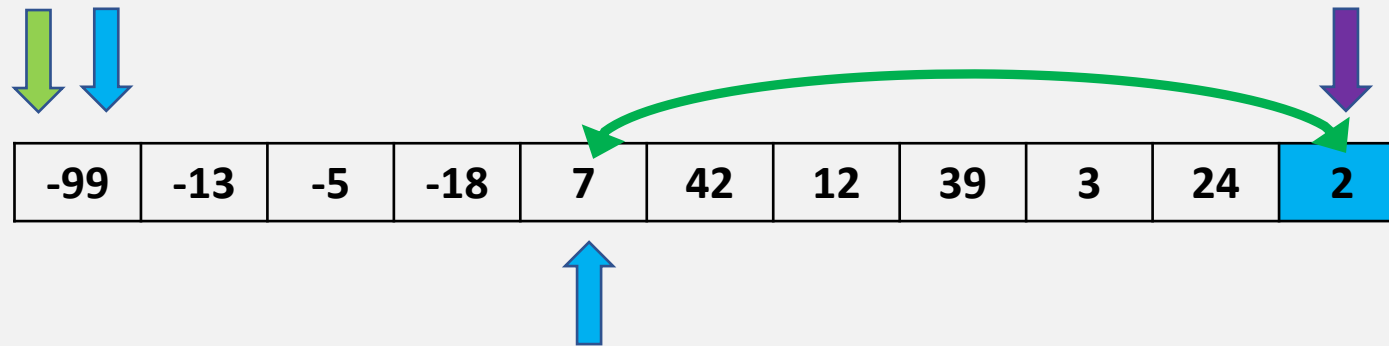
$24 \leq 2 \Rightarrow$ faux



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

Partitionnement optimisé

Échanger la valeur du pivot avec le celle du futurPivot

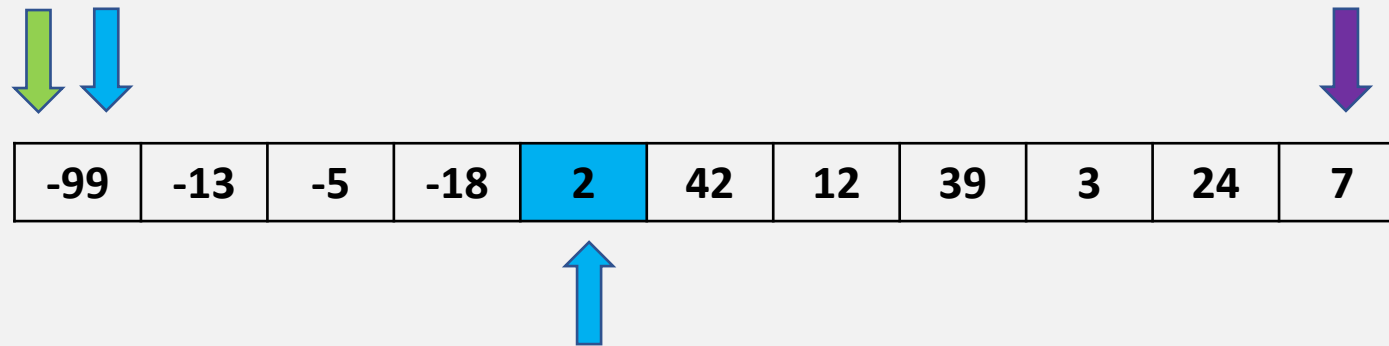







↓ indicePivot ↓ indicePremier ↓ indiceDernier ↑ indiceValeurARanger ↑ futurIndicePivot

Partitionnement optimisé

Échanger la valeur du pivot avec le celle du futurPivot

STOP



 indicePivot  indicePremier  indiceDernier  indiceValeurARanger  futurIndicePivot

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;

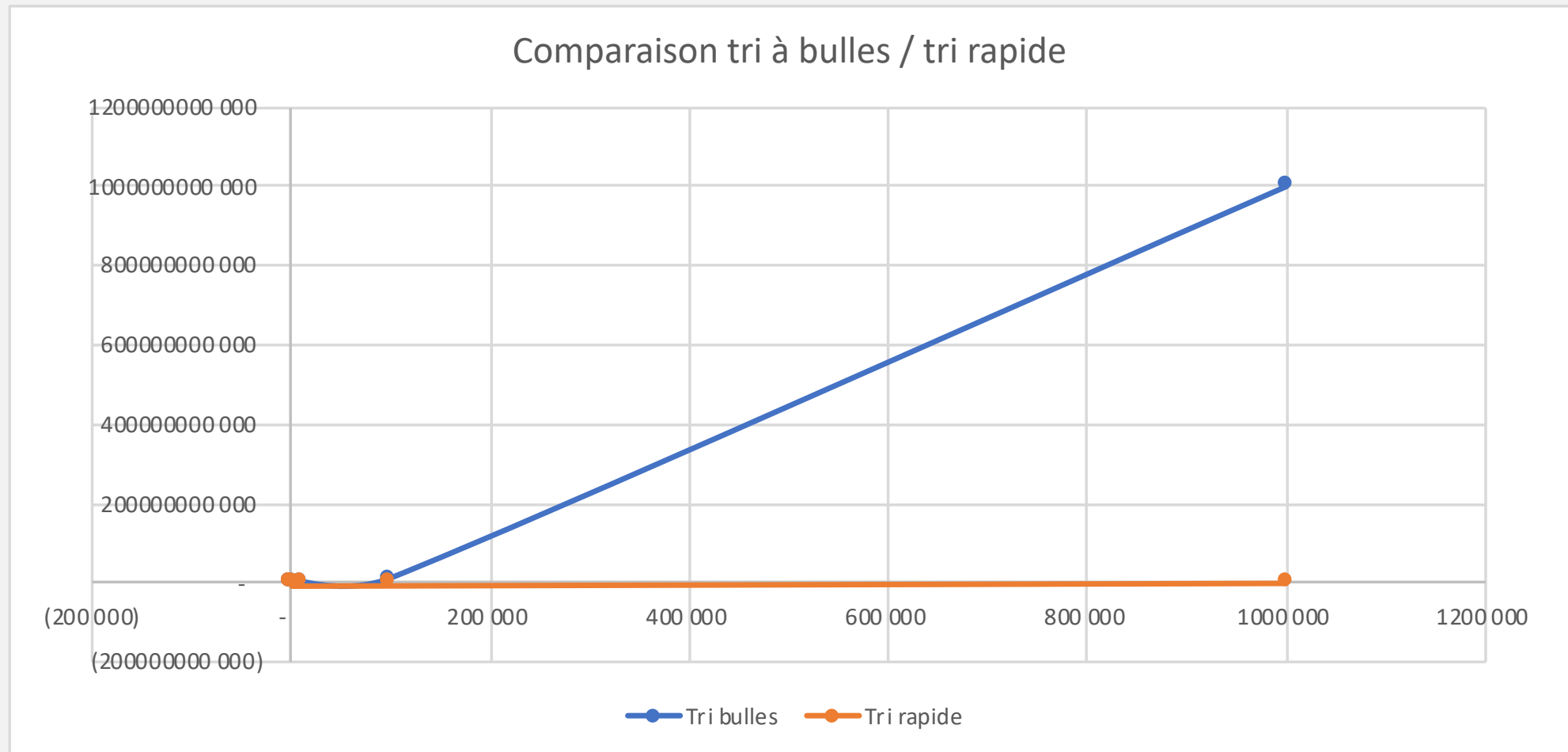
    ancienneValeur = p_valeurs[p_indicePivot];
    p_valeurs[p_indicePivot] = p_valeurs[p_indiceDernier];
    p_valeurs[p_indiceDernier] = ancienneValeur;

    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;
        }
    }

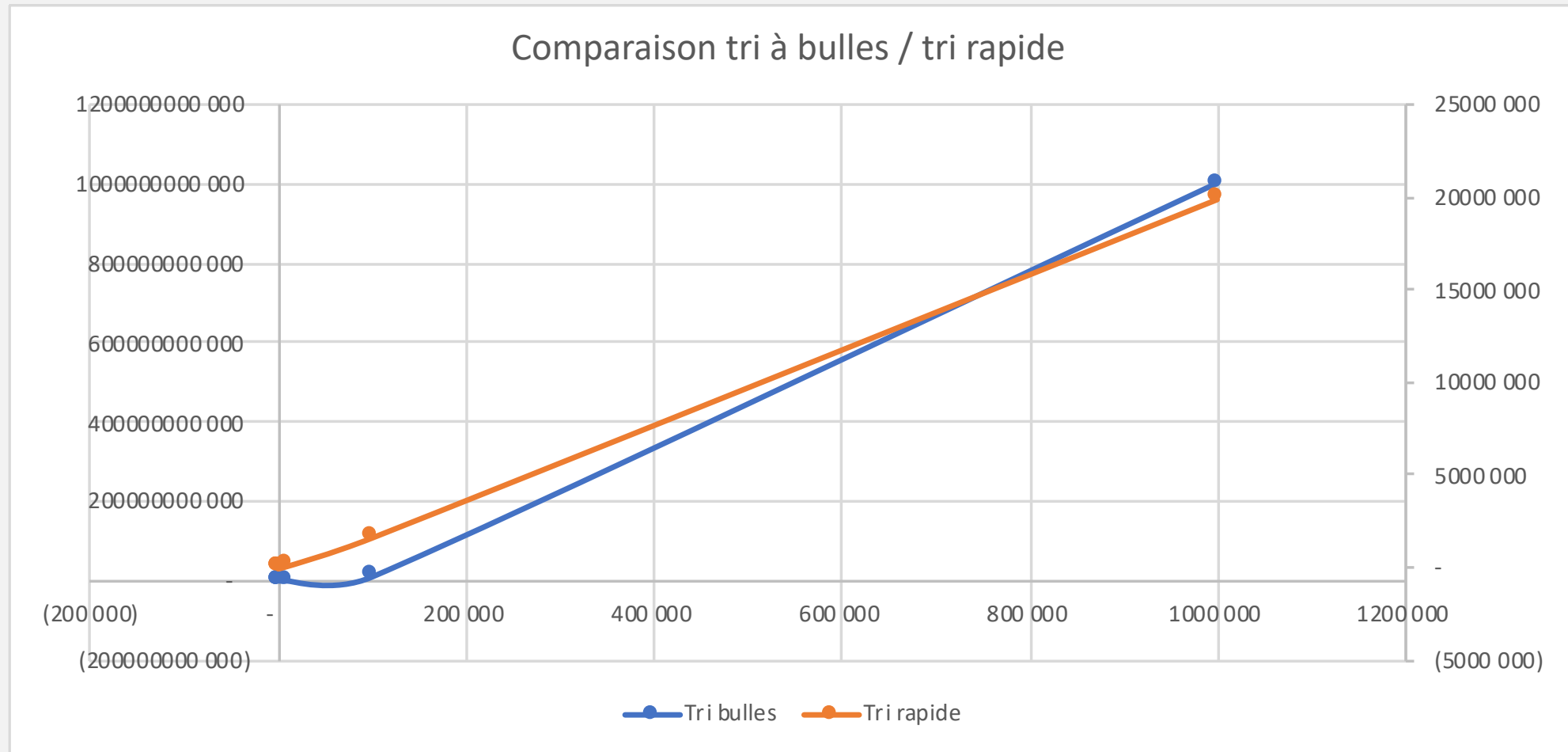
    ancienneValeur = p_valeurs[futurIndicePivot];
    p_valeurs[futurIndicePivot] = p_valeurs[p_indiceDernier];
    p_valeurs[p_indiceDernier] = ancienneValeur;

    renvoyer futurIndicePivot;
}
```

Intuition sur ces deux algorithmes



Intuition sur ces deux algorithmes – Avec deux axes



Algorithme intuitif V. 1 – Rappels

```
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;  
}
```

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

```
booléen RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {  
    booléen 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

```
bool RechercherValeurOptimisee(int[] p_collection, int p_valeurAChercher) {  
    bool 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

$3 == -5 \Rightarrow$ **faux**

$3 < -5 \Rightarrow$ **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

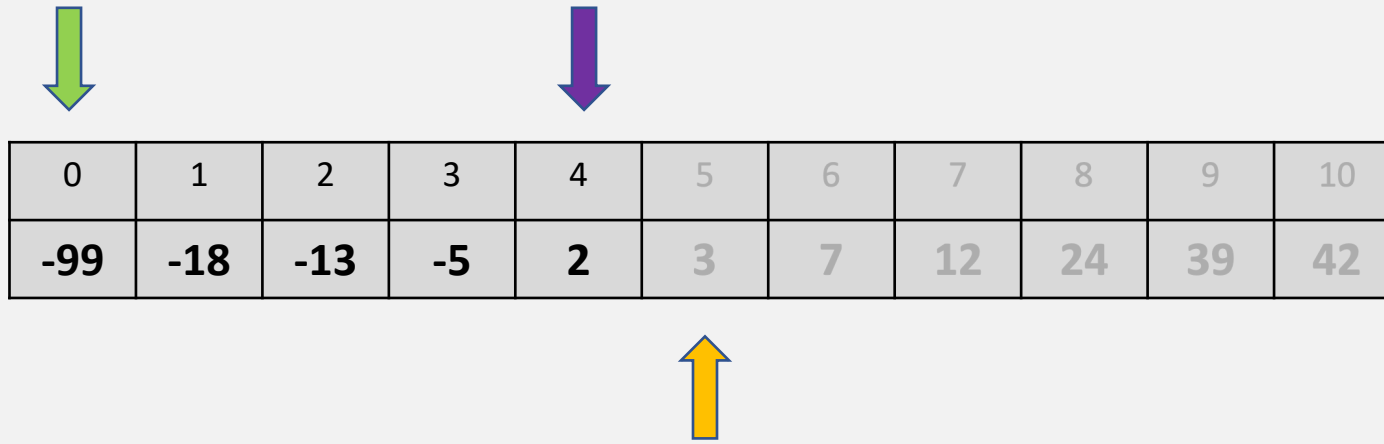
4 comparaisons

Principes de la recherche dichotomique

Recherche de -5




$3 == -5 \Rightarrow$ **faux**

$3 < -5 \Rightarrow$ **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

$-13 == -5 \Rightarrow$ **faux**

$-13 < -5 \Rightarrow$ **vrai**

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

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

 indicePremier  indiceDernier  indiceMilieu

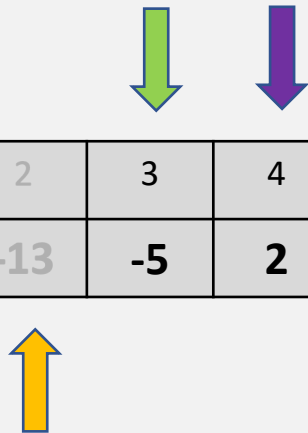
4 comparaisons

Principes de la recherche dichotomique

Recherche de -5

$-13 == -5 \Rightarrow$ **faux**

$-13 < -5 \Rightarrow$ **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

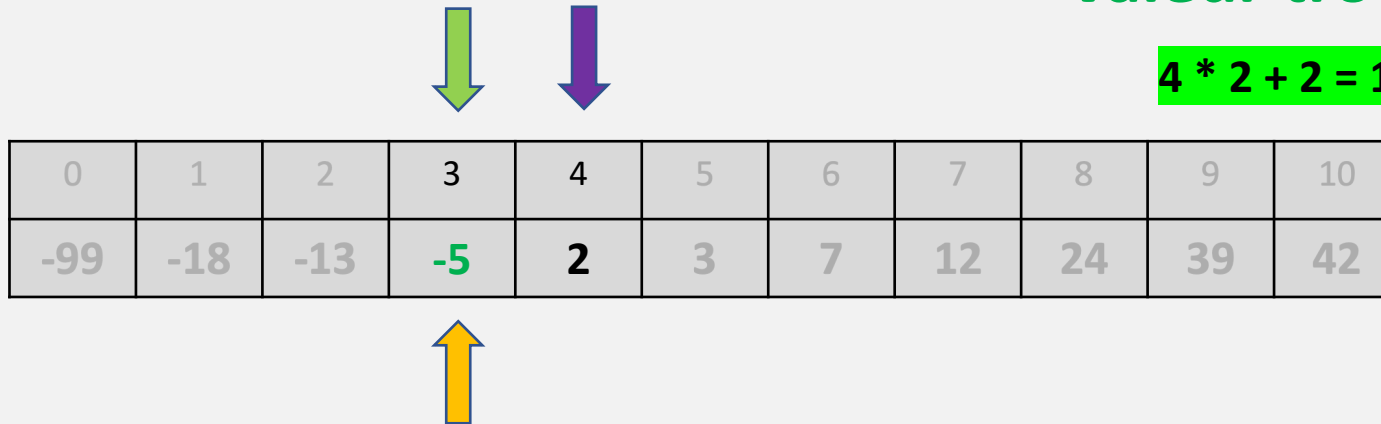
Principes de la recherche dichotomique

Recherche de -5

-5 == -5 => **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

$3 == -3 \Rightarrow$ **faux**

$3 < -3 \Rightarrow$ **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

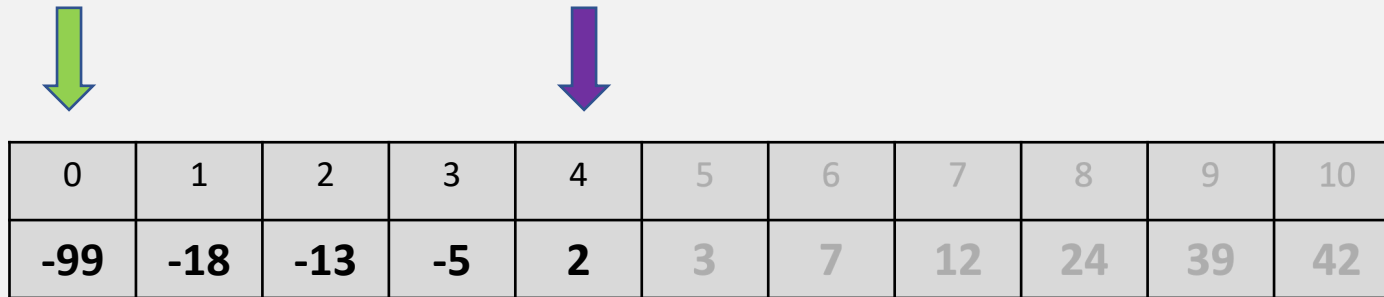
4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$3 == -3 \Rightarrow$ **faux**

$3 < -3 \Rightarrow$ **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



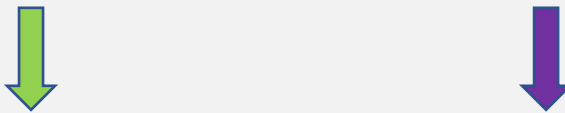
indiceMilieu

Principes de la recherche dichotomique

Recherche de -3




$-13 == -3 \Rightarrow$ **faux**

$-13 < -3 \Rightarrow$ **vrai**



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

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

 indicePremier  indiceDernier  indiceMilieu

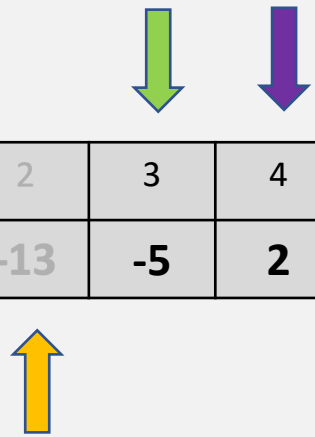
4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$-13 == -3 \Rightarrow$ **faux**

$-13 < -3 \Rightarrow$ **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



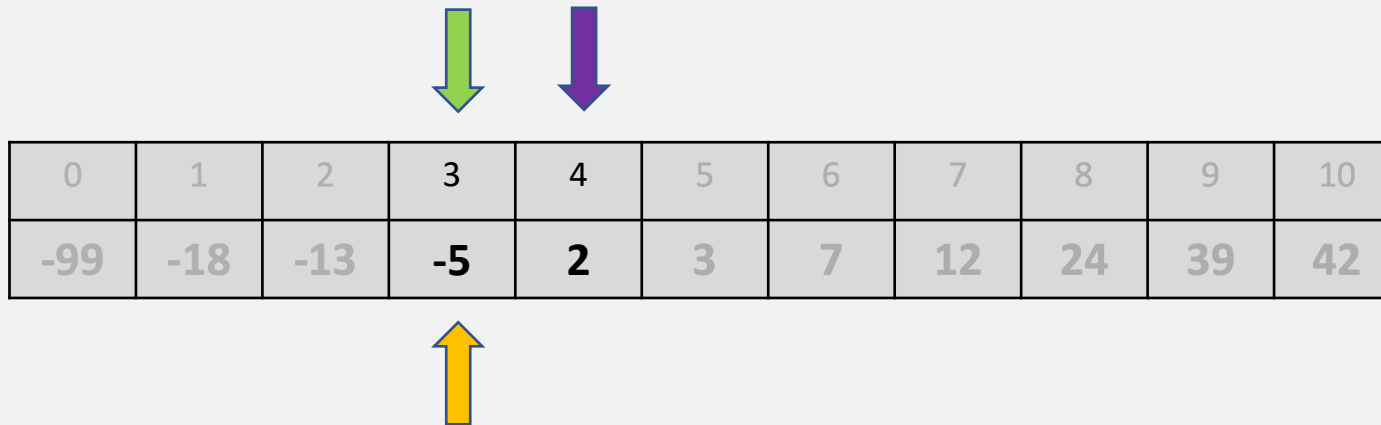
indiceMilieu

Principes de la recherche dichotomique

Recherche de -3




$-5 == -3 \Rightarrow$ **faux**

$-5 < -3 \Rightarrow$ **vrai**



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

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

 indicePremier  indiceDernier  indiceMilieu

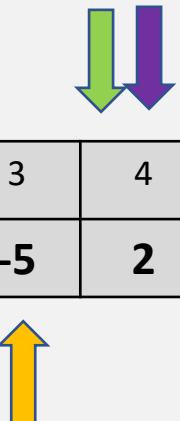
4 comparaisons

Principes de la recherche dichotomique

Recherche de -3

$-5 == -3 \Rightarrow$ **faux**

$-5 < -3 \Rightarrow$ **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 1 élément



indicePremier



indiceDernier




indiceMilieu

Principes de la recherche dichotomique


Recherche de -3

$2 == -3 \Rightarrow$ **faux**

$2 < -3 \Rightarrow$ **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

$\text{indicePremier} \leq \text{indiceDernier} \Rightarrow$ **faux**

STOP \Rightarrow 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

```
bool RechercherValeurDichomie(int p_collection, int p_valeurAChercher) {
    bool 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

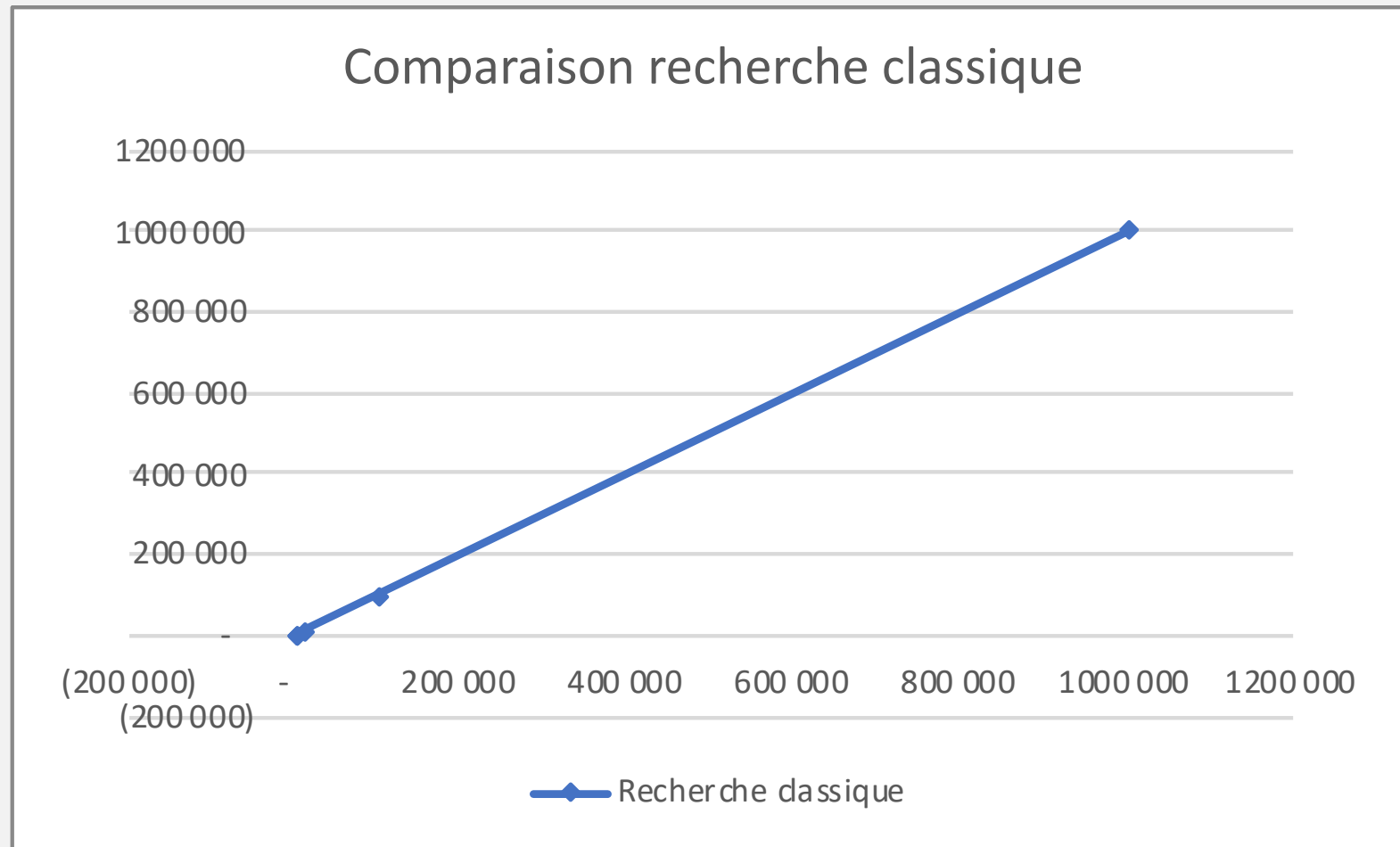
```
booléen RechercherValeurDichomie(int p_collection, int p_valeurAChercher) {
    booléen estTrouvee = faux;
    entier indicePremier = 0;
    entier indiceDernier = p_collection.Capacité - 1;
    entier indiceMilieu = 0;

    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;
}
```

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 **$O(\log_2(n))$** .

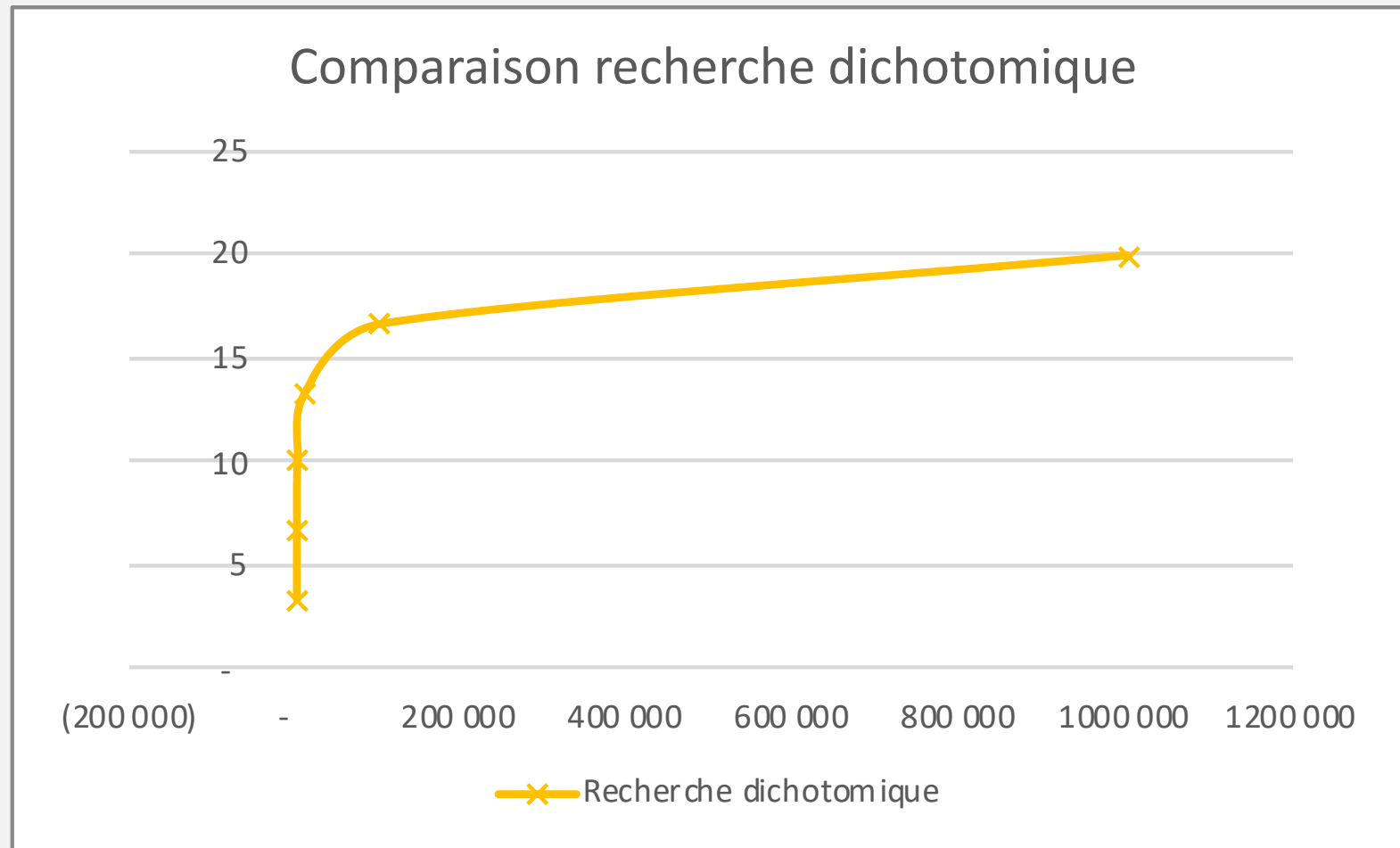
Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons

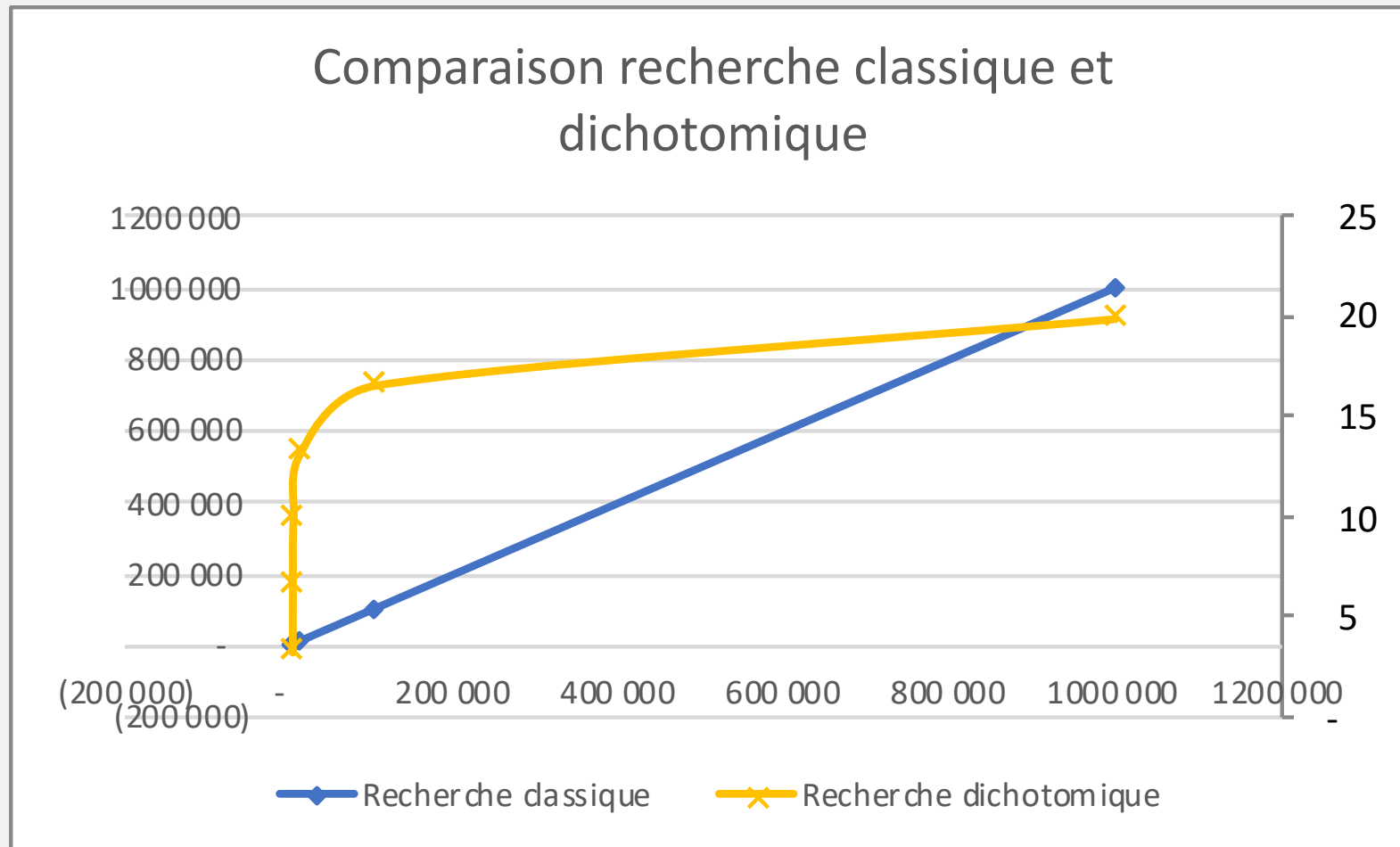
Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons

Intuition sur la complexité des algorithmes



x : quantité de données

y : nombre de comparaisons