


chapitre 3.-

Modèles d'algorithmes sur collections d'éléments homogènes

à connaître par cœur



Ressource R1.01 : Initiation au développement - Partie 2

Plan

<u>1.- Introduction</u>3
2.- Rappels	
– <u>Structure ordonnée</u>5
– <u>Accès direct</u>6
– <u>Accès séquentiel</u>7
– <u>Accès dynamique</u>8
<i>Parcours séquentiels</i>	
<u>3.- Parcours séquentiel complet avec traitement systématique</u>9
<u>4.- Parcours séquentiel complet avec traitement conditionné</u>12
<i>Recherches de 1ère occurrence</i>	
<u>5.- Recherche séquentielle</u>15
<u>6.- Recherche dichotomique</u>19
<u>7.- Tris sur collections à accès direct</u>30
– <u>Introduction – Propriétés des tris</u>31
– <u>Tri par sélection de place</u>34
– <u>Tri de la bulle</u>44
 <u>8.- Parcours séquentiel parallèle de deux collections ordonnées</u>	 ...57

1.- Introduction – But du chapitre (1/2)

- ❑ En Algorithmique/Programmation, il existe des problèmes **récurrents** posés sur des collections d'éléments **homogènes**
 - Parcours d'une collection
 - Recherche de la première occurrence d'un élément dans une collection
 - Tri des éléments d'une collection
 - Fusion de deux collections
 - ...
- ❑ Des solutions connues sous formes de modèles d'algorithmes, validées et réutilisables, ont été mises au point pour répondre à ces questions de manière efficace.
- ❑ Mais un modèle d'algorithme ne peut s'appliquer à n'importe quelle structure de données : celle-ci doit vérifier un certain nombre de **propriétés**.
- ❑ Parmi ces propriétés, on distingue :
 - L'existence d'un **ordre** entre les éléments
 - Les **mécanismes d'accès** à la structure de données :
Ainsi, les mécanismes d'accès aux données disponibles sur la structure de données doivent être compatibles avec ceux utilisés par l'algorithme

1.- Introduction – But du chapitre (2/2)

- ❑ Trois principaux **mécanismes d'accès** à une structure ordonnée
= *moyens* mis à disposition par la structure de données pour accéder à ses éléments
 - Accès séquentiel
 - **Accès direct** → *utilisé en R1.01- partie 1*
 - Accès dynamique
- ❑ Algorithmes vus... **sur une structure ordonnée à accès direct**
 - Parcours séquentiel complet avec traitement systématique
 - Parcours séquentiel complet avec traitement conditionné
 - Recherche séquentielle de la première occurrence d'un élément
- ❑ Algorithmes restant à découvrir = **contenu du présent chapitre**
 - Recherche dichotomique de la première occurrence d'un élément dans une structure ordonnée **à accès direct**
 - Algorithmes de parcours séquentiel **généralisés**, pour une structure ordonnée à **accès séquentiel ou direct** (vu en 1^{ère} partie pour structure à accès direct uniquement)
 - Tris d'une collection d'éléments (il en existe plusieurs – nous verrons ceux applicables aux collections **à accès direct**)
 - Fusion de collections triées

2.- Rappels – Structure ordonnée

□ Définition

Une structure est dite **ordonnée** lorsqu'il est possible de définir une relation d'ordre entre ses éléments permettant de les comparer.

Cette relation d'ordre peut s'appuyer, par exemple :

- Sur la position des éléments dans la structure
- Sur l'ordre chronologique d'arrivée des éléments dans la structure
-

□ Exemples

- Un tableau est une structure ordonnée : l'ordre se base sur le rang (indice) de l'élément dans le tableau
- Un enregistrement n'est pas une structure ordonnée : il n'existe pas de relation d'ordre permettant de comparer ses champs entre eux, les champs sont juste 'regroupés' dans l'enregistrement

2.- Rappels - Mécanismes d'accès (1/3)

❑ Mécanisme d'accès direct

– Définition

Une structure ordonnée propose un **mécanisme d'accès direct** à ses éléments lorsque l'accès à un élément peut se faire indépendamment des autres éléments.

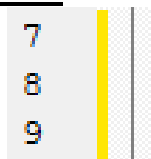
Le mécanisme d'accès direct peut se baser, par exemple, sur le **rang** ou **position** de chaque élément dans la structure.

– Propriété

Le mécanisme d'accès direct est souvent accompagné d'un mécanisme permettant de déterminer **le nombre** d'éléments stockés.

– Exemples de structures de données à accès direct :

Chaînes de caractères, fichiers à organisation relative, certaines listes, dictionnaires (map), **tableaux** :

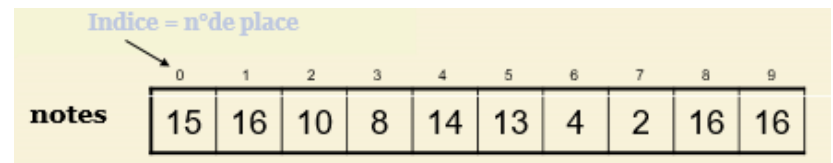


7
8
9

```
const unsigned int TAILLETAB = 10;  
// taille = nbre d'éléments du tableau  
unsigned int notes[TAILLETAB];
```

*accès direct à un
élément grâce à
son **indice***

```
notes[0] = 15;  
notes[1] = 16;  
//...  
notes[9] = 16;
```



Indice = n°de place									
0	1	2	3	4	5	6	7	8	9
15	16	10	8	14	13	4	2	16	16

2.- Rappels - Mécanismes d'accès (2/3)

❑ Mécanisme d'accès séquentiel

– Définition

Une structure ordonnée propose un **mécanisme d'accès séquentiel** lorsqu'elle combine les 3 dispositifs suivants pour accéder à ses éléments :

- (1) dispositif d'accès au premier élément de la structure (et parfois au dernier)
- (2) à partir de la position courante, dispositif d'accès à l'élément suivant (et parfois au précédent)
- (3) dispositif permettant la détection de la fin de structure

Avec ce mécanisme, on ne peut accéder au $n^{\text{ième}}$ élément qu'en passant par les $n-1$ qui le précèdent.

– Exemples de structures de données à accès séquentiel :

fichiers séquentiels (dont fichiers texte), piles, listes,

files d'attente (premier arrivé = premier servi) :

Queue de file
= dernier arrivé

Tête de file
= premier arrivé



2.- Rappels - Mécanismes d'accès (3/3)

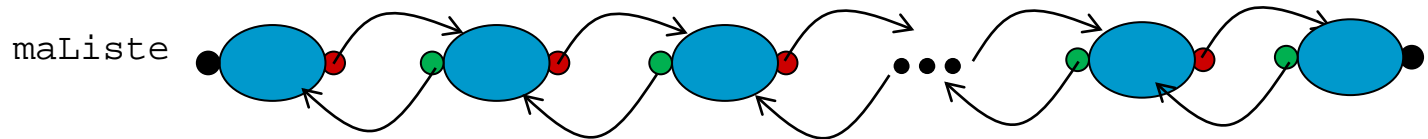
□ Mécanisme d'accès dynamique

– Définition

Une structure ordonnée propose un **mécanisme d'accès dynamique** lorsqu'elle met à disposition les mécanismes d'accès direct **et** séquentiel pour accéder à ses éléments.

– Exemples de structures de données proposant un accès dynamique :

certaines fichiers, **certaines listes** :



`allerAuPremier (maListe)`
`allerAuSuivant (maListe)`
`allerAuPrécédent (maListe)`

} accès séquentiel

`allerAElement (maListe, 3)`
`allerAuDernier (maListe)`
`allerAElement (maListe, positionDuDernier(maListe) - 1)`

} accès direct

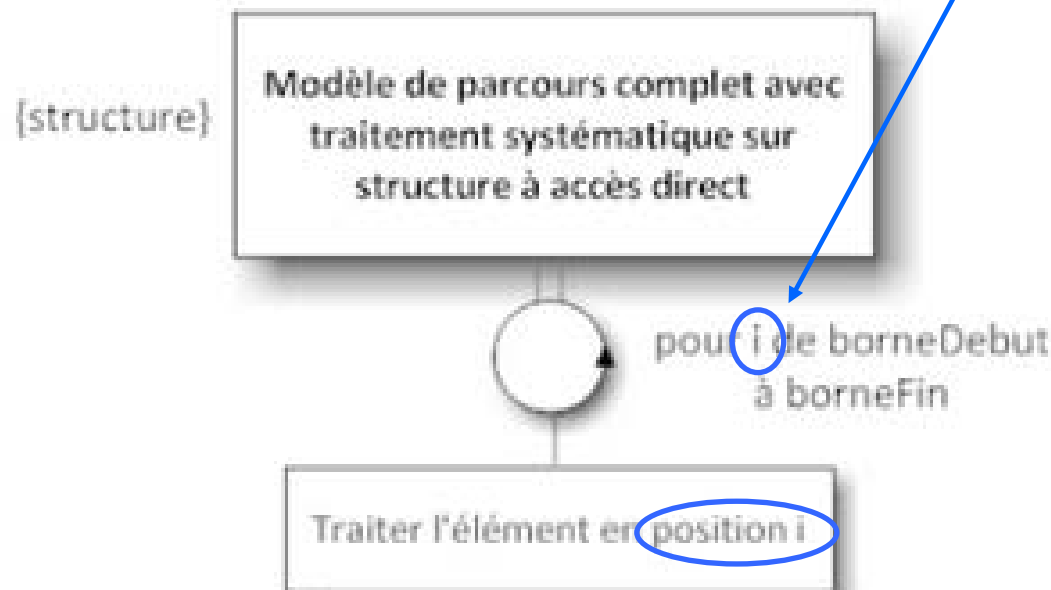
3.- Parcours séquentiel complet avec traitement systématique (1/3)

- ❑ Nom du modèle : **parcours séquentiel complet avec traitement systématique sur la structure**
- ❑ Quand l'utiliser :
 - Lorsque l'on doit appliquer un même traitement à **tous** les éléments d'une collection
- ❑ Exemples :
 - Remplir toutes les cases d'un tableau
 - Incrémenter de 1 la valeur de chaque élément d'une file
 - Afficher toutes les lignes d'un fichier de texte
- ❑ Pré-conditions :
 - **La collection doit proposer un mécanisme d'accès séquentiel ou direct**
- ❑ A préciser :
 - Le sens du parcours. Par exemple : du plus ancien (indice plus élevé) au plus récent (indice plus petit); ou l'inverse
- ❑ Remarques
 - Les éléments de la collection peuvent être triés ou pas selon leur contenu
 - La collection peut comporter des doublons ou pas

3.- Parcours séquentiel complet avec traitement systématique (2/3)

- Algorithme ***appliqué à une structure à accès direct***
(*rappel cours R1.01 – partie 1 !*)

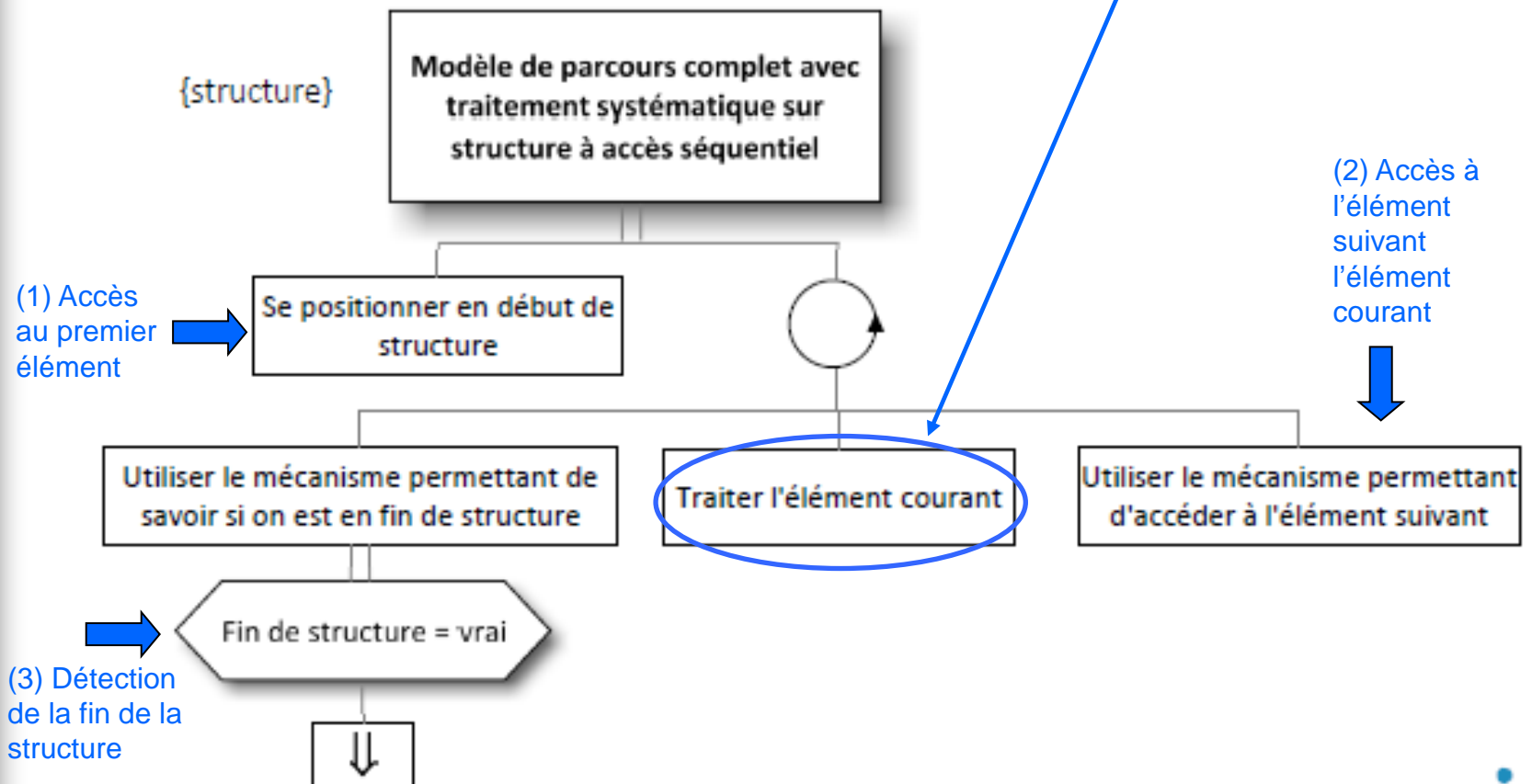
- On accède **directement** à tous les éléments situés entre une borne de départ et une borne d'arrivée, grâce à une **position** ou **indice** ou **rang** (ici **i**)



3.- Parcours séquentiel complet avec traitement systématique (3/3)



- ❑ Algorithme **généralisé** cad **applicable à une structure à accès direct ou séquentiel**, grâce à la notion d'**élément courant** : l'élément de la structure sur lequel on se trouve en ce moment.



4.- Parcours séquentiel complet avec traitement conditionné (1/3)

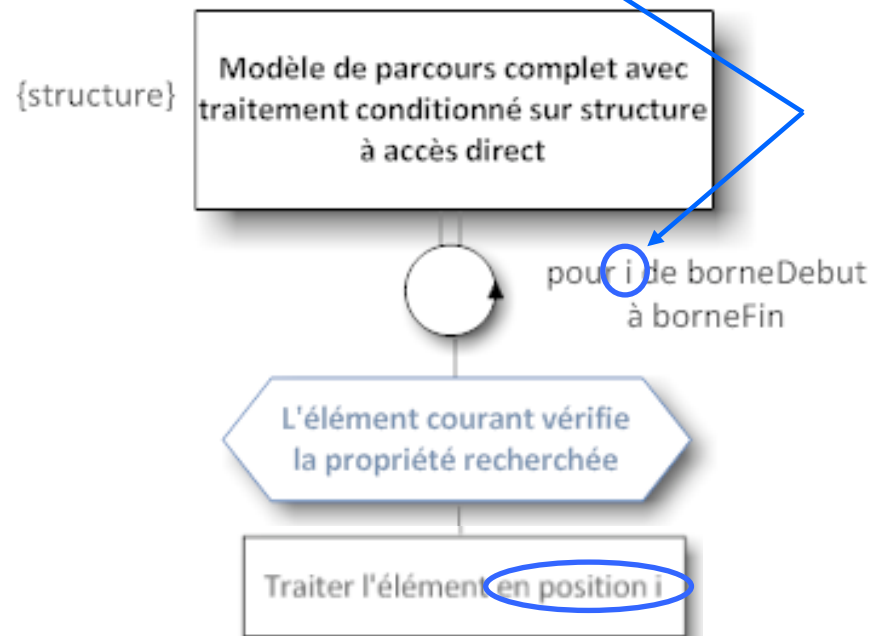
- ❑ Nom du modèle : **parcours séquentiel complet avec traitement conditionné sur la structure**
- ❑ Quand l'utiliser :

Lorsque l'on doit appliquer un traitement à **certains** éléments d'une collection : ceux vérifiant une propriété particulière.
- ❑ Exemples :
 - Afficher les *voyelles* d'une chaîne de caractères
 - Incrémenter de 1 la valeur des éléments d'une file de valeur < 10
 - Afficher les lignes d'un fichier texte contenant le mot « algorithmique »
- ❑ Pré-conditions :
 - La collection doit proposer un mécanisme d'accès **séquentiel** ou **direct**
- ❑ A préciser :
 - Le sens du parcours. Par exemple : du plus ancien (indice plus élevé) au plus récent (indice plus petit); ou l'inverse
- ❑ Remarques
 - Les éléments de la collection peuvent être triés ou pas selon leur contenu
 - La collection peut comporter des doublons ou pas

4.- Parcours séquentiel complet avec traitement conditionné (2/3)

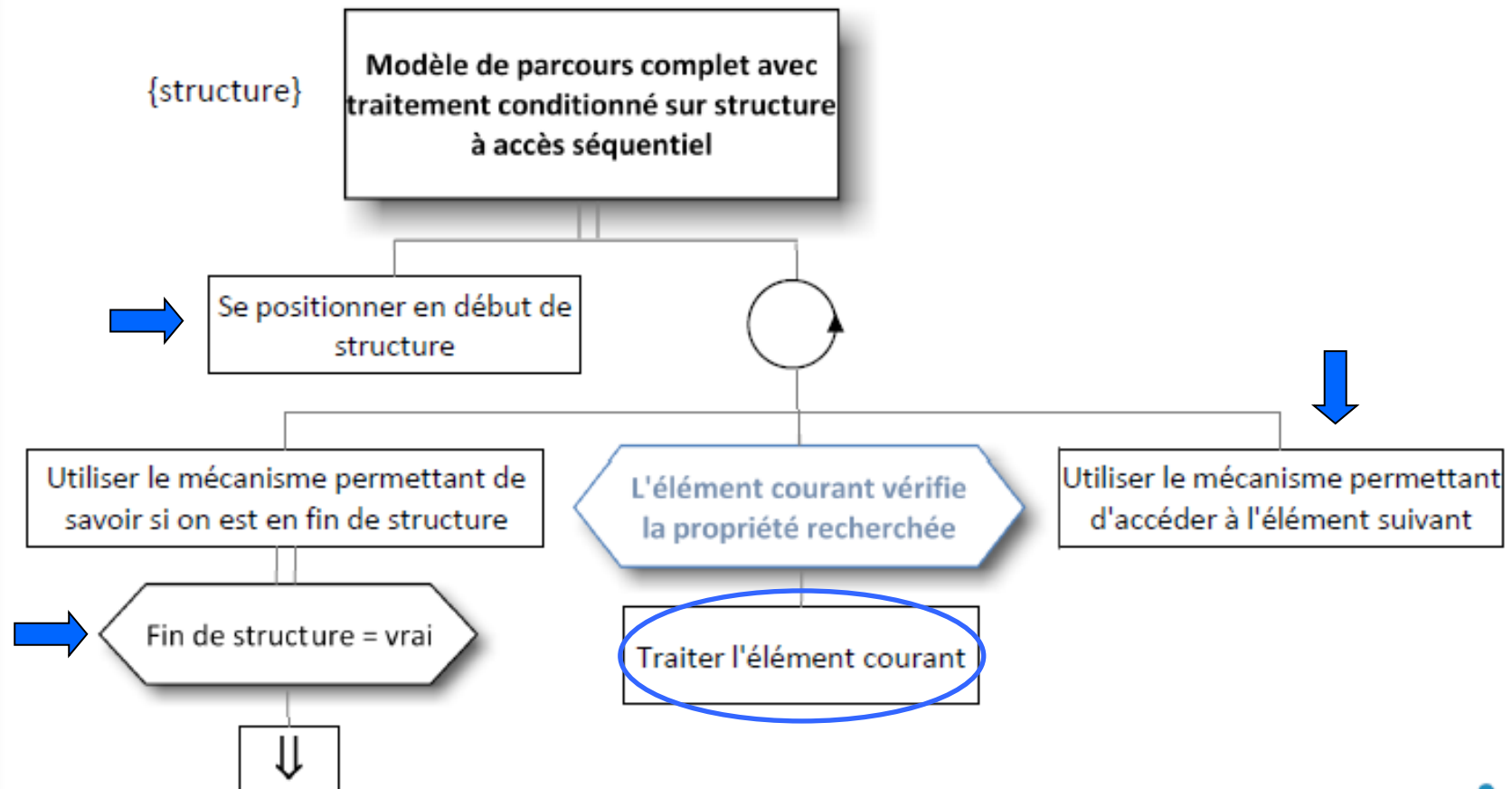
- ❑ Algorithme ***appliqué à une structure à accès direct***
(rappel cours R1.01 – partie 1 !)

- On accède **directement** à tous les éléments situés entre une borne de départ et une borne d'arrivée grâce à une **position** ou **indice** ou **rang**



4.- Parcours séquentiel complet avec traitement conditionné (3/3)

- ❑ Algorithme **généralisé** c'est-à-dire **applicable à une structure à accès direct ou séquentiel**



5.- Recherche séquentielle de première occurrence (1/4)

- ❑ Nom du modèle : recherche séquentielle de première occurrence
- ❑ Quand l'utiliser :
 - Lorsque l'on recherche « *le premier* » élément d'un ensemble satisfaisant une propriété particulière.
 - Ce « premier » élément peut être différent selon le sens de parcours de la collection.

- ❑ Exemple :

0	1	2	3	4	5	6	7	8	9
7	10	-6	7	-2	9	1	0	8	-6

- chercher la position de la première occurrence de la valeur 7 dans ce tableau : le résultat est **0** si on démarre la recherche au plus petit indice, ou bien **3** si on démarre la recherche au plus grand indice
- ❑ Pré-conditions :
 - La collection doit proposer un mécanisme d'accès **séquentiel** ou **direct**
- ❑ Résultat(s) de l'algorithme
 - **Au minimum**, l'algorithme produit un indicateur qui précise si la recherche a abouti ou pas.
 - D'autres résultats peuvent être produits, comme par exemple la position de l'élément trouvé.

5.- Recherche séquentielle de première occurrence (2/4)

❑ Remarques

- Sur les **éléments** de la structure :
 - Ils peuvent être triés, ou pas, selon leur contenu.
 - Il peut y avoir des doublons
- Sur la **première occurrence trouvée** :
 - Elle fait référence **au 1er élément rencontré** lors du parcours de recherche. Ce premier élément rencontré peut donc **varier selon le sens de parcours** de la structure (du plus ancien au plus récent, ou inversement)...
- Sur les **conditions de fin de parcours** :

Ce type d'algorithme a toujours **au moins deux conditions d'arrêt** :

 - ✓ recherche *fructueuse* (l'élément cherché a été trouvé)
 - ou bien
 - ✗ recherche *infructueuse* (on a parcouru toute la structure et l'élément cherché n'a pas été trouvé).
- Sur le **modèle d'algorithme** :

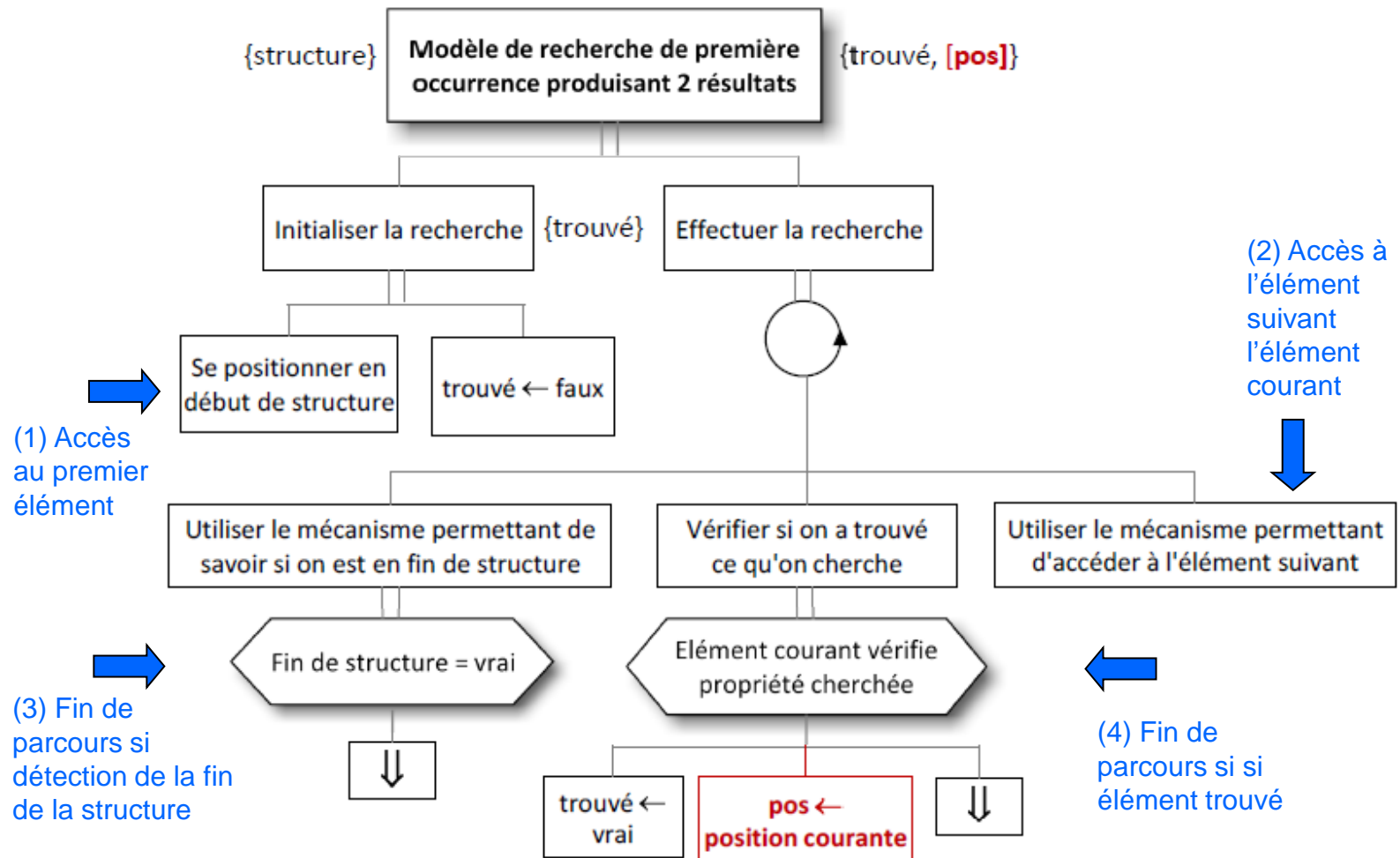
Il est identique pour une structure à accès séquentiel et pour une structure à accès direct.

5.- Recherche séquentielle de première occurrence (3/4)



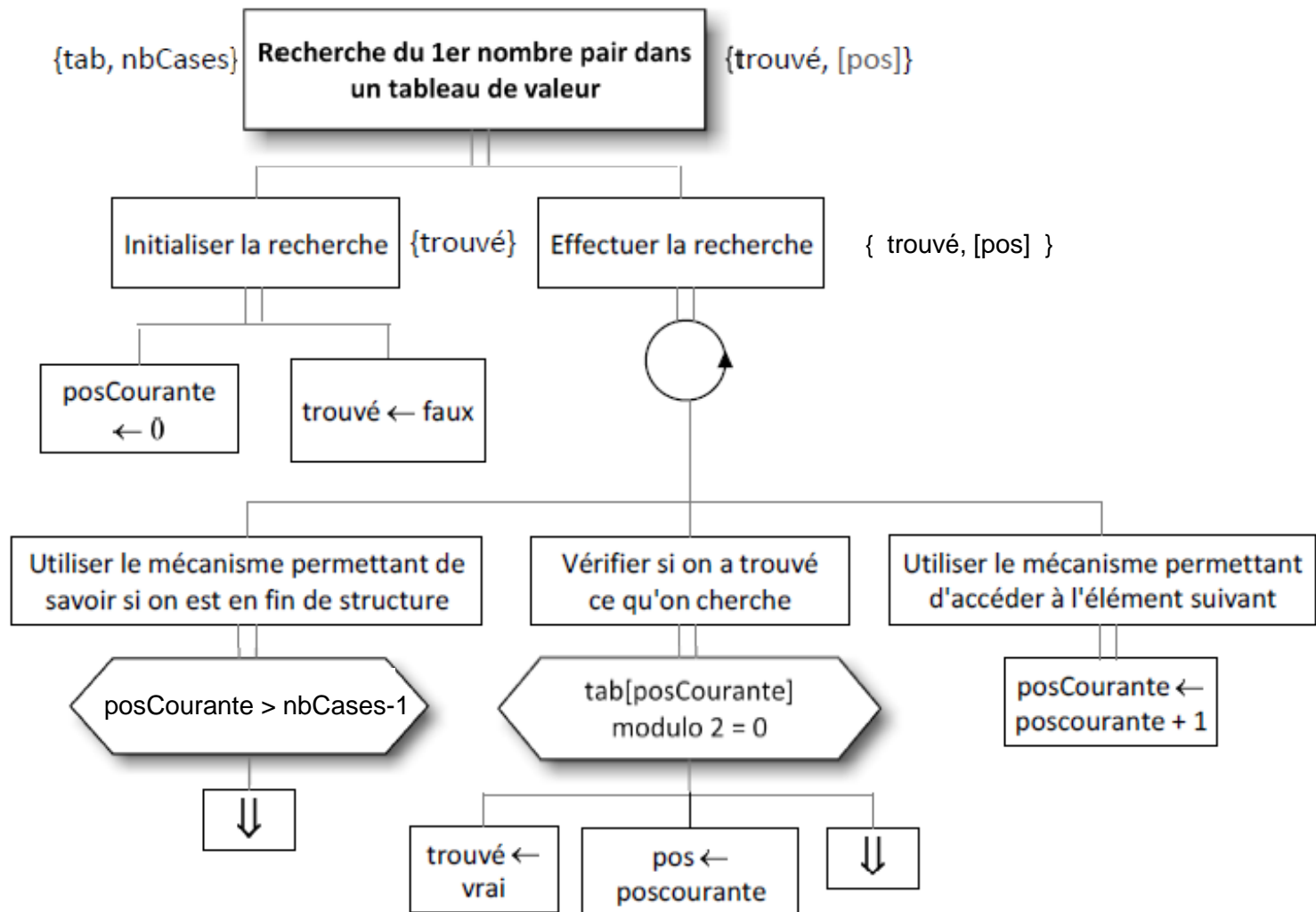
❑ Algorithme avec 2 résultats : indicateur et position

(C'est celui déjà vu dans R1.01-partie1, pour les variantes, se rapporter à ce cours)



5.- Recherche séquentielle de première occurrence (4/4)

□ Exemple



6.- Recherche dichotomique de première occurrence (1/10)

- ❑ Nom du modèle : recherche dichotomique de première occurrence
- ❑ Quand l'utiliser :
 - Lorsque l'on recherche le « premier élément » de la collection satisfaisant une propriété particulière.
 - Ce « premier » élément peut être différent selon la manière de parcourir la collection.
- ❑ Pré-conditions :
 - Les éléments de la collection doivent être **triés** selon le critère correspondant à la propriété cherchée.
 - La collection doit proposer un mécanisme d'accès **direct**
- ❑ Remarques :
 - L'ordre des éléments peut être croissant ou décroissant
 - L'ordre des éléments peut ne pas être strict (il peut y avoir des doublons)
- ❑ Exemples :
 - Chercher une valeur entière particulière dans un tableau trié d'entiers,
 - Dans un tableau de personnes (enregistrements [nom, prenom, dateNaissance]) trié par âges, chercher le premier élément dont la date de naissance est antérieure au 01/12/2005.

6.- Recherche dichotomique de première occurrence (2/10)

- ❑ Résultat(s) produit(s) par l'algorithme
 - Un **indicateur**, indiquant au minimum si la recherche a **réussi** ou **échoué**
 - Autres résultats possibles : selon l'algorithme, cela peut être la position dans le tableau où cette première occurrence a été trouvée
- ❑ Au moins 2 conditions d'arrêt :
 - une première occurrence de la valeur cherchée a été trouvée (unique ou pas)
 - on a parcouru tout le tableau (l'espace de recherche a été épuisé) sans trouver d'occurrence de la valeur cherchée
- ❑ Principe du parcours dichotomique
 - Il consiste à diviser par 2, à chaque itération, l'espace restant à parcourir.
 - Pour ce faire, la position du prochain élément à analyser (position de l'élément courant) est calculée :
C'est le **milieu** de l'espace de recherche restant à parcourir

6.- Recherche dichotomique de première occurrence (2/10)

- ❑ Résultat(s) produit(s) par l'algorithme
 - Un **indicateur**, indiquant au minimum si la recherche a **réussi** ou **échoué**
 - Autres résultats possibles : selon l'algorithme, cela peut être la position dans le tableau où cette première occurrence a été trouvée
- ❑ Au moins 2 conditions d'arrêt :
 - une première occurrence de la valeur cherchée a été trouvée (unique ou pas)
 - on a parcouru tout le tableau (l'espace de recherche a été épuisé) sans trouver d'occurrence de la valeur cherchée : **borneInf > borneSup**
- ❑ Principe du parcours dichotomique
 - Il consiste à diviser par 2, à chaque itération, l'espace restant à parcourir.
 - Pour ce faire, la position du prochain élément à analyser (position de l'élément courant) est calculée :

C'est le **milieu** de l'espace de recherche restant à parcourir,
c'est-à-dire la **valeur entière de $(\text{borneInf} + \text{borneSup}) / 2$** , où :

 - **borneInf** : borne inférieure de l'espace de recherche restant, recalculée pour restreindre cet espace de recherche
 - **borneSup** : borne supérieure de l'espace de recherche restant, recalculée pour restreindre cet espace de recherche

6.- Recherche dichotomique de première occurrence (3/10)

❑ Exemple d'application aux tableaux simples

Recherche d'une valeur dans un tableau d'entiers **triés par ordre croissant**

0	1	2	3	4	5	6	7	8	9	10
-45	-20	-15	0	10	10	15	25	30	45	60

– Jeu d'essai 1 :

- Valeur cherchée : 25
- Déroulement de la recherche

Itération	début zone recherche	fin zone recherche	milieu
1	0	10	5
2	6	10	8
3	6	7	6
4	7	7	7

0	1	2	3	4	5	6	7	8	9	10
-45	-20	-15	0	10	10	15	25	30	45	60

6	7	8	9	10
15	25	30	45	60

6 - 6	7
15	25

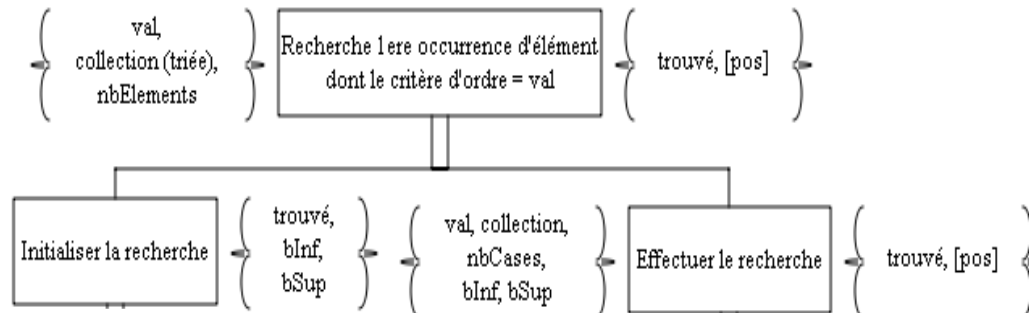
7 - 7
25

- 2 Résultats produits :
 - élément **trouvé** -- en position 7



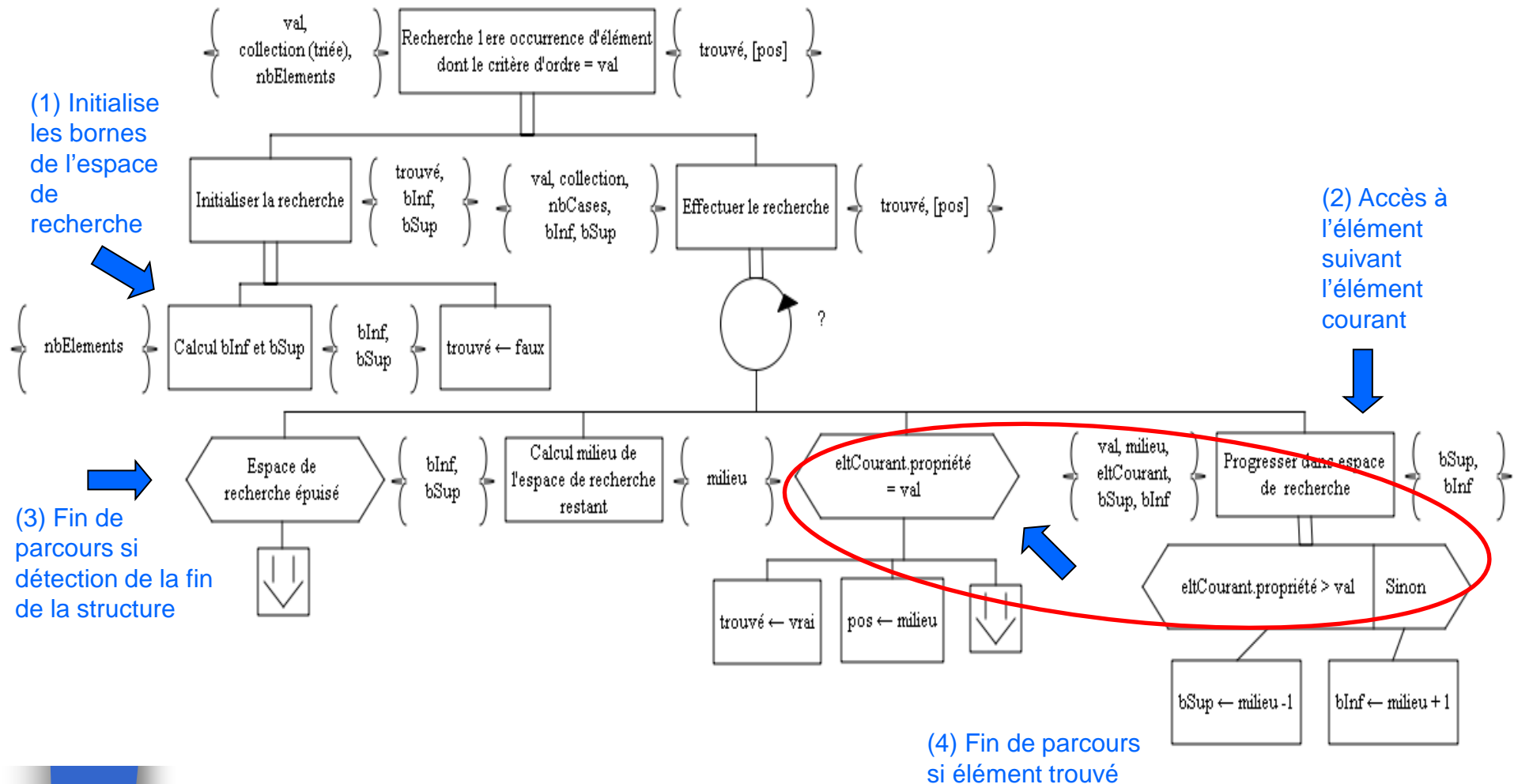
6.- Recherche dichotomique de première occurrence (4/10)

□ Algorithme (version 1)



6.- Recherche dichotomique de première occurrence (4/10)

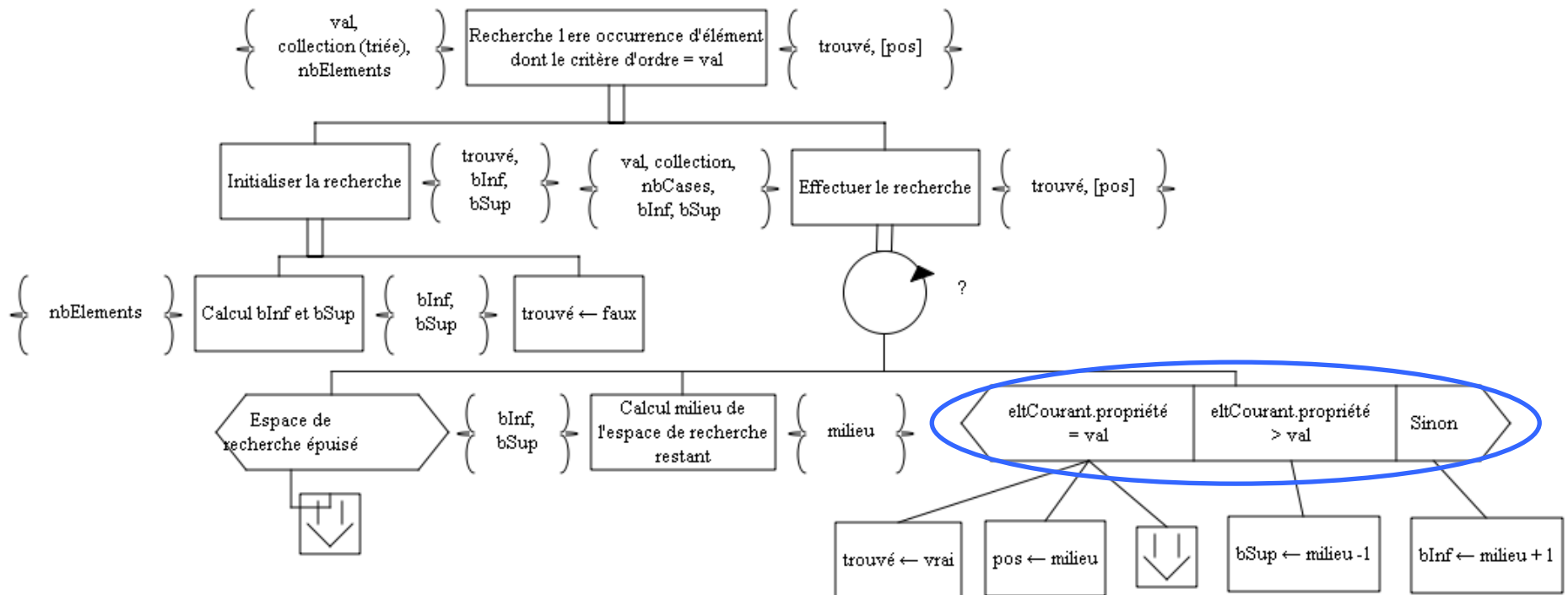
❑ Algorithme (version 1)



6.- Recherche dichotomique de première occurrence (5/10)

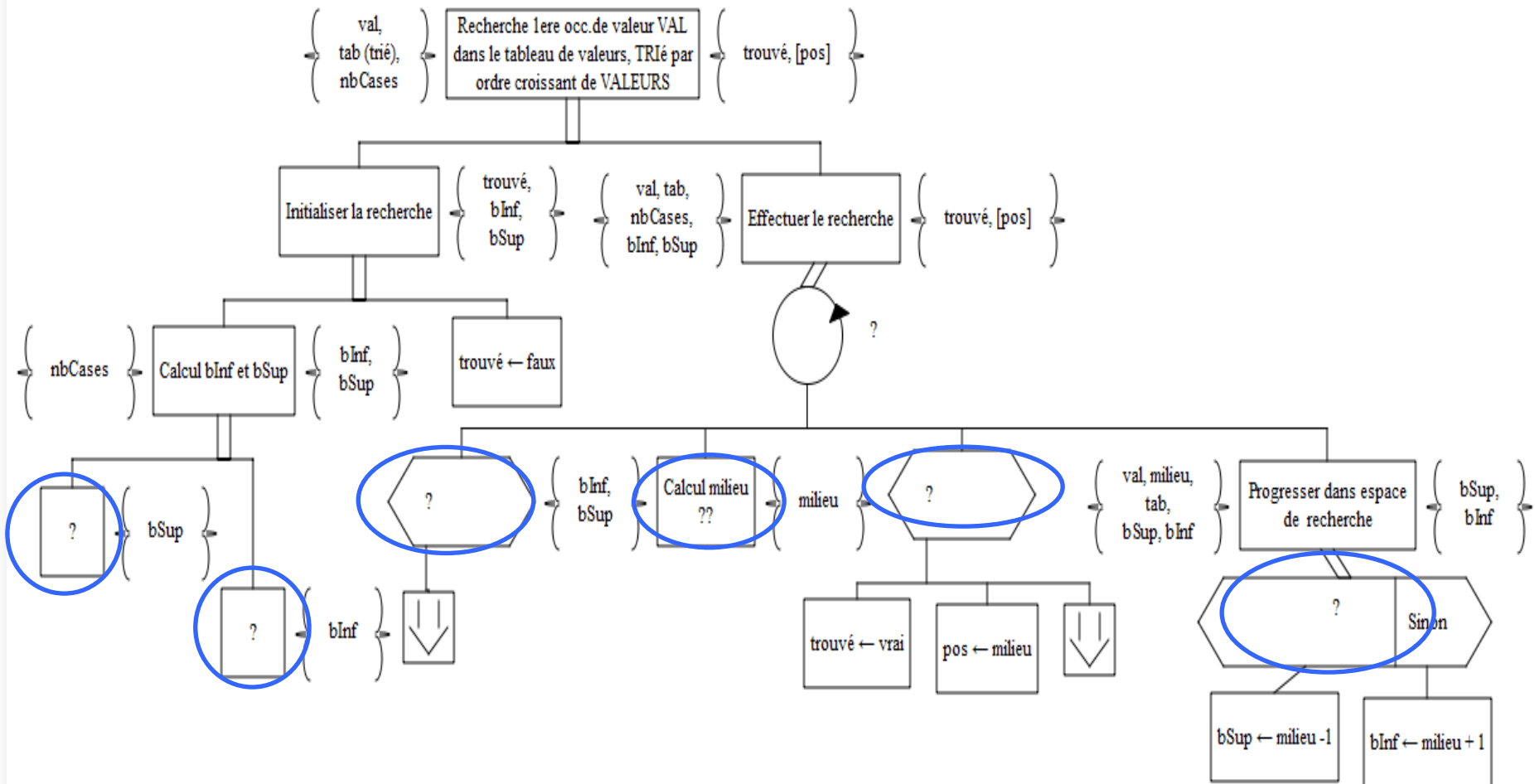


❑ Algorithme (version 2)



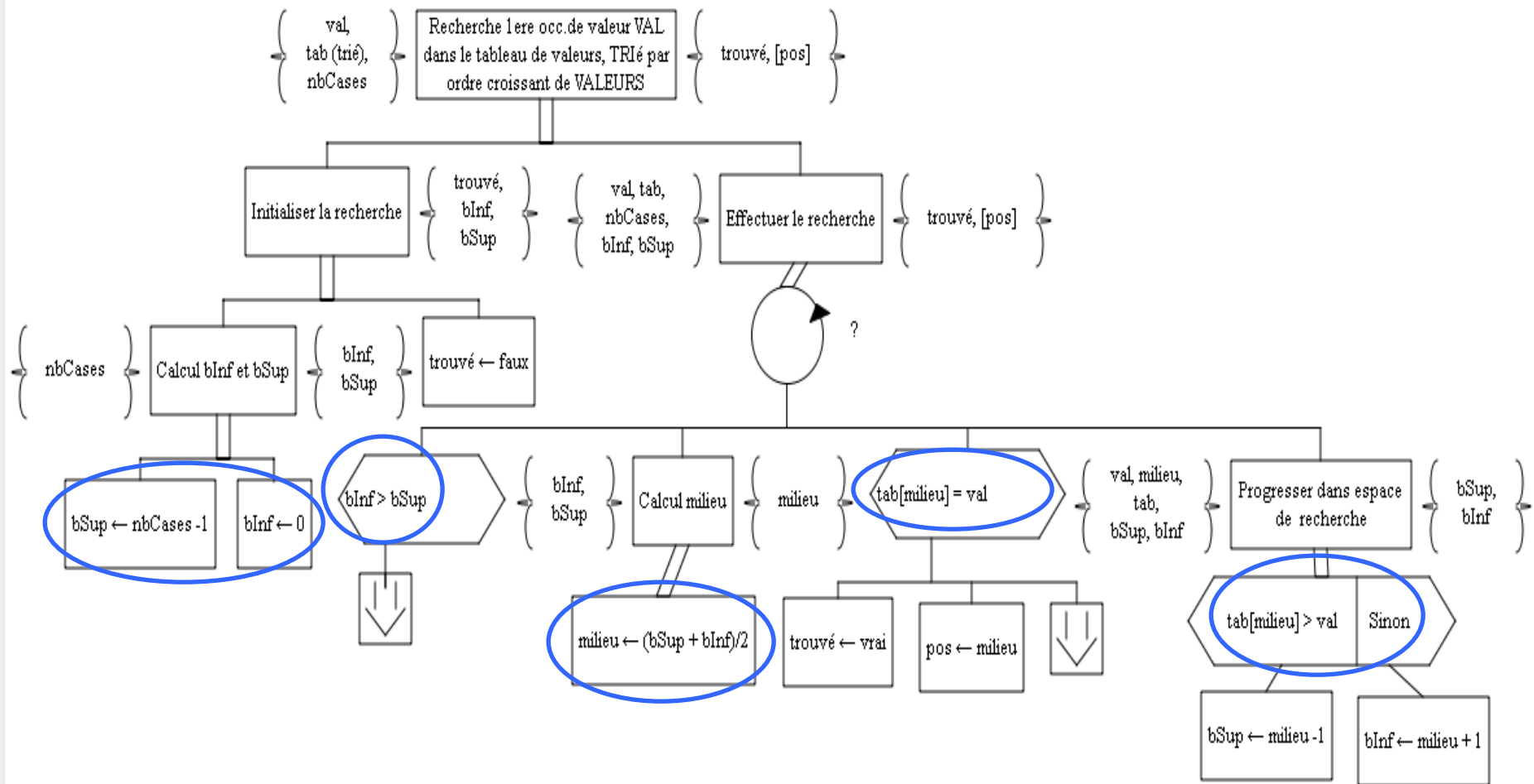
6.- Recherche dichotomique de première occurrence (6/10)

- ❑ Exemple d'application aux tableaux simples : (version 1)
 - Recherche d'une valeur dans un tableau d'entiers **triés par ordre croissant**



6.- Recherche dichotomique de première occurrence (7/10)

- ❑ Exemple d'application aux tableaux simples : (version 1)
 - Recherche d'une valeur dans un tableau d'entiers **triés par ordre croissant**



6.- Recherche dichotomique de première occurrence (8/10)

❑ Bilan :

- Conformément à la spécification d'une Stratégie algorithmique basée sur un modèle de recherche de première occurrence il s'est agi de préciser :
 - ✓ le positionnement en début de collection
 - ✓ la condition permettant de détecter la fin de la collection
 - ✓ le mécanisme permettant de progresser dans la collection
 - ✓ condition permettant de vérifier si l'élément courant a été trouvé
- Cet algorithme est donc une recherche de première occurrence :
 - il a toujours au moins deux conditions d'arrêt de recherche :
 - la recherche a **réussi** : une occurrence de l'élément cherché a été trouvée. Elle peut être unique ou pas.
 - la recherche a **échoué** : on a parcouru toute la structure et l'élément cherché n'a pas été trouvé
 - les variations concernent les modalités de parcours, de positionnement en début et de progression dans la structure (toujours l'élément situé au milieu de l'ensemble des éléments restants à visiter), ainsi que la condition détectant la fin de la collection

❑ Avantage de cette variante

- Plus rapide (cf. calculs de la **complexité**)

6.- Recherche dichotomique de première occurrence – Complexité (9/10)

❑ Complexité dans le pire des cas :

- On peut calculer le nombre maximum d'accès au tableau tab, de taille $n = 100$, **trié par ordre croissant** de valeurs
- Le pire des cas : la valeur cherchée est hors tableau :
Par exemple, on cherche une valeur x telle que : $\text{tab}[41] < x < \text{tab}[42]$

Dans le pire des cas, l'algorithme va diviser l'espace de recherche en deux jusqu'à ce qu'il ne reste qu'une case où chercher.

Le nombre maximum d'itérations m de cet algorithme est :

$$m \leq \log_2(n) + 1 \quad \text{où } n \text{ est la taille du tableau}$$

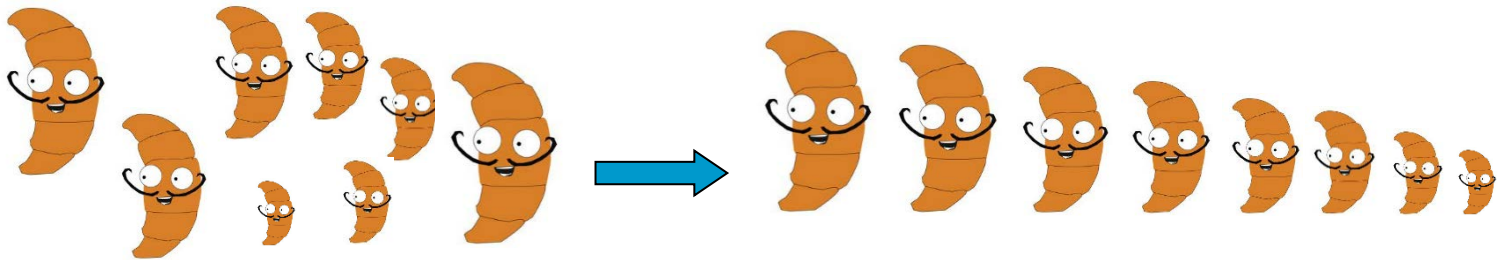
Le nombre d'accès au tableau croît **très faiblement** lorsque la taille du tableau augmente.



7.- Tris de collections homogènes (1/31)

□ Introduction

Il s'agit d'ordonner les éléments d'une collection (un ensemble) selon un critère (une relation d'ordre) lié à leur contenu



C'est une problématique très courante en informatique.

7.- Tris de collections homogènes (2/31)

□ Introduction - La collection à trier peut

- être constituée **d'informations simples**, c'est-à-dire pouvant être comparées entre elles (nombres, chaînes de caractères).

Etant donnée une suite d'informations simples $(a_0, a_1, a_2, \dots, a_{n-1})$,

le tri consiste à les permuter de sorte à obtenir une suite $(a'_0, a'_1, a'_2, \dots, a'_{n-1})$ telle que $a'_0 \leq a'_1 \leq a'_2 \leq \dots \leq a'_{n-1}$

- mais aussi **d'informations complexes**, par exemple, enregistrements $(r_0, r_1, r_2, \dots, r_{n-1})$, où chaque r_i est une information complexe.

Exemple : $r_i = \{ \text{nom, prénom, ville, téléphone} \}$

Dans ce cas, le tri se fera selon un **critère** (= **champ**) de cette information complexe).

Il consistera à permuter les éléments de la collection de sorte à obtenir la suite $(r'_0, r'_1, r'_2, \dots, r'_{n-1})$

telle que : $r'_0.\text{critère} \leq r'_1.\text{critère} \leq r'_2.\text{critère} \leq \dots \leq r'_{n-1}.\text{critère}$

Exemple : Trier les contacts d'un agenda (nom, prénom, codePostal, ville, numTél) :

- *par ordre croissant sur le champ **prénom**; en cas de doublons, le second critère de tri croissant sera le champ nom.*
- *par ordre croissant sur le champ **ville**; en cas de doublons, le second critère sera le nom.*

7.- Tris de collections homogènes (3/31)

❑ Propriétés d'un tri

Elles peuvent décrire les conditions de réalisation du tri, ou encore les propriétés des éléments résultants du tri.

- **par comparaison** : le tri se fait en comparant les éléments entre eux.
- **en place** : le tri n'utilise pas d'espace mémoire supplémentaire et modifie directement les éléments de la collection.
- **stable** : le tri préserve l'ordre initial des éléments en cas d'égalité sur le critère de tri. Important pour le **tri successif** de la collection selon plusieurs critères différents.

Exemple : Liste d'étudiants décrits par (nom, prénom, groupeTP, moyenne générale).

Si l'on applique à la liste d'étudiants 2 tris successifs :

- par rapport à leur nom (ordre lexicographique croissant)
- **puis** par rapport à leur moyenne générale (moyenne décroissante)

Le tri sera dit stable si, pour deux étudiants ayant même moyenne générale (tri2), ils sont présentés selon l'ordre donné par le tri1 (ordre lexicographique croissant sur le nom)

7.- Tris de collections homogènes (4/31)

- ❑ Tris présentés dans le présent chapitre

- Tri par sélection de place
- Tri par insertion de valeur
- Tri bulle

Il s'agit de tris **en place** et **par comparaison**.

- ❑ Pré-conditions

Les collections à trier devront proposer des mécanismes d'accès **direct**.

- ❑ Webographie

- <http://www.sorting-algorithms.com/>
- Une institution hongroise, (mais située en Roumanie), l'université Sapiientia, a produit une série de vidéos montrant des groupes folkloriques d'Europe centrale dansant les divers algorithmes de tri existant en programmation...
<http://www.i-programmer.info/news/150-training-a-education/2255-sorting-algorithms-as-dances.html>

34

7.- Tris de collections homogènes (6/31)

□ Tri par sélection de place (2/13)

– Démonstration

Vue statique complète de l'évolution des éléments au cours de l'exécution de l'algorithme

	ici	pos		0	1	2	3	4	5	6	7	8	9
				7	10	-6	7	-2	9	1	0	8	-6
	9	1		7	10	-6	7	-2	9	1	0	8	-6
	9	1	échange	7	-6	-6	7	-2	9	1	0	8	10
	8	5		7	-6	-6	7	-2	9	1	0	8	10
	8	5	échange	7	-6	-6	7	-2	8	1	0	9	10
	7	5		7	-6	-6	7	-2	8	1	0	9	10
	7	5	échange	7	-6	-6	7	-2	0	1	8	9	10
	6	0		7	-6	-6	7	-2	0	1	8	9	10
	6	0	échange	1	-6	-6	7	-2	0	7	8	9	10
	5	3		1	-6	-6	7	-2	0	7	8	9	10
	5	3	échange	1	-6	-6	0	-2	7	7	8	9	10
	4	0		1	-6	-6	0	-2	7	7	8	9	10
	4	0	échange	-2	-6	-6	0	1	7	7	8	9	10
	3	3	pas d'échange	-2	-6	-6	0	1	7	7	8	9	10
	2	0		-2	-6	-6	0	1	7	7	8	9	10
	2	0	échange	-6	-6	-2	0	1	7	7	8	9	10
	1	1	pas d'échange	-6	-6	-2	0	1	7	7	8	9	10

7.- Tris de collections homogènes (7/31)

□ Tri par sélection de place (3/13)

Principe de résolution - pour le tri **croissant**

La démarche comporte 2 étapes :

– Etape 1 :

- On se place **en cours** de tri, où une partie du tableau est déjà triée, pour tenter d'imaginer l'action à faire.
(démarche proche de la preuve par récurrence en mathématiques)

• Illustration

`tab (borneDeb..borneFin)` est le tableau déjà partiellement trié par ordre *croissant* de valeurs

Il est dans l'état suivant :

- la partie droite, cad la tranche de tableau `tab(ici+1..borneFin)` est déjà triée
- la partie gauche, cad la tranche de tableau `tab(borneDeb..ici)` n'est pas encore triée

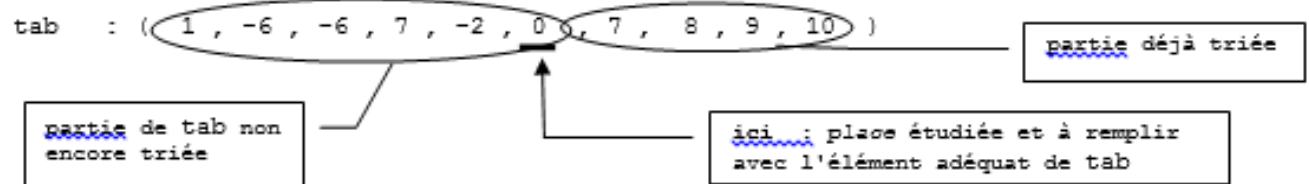
avec `borneDeb` = premier indice valide de `tab` (0 en C++),

et `borneFin` = dernier indice valide de `tab`.

7.- Tris de collections homogènes (8/31)

□ Tri par sélection de place (4/13)

– Etape 1 :



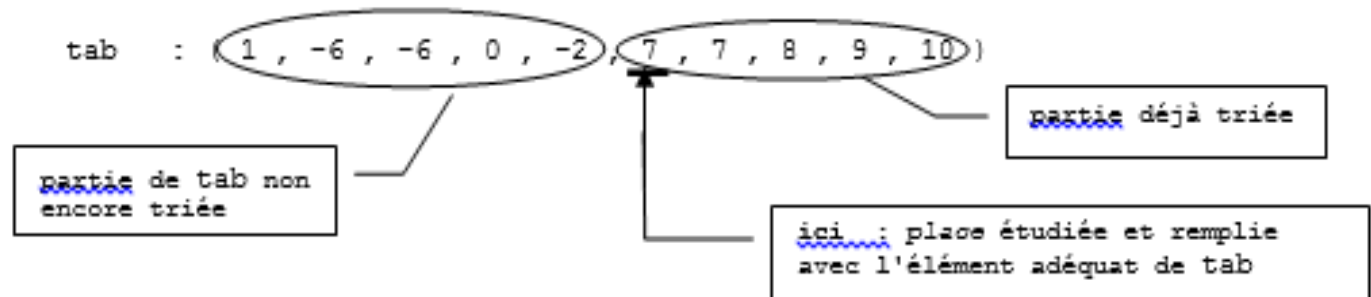
Sous-Problèmes à résoudre dans étape 1 :

- (a) Déterminer l'élément de `tab(borneDeb..ici)` devant occuper la place de position `ici`
- (b) Puis de le placer à sa place, en position `ici`.

Actions résolvant ces problèmes :

- (a) Trouver la position (`pos`) du plus grand élément de cette tranche/partie de tableau
- (b) Echanger `tab(pos)` et `tab(ici)`

Et le résultat obtenu une fois les actions réalisées :



7.- Tris de collections homogènes (9/31)

❑ Tri par sélection de place (5/13)

– Etape 1 : *Remarque*

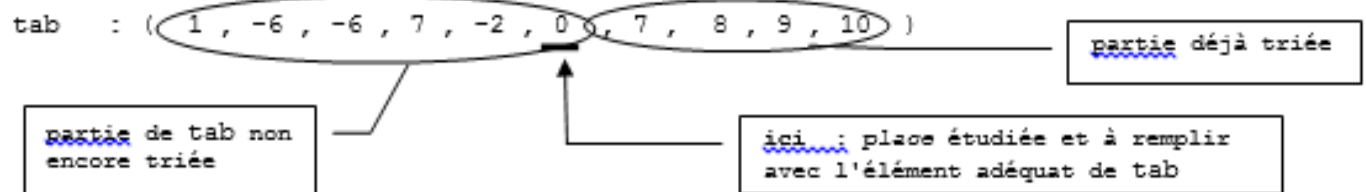
Voici où se trouve cette étape dans le déroulé d'exécution du jeu d'essai présenté

ici	pos	:	0	1	2	3	4	5	6	7	8	9		0	1	2	3	4	5	6	7	8	9
			7	10	-8																		
9	1		7	10	-8																		
9	1	échange	7	-6	-8	5	3							1	-8	-8	7	-2	0	7	8	9	10
8	5		7	-8	-8	5	3							1	-8	-8	0	-2	7	7	8	9	10
8	5	échange	7	-8	-8									1	-8	-8	0	-2	7	7	8	9	10
7	5		7	-8	-8	7	-2	8	1	0	9	10											
7	5	échange	7	-8	-8	7	-2	0	1	8	9	10											
6	0		7	-8	-8	7	-2	0	1	8	9	10											
6	0	échange	1	-8	-8	7	-2	0	7	8	9	10											
5	3		1	-8	-8	7	-2	0	7	8	9	10											
5	3	échange	1	-8	-8	0	-2	7	7	8	9	10											
4	0		1	-8	-8	0	-2	7	7	8	9	10											
4	0	échange	-2	-8	-8	0	1	7	7	8	9	10											
3	3	pas d'échange	-2	-8	-8	0	1	7	7	8	9	10											
2	0		-2	-8	-8	0	1	7	7	8	9	10											
2	0	échange	-6	-8	-2	0	1	7	7	8	9	10											
1	1	pas d'échange	-8	-8	-2	0	1	7	7	8	9	10											

7.- Tris de collections homogènes (10/31)

□ Tri par sélection de place (6/13)

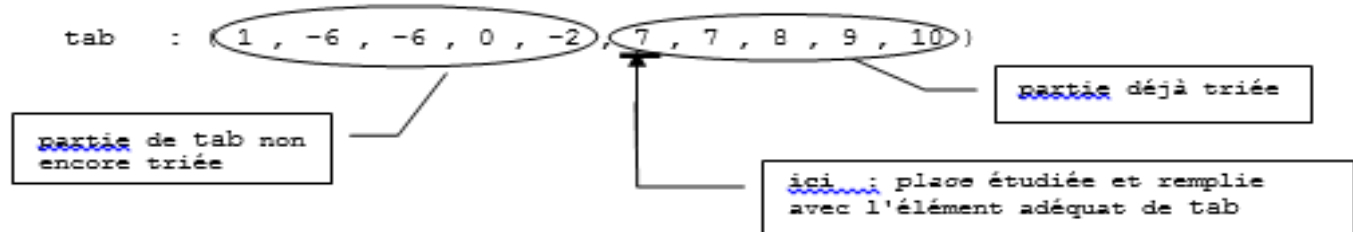
– Etape 1:



Actions à réaliser pour résoudre l'étape 1 :

- (a) Trouver la position (pos) du plus grand élément de cette tranche/partie de tableau
- (b) Echanger $tab(pos)$ et $tab(ici)$

Et le résultat obtenu :



– Etape 2

- Comment réutiliser l'étape 1 pour trier tout le tableau ?

En répétant cette étape pour toutes les valeurs d' ici , par valeur décroissante jusqu'à $ici = 1$...
Justifiez la valeur 1.

7.- Tris de collections homogènes (11/31)

□ Tri par sélection de place (7/13)

– Reste à faire :

- Écrire les déclarations C++ des sous-programmes réalisant les actions (a) et (b) de l'étape 1
- Donner un nom adéquat à l'action réalisée par l'étape 1
- Écrire les algorithmes des actions (a) et (b) de l'étape 1 - réutiliser les modèles d'algorithmes connus !
- Écrire l'algorithme de l'action réalisée dans l'étape 2

7.- Tris de collections homogènes (12/31)

❑ Tri par sélection de place (8/13)

– Faisons-le !

- Écrire les déclarations C++ des sous-programmes réalisant les actions (a) et (b) de l'étape 1

(a)

```
unsigned int positionDuMax (const int tab [],  
                           unsigned int bDeb,  
                           unsigned int bFin)  
  
//retourne l'indice de la plus grande valeur du tableau tab  
//comprise entre les bornes bDeb et bFin
```

(b)

```
void echanger (int& x, int& y)  
// échange le contenu de deux entiers x et y
```

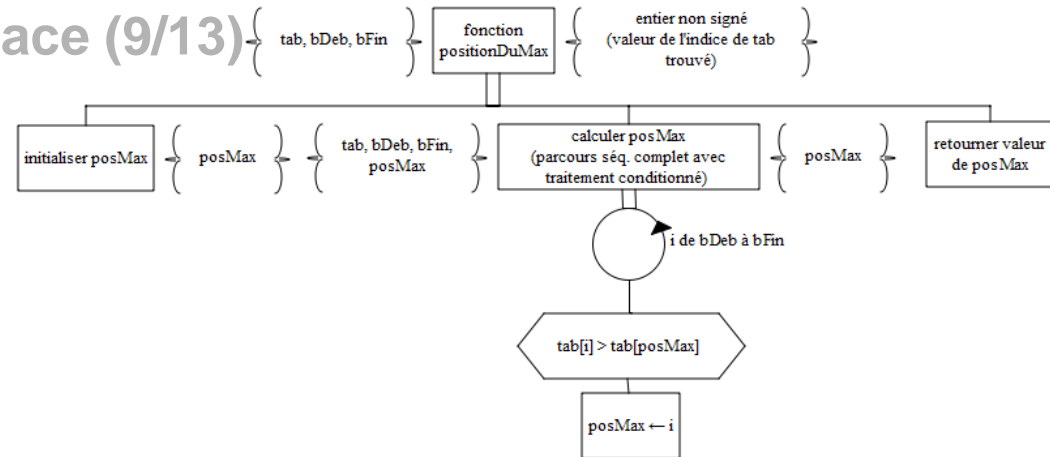
- Donner un nom adéquat à l'action réalisée par l'étape 1
Placer le plus grand élément de la tranche `tab(borneDeb..ici)` en position `ici`
- Écrire l'algorithme de l'action (a) de l'étape 1 - réutiliser les modèles d'algorithmes !
Cf diapositive suivante
- Écrire l'algorithme de l'action étape 2 puis celui du tri par sélection de place complet
Cf diapositive suivante

7.- Tris de collections homogènes (13/31)

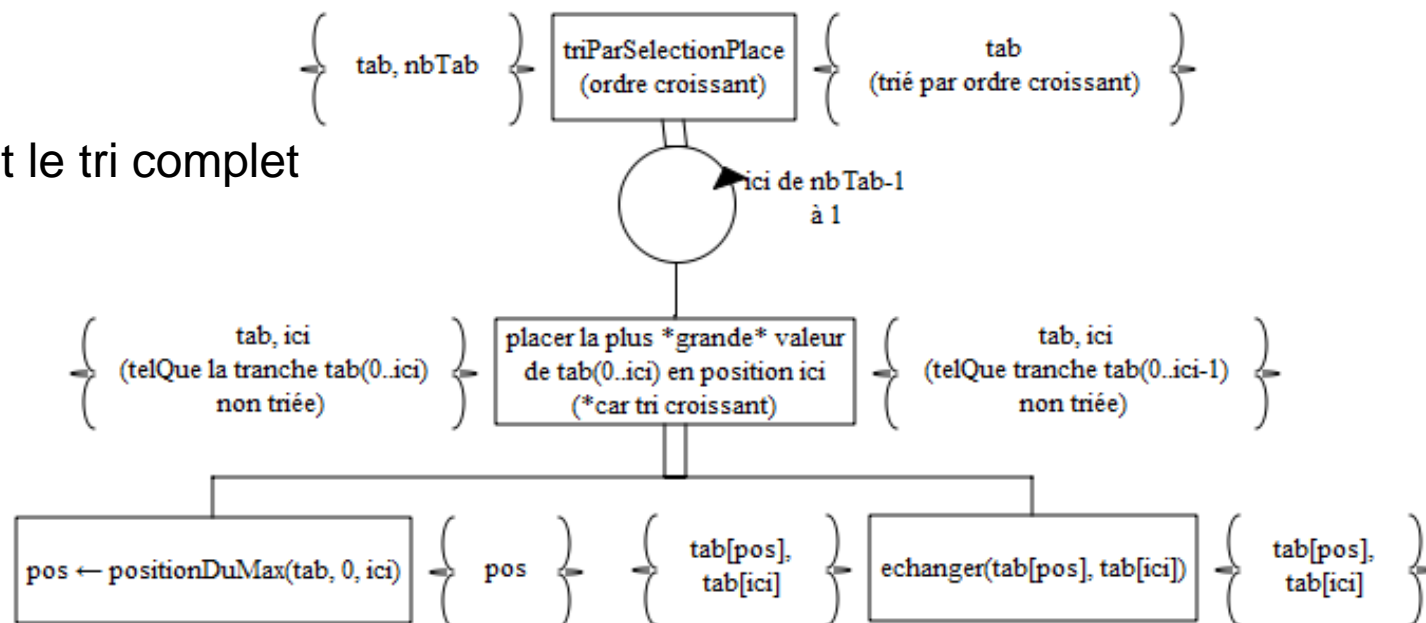


Tri par sélection de place (9/13)

– algorithme (a) :



– ... et le tri complet



7.- Tris de collections homogènes (14/31)

□ Tri par sélection de place (10/13)

- Complexité du tri **ParSélectionDePlace** pour un tableau de n éléments :

*Le **nombre total d'accès** à un élément du tableau est indépendant du contenu du tableau :*

$$n^2 + 3n - 4$$

*c'est-à-dire variant comme **n^2***

7.- Tris de collections homogènes (18/31)

□ Tri de la bulle (1/14)

- Exemple d'application : un tableau d'entiers – tri **croissant**

```
void triBulle (int tab [], unsigned int nbTab);  
// tri des nbTab (>0) éléments de tab (avec d'éventuels doublons)  
// par ordre croissant de valeur par la méthode de tri de même nom
```

- Principe

- On choisit un sens de parcours, par exemple de gauche à droite.
- À partir du premier indice de gauche, comparer les éléments du tableau deux à deux, les échanger si celui de gauche est plus grand que celui de droite.
Durant cette action, cet élément (le plus à gauche) sera comparé avec son premier voisin de droite, et déplacé vers la droite par échanges successifs, jusqu'à ce qu'il rencontre un élément plus grand que lui. L'élément plus grand poursuit alors le même mouvement, et ainsi de suite jusqu'à ce que la dernière place du tableau soit occupée.
- On recommence avec le (nouveau) élément du tableau situé le plus à gauche, jusqu'à ce que l'avant-dernière place soit occupée.
- Et ceci pour toutes les (nouveaux) éléments situés le plus à gauche, jusqu'à ce que le tableau soit complètement trié.

7.- Tris de collections homogènes (19/31)

□ Tri de la bulle (2/14)

– Démonstration

Etape n° 1

- Premier parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110
7	3
6	31
5	70
4	95
3	27
2	5
1	82
0	10

Le plus grand élément du tableau à trier est placé

7.- Tris de collections homogènes (20/31)

□ Tri de la bulle (3/14)

– Démonstration

Etape n° 2

- Second parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110	8	110	8	110	8	110	8	110	8	110	8	110	8	110	8	110	8	110
7	3	7	3	7	3	7	3	7	3	7	3	7	3	7	3	7	3	7	95
6	31	6	31	6	31	6	31	6	31	6	31	6	31	6	95	6	3	6	3
5	70	5	70	5	70	5	70	5	70	5	70	5	95	5	31	5	31	5	31
4	95	4	95	4	95	4	95	4	95	4	95	4	95	4	70	4	70	4	70
3	27	3	27	3	27	3	27	3	82	3	82	3	82	3	82	3	82	3	82
2	5	2	5	2	5	2	82	2	27	2	27	2	27	2	27	2	27	2	27
1	82	1	82	1	82	1	5	1	5	1	5	1	5	1	5	1	5	1	5
0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10	0	10

7.- Tris de collections homogènes (21/31)

□ Tri de la bulle (4/14)

– Démonstration

Etape n° 3

- Troisième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110	8	110	8	110	8	110	8	110	8	110	8	110	8	110	8	110
7	95	7	95	7	95	7	95	7	95	7	95	7	95	7	95	7	95
6	3	6	3	6	3	6	3	6	3	6	3	6	3	6	3	6	82
5	31	5	31	5	31	5	31	5	31	5	82	5	3	5	3	5	3
4	70	4	70	4	70	4	70	4	82	4	31	4	31	4	31	4	31
3	82	3	82	3	82	3	82	3	82	3	70	3	70	3	70	3	70
2	27	2	27	2	27	2	27	2	27	2	27	2	27	2	27	2	27
1	5	1	5	1	10	1	10	1	10	1	10	1	10	1	10	1	10
0	10	0	10	0	5	0	5	0	5	0	5	0	5	0	5	0	5

7.- Tris de collections homogènes (22/31)

□ Tri de la bulle (5/14)

– Démonstration

Etape n° 4

- Quatrième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110	8	110	8	110	8	110	8	110	8	110	8	110
7	95	7	95	7	95	7	95	7	95	7	95	7	95
6	82	6	82	6	82	6	82	6	82	6	82	6	82
5	3	5	3	5	3	5	3	5	3	5	3	5	70
4	31	4	31	4	31	4	31	4	31	4	70	4	3
3	70	3	70	3	70	3	70	3	70	3	31	3	31
2	27	2	27	2	27	2	27	2	27	2	27	2	27
1	10	1	10	1	10	1	10	1	10	1	10	1	10
0	5	0	5	0	5	0	5	0	5	0	5	0	5

The diagram illustrates the fourth pass of the bubble sort algorithm. It shows a sequence of seven states of a 14-element array. The elements are: 110, 95, 82, 3, 31, 70, 27, 10, 5, 70, 31, 27, 10, 5. The first three elements (110, 95, 82) are already sorted and remain in their positions. The last element (5) is the smallest and remains at the bottom. The elements 3, 31, 70, 27, 10, and 70 are the ones being compared and potentially swapped. In each state, the largest element in the unsorted portion (the 'bubble') is circled in red. Blue arrows indicate the swaps between adjacent elements. The unsorted portion is indicated by a bracket on the left side of the array.

7.- Tris de collections homogènes (23/31)

□ Tri de la bulle (6/14)

– Démonstration

Etape n° 5

- Cinquième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110	8	110	8	110	8	110	8	110	8	110
7	95	7	95	7	95	7	95	7	95	7	95
6	82	6	82	6	82	6	82	6	82	6	82
5	70	5	70	5	70	5	70	5	70	5	70
4	3	4	3	4	3	4	3	4	3	4	31
3	31	3	31	3	31	3	31	3	31	3	3
2	27	2	27	2	27	2	27	2	27	2	27
1	10	1	10	1	10	1	10	1	10	1	10
0	5	0	5	0	5	0	5	0	5	0	5

The diagram illustrates the 5th pass of the bubble sort algorithm. It shows a sequence of 6 states of a 9-element array. The elements are 110, 95, 82, 70, 3, 31, 27, 10, and 5. The first four elements (110, 95, 82, 70) are already sorted and are shown in green. The last four elements (3, 31, 27, 10, 5) are the 'portion de tableau à trier' (portion of the array to be sorted). In each state, the largest element in the unsorted portion is circled in red and an arrow indicates its movement to the end of the array. In the final state, the array is fully sorted: 110, 95, 82, 70, 31, 27, 10, 5.

7.- Tris de collections homogènes (24/31)

□ Tri de la bulle (7/14)

– Démonstration

Etape n° 6

- Sixième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone

portion de tableau à trier

8	110	8	110	8	110	8	110	8	110
7	95	7	95	7	95	7	95	7	95
6	82	6	82	6	82	6	82	6	82
5	70	5	70	5	70	5	70	5	70
4	31	4	31	4	31	4	31	4	31
3	3	3	3	3	3	3	3	3	27
2	27	2	27	2	27	2	27	2	3
1	10	1	10	1	10	1	10	1	10
0	5	0	5	0	5	0	5	0	5

Diagram illustrating the 6th step of the bubble sort algorithm. The array is shown in a grid with indices 0 to 8. The elements are: 110, 95, 82, 70, 31, 3, 27, 10, 5. The current comparison is between index 0 (5) and index 1 (10). The element 10 is circled in red, and an arrow points to it from the element 5 at index 0. The element 27 at index 2 is also circled in red, and an arrow points to it from the element 10 at index 1. The label 'portion de tableau à trier' is on the left, with a bracket indicating the range from index 0 to 8.

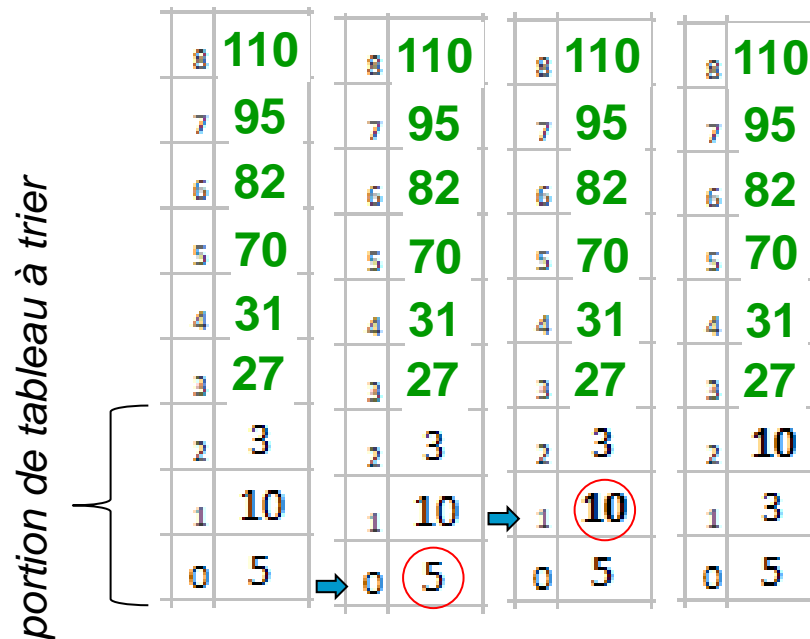
7.- Tris de collections homogènes (25/31)

□ Tri de la bulle (8/14)

– Démonstration

Etape n° 7

- Septième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone



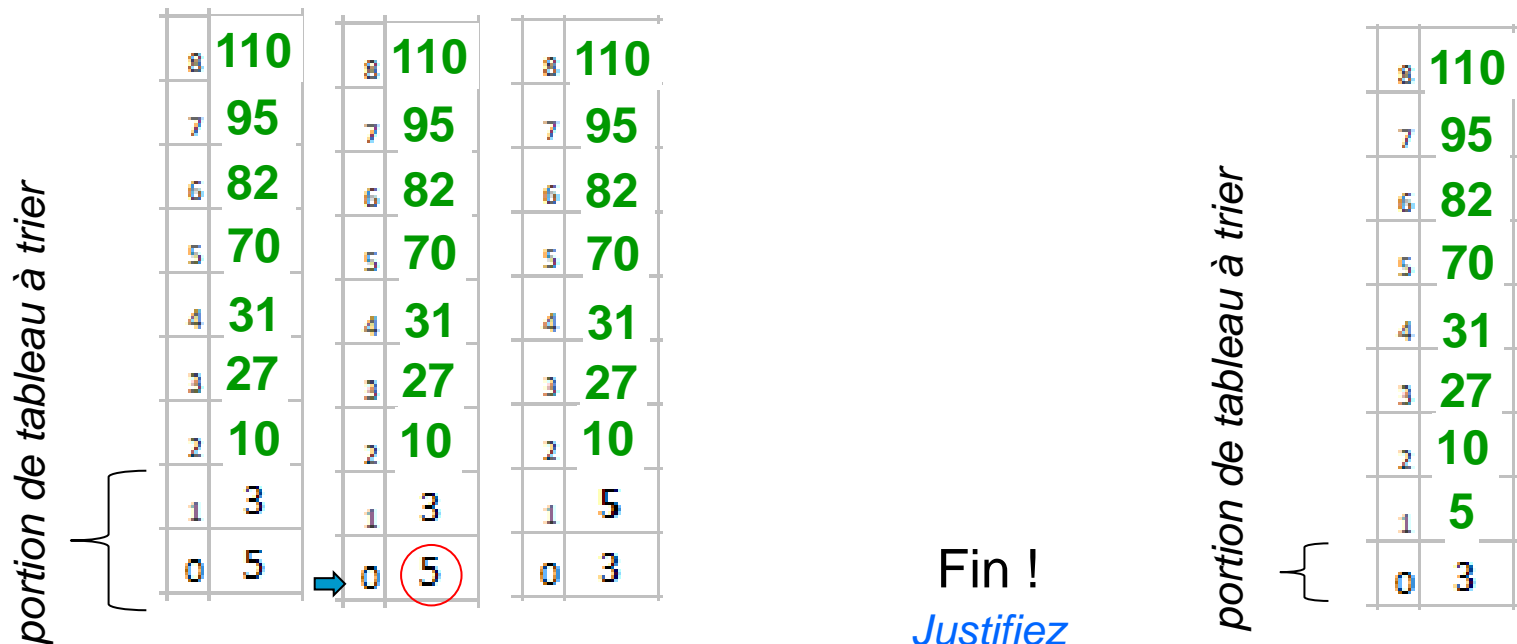
7.- Tris de collections homogènes (26/31)

□ Tri de la bulle (9/14)

– Démonstration

Etape n° 8

- Huitième parcours complet de la *portion de tableau à trier*, en comparant les éléments deux à deux ($\text{tab}[i]$ et $\text{tab}[i+1]$).
- On fera 'remonter' le plus grand élément de cette zone



7.- Tris de collections homogènes (27/31)

□ Tri de la bulle (10/14)

Principe de résolution - pour le tri **croissant**

La démarche comporte 2 étapes :

– Etape 1 :

- On se place **en cours** de tri, où une partie du tableau est déjà triée, pour tenter d'imaginer l'action à faire.

• Illustration

`tab (borneDeb..borneFin)` est le tableau déjà partiellement trié par ordre *croissant* de valeurs

Il est dans l'état suivant :

- la partie droite, cad la tranche de tableau `tab(ici+1..borneFin)` est déjà triée
- la partie gauche, cad la tranche de tableau `tab(borneDeb..ici)` n'est pas encore triée

0	1	2	3	4	5	6	7	8
10	5	27	82	70	31	3	95	110

ici

avec `borneDeb` = premier indice valide de `tab` (0 en C++),

et `borneFin` = dernier indice valide de `tab`.

7.- Tris de collections homogènes (28/31)

□ Tri de la bulle (11/14)

– Etape 1:

0	1	2	3	4	5	6	7	8
10	5	27	82	70	31	3	95	110

ici

Sous-Problème à résoudre dans étape 1 :

Dans la partie non triée,

faire remonter la bulle (le plus grand élément) jusqu'à la position *ici*,
par échanges 2 à 2

à partir de l'élément le plus à gauche

Et le résultat obtenu :

0	1	2	3	4	5	6	7	8
5	10	27	70	31	3	82	95	110

ici

- La partie triée a grandi
- La partie non triée s'est réduite

– Etape 2

- Comment réutiliser l'étape 1 pour trier tout le tableau ?

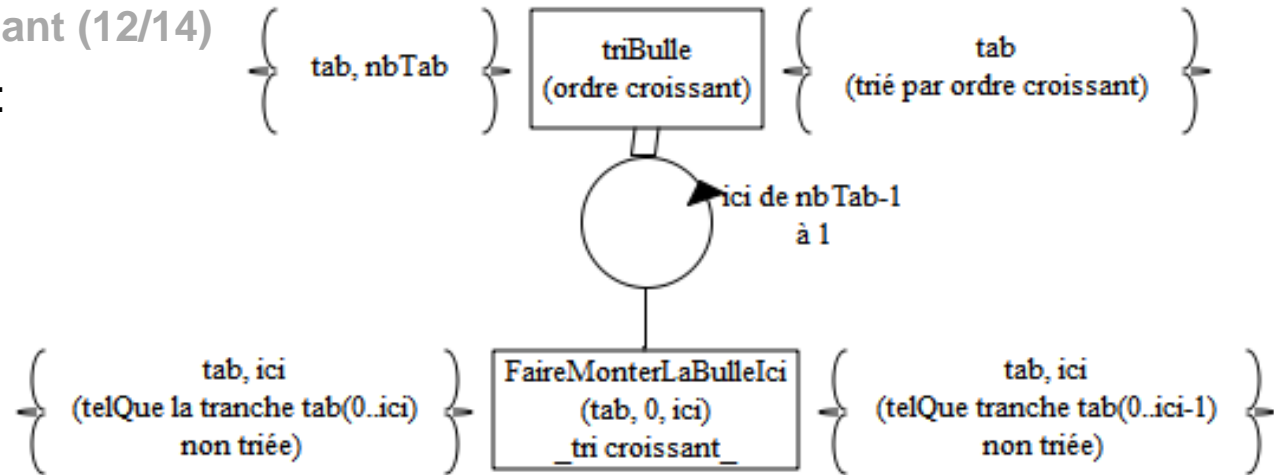
En répétant cette étape pour toutes les valeurs d'*ici*, par valeur décroissante jusqu'à *ici* = 1 ...
Justifiez la valeur 1.

7.- Tris de collections homogènes (29/31)

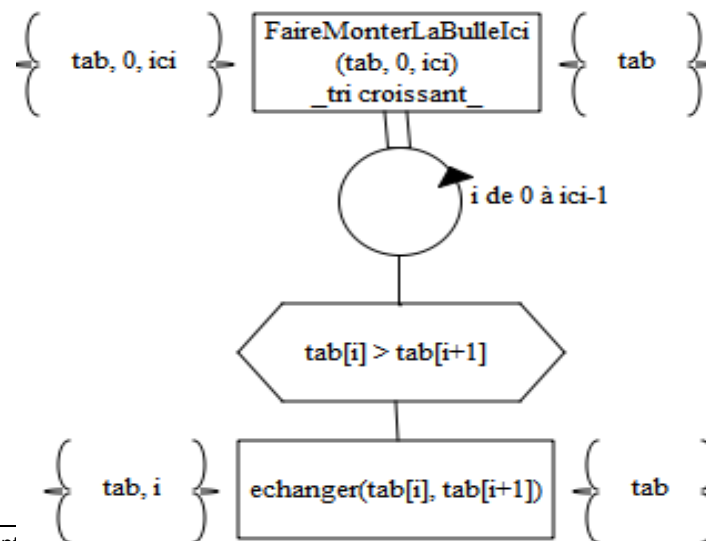


□ Tri bulle croissant (12/14)

– Tri complet :



– ... avec :



7.- Tris de collections homogènes (30/31)

❑ Tri de la bulle croissant (13/14)

- Complexité de l'algorithme **triBulle** pour un tableau de n éléments :

Le nombre total d'accès à un élément du table **tab** n'est pas fixe, il dépend du contenu du tableau :

- au max $6*n*(n-1)/2$
- au min $2*n*(n-1)/2$,
- en moyenne $2*n*(n-1)$ accès au tableau

c'est-à-dire variant comme n^2

8.- Parcours parallèle de deux collections ordonnées (1/8)

- ❑ Nom du modèle : **parcours séquentiel parallèle de 2 collections ordonnées**
- ❑ Quand l'utiliser :
 - Lorsque l'on cherche :
 - à interclasser (= fusionner) les 2 collections (→ **tri fusion**)
 - ou bien à appliquer un même traitement à tous les éléments, en respectant l'ordre inter-collection des éléments (= l'ordre des éléments entre les 2 collections)
- ❑ Exemple d'interclassement :
 - À partir des sous-listes d'étudiants des groupes TP1 et TP2,
triées chacune par ordre alphabétique croissant selon le nom,
produire une unique liste des étudiants de ces 2 groupes,
également triée par ordre alphabétique croissant selon le nom.
- ❑ Pré-conditions :
 - Les éléments des deux collections doivent être **ordonnés** selon le même critère.
 - L'ordre peut être croissant ou décroissant
 - L'ordre peut ne pas être strict (il peut y avoir des doublons)
 - Les collections doivent proposer un mécanisme d'accès **séquentiel** et/ou **direct**

8.- Parcours parallèle de deux collections ordonnées (2/8)

❑ Illustration de la fusion / interclassement

Données :

deux tableaux (nom, prénom, groupe de TP)
triés par ordre croissant sur le **NOM**

	Nom	Prénom	TP
1	AUBRY	Loïc	1
2	BACONNIER	Anaïs	1
3	BOISSE	Lucas	1
4	DAUBA	Roxanne	1
5	FAGET	Corentin	1
6	ZINANE	Ayoub	1

×

	Nom	Prénom	TP
1	BOSCALS	Tanguy	2
2	CHIGNAGUET	Adrien	2
3	FLEURY	Arnaud	2
4	LE ROY	Gabriel	2
5	LEVELU	Pierre	2

×

Résultat fusion :

	Nom	Prénom	TP
1	AUBRY	Loïc	1
2	BACONNIER	Anaïs	1
3	BOISSE	Lucas	1
4	BOSCALS	Tanguy	2
5	CHIGNAGUET	Adrien	2
6	DAUBA	Roxanne	1
7	FAGET	Corentin	1
8	FLEURY	Arnaud	2
9	LE ROY	Gabriel	2
10	LEVELU	Pierre	2
11	ZINANE	Ayoub	1



❑ À remarquer :

Les 2 collections ont été parcourues **simultanément**, on dit aussi **de manière parallèle**

8.- Parcours parallèle de deux collections ordonnées (3/8)

❑ Stratégie

- L'algorithme est conforme au modèle de **parcours séquentiel complet** , **et donc**
 - 1 seule condition de fin de parcours (composée de sous-conditions) :
*On a atteint (la fin de Collection1) **ET** (la fin de Collection2)*
- D'autres conditions de fin de parcours peuvent se rajouter
Exemple : Fusionner les 5 premiers éléments de chaque collection
- Action à faire (et à répéter) à l'intérieur du parcours :
 - Cela dépend du travail demandé ...

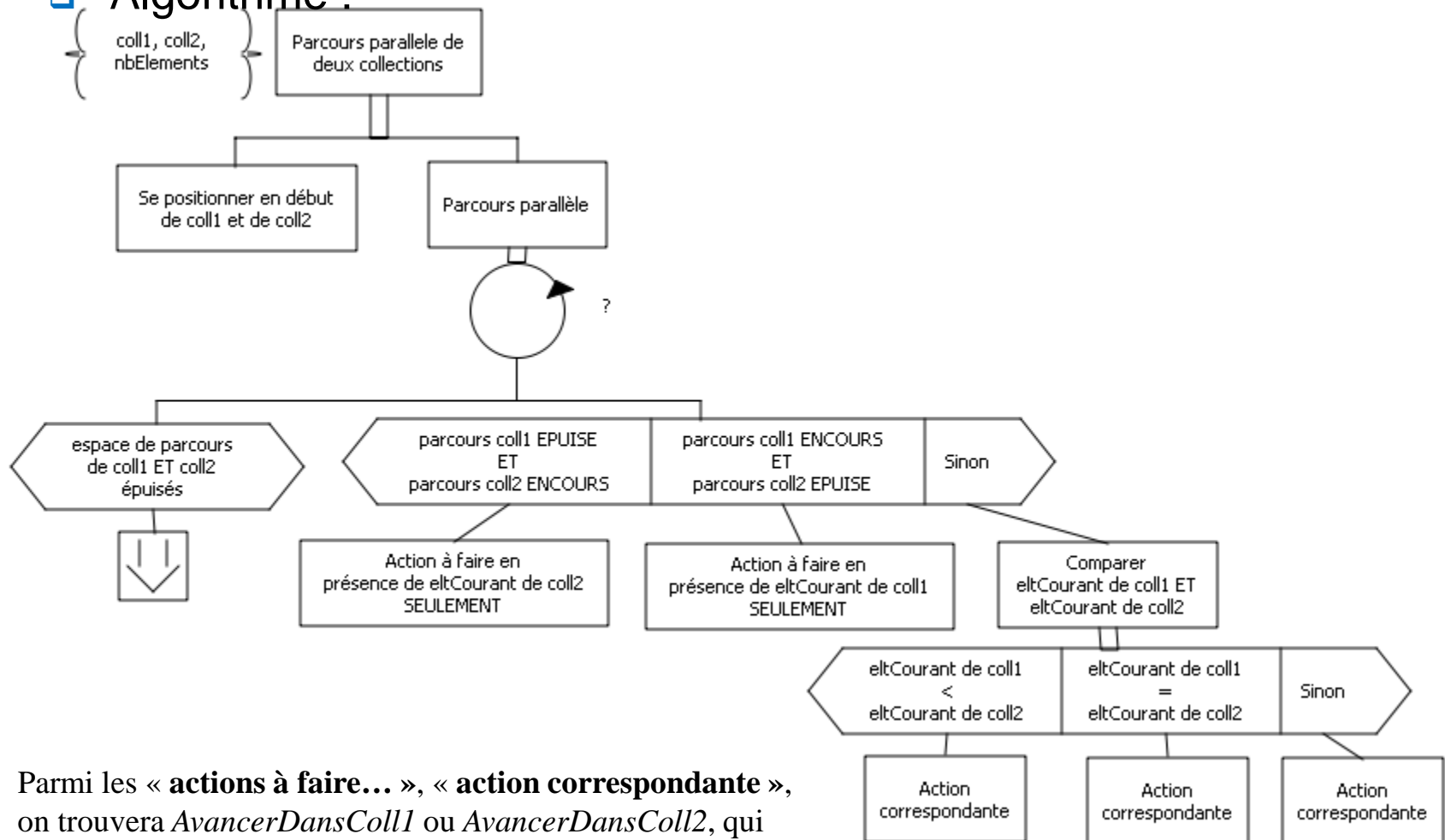
❑ Remarque

- La forme de l'algorithmique (1 parcours séquentiel complet) resterait valable s'il y avait 3 ou plus de collections à fusionner
- il n'y aurait toujours qu'une même seule condition de fin :
*On a atteint (la fin de Collection1) **ET** (la fin de Collection2) **ET** (la fin de Collection3) ... **ET** (la fin de CollectionN)*
- MAIS ce n'est pas cette solution qui est choisie pour fusionner N collections...

8.- Parcours parallèle de deux collections ordonnées (4/8)



□ Algorithme :



Parmi les « **actions à faire...** », « **action correspondante** », on trouvera *AvancerDansColl1* ou *AvancerDansColl2*, qui permettent d'accéder à l'élément suivant dans chacune des collections

8.- Parcours parallèle de deux collections ordonnées (5/8)

□ Exemple d'application aux tableaux simples :

- Soient tabP et tabQ deux tableaux d'entiers **triés par ordre croissant**

	0	1	2	3						
tabP	-44	-20	-2	1						
	0	1	2	3	4	5	6			
tabQ	-60	-20	-2	-2	0	40	55			

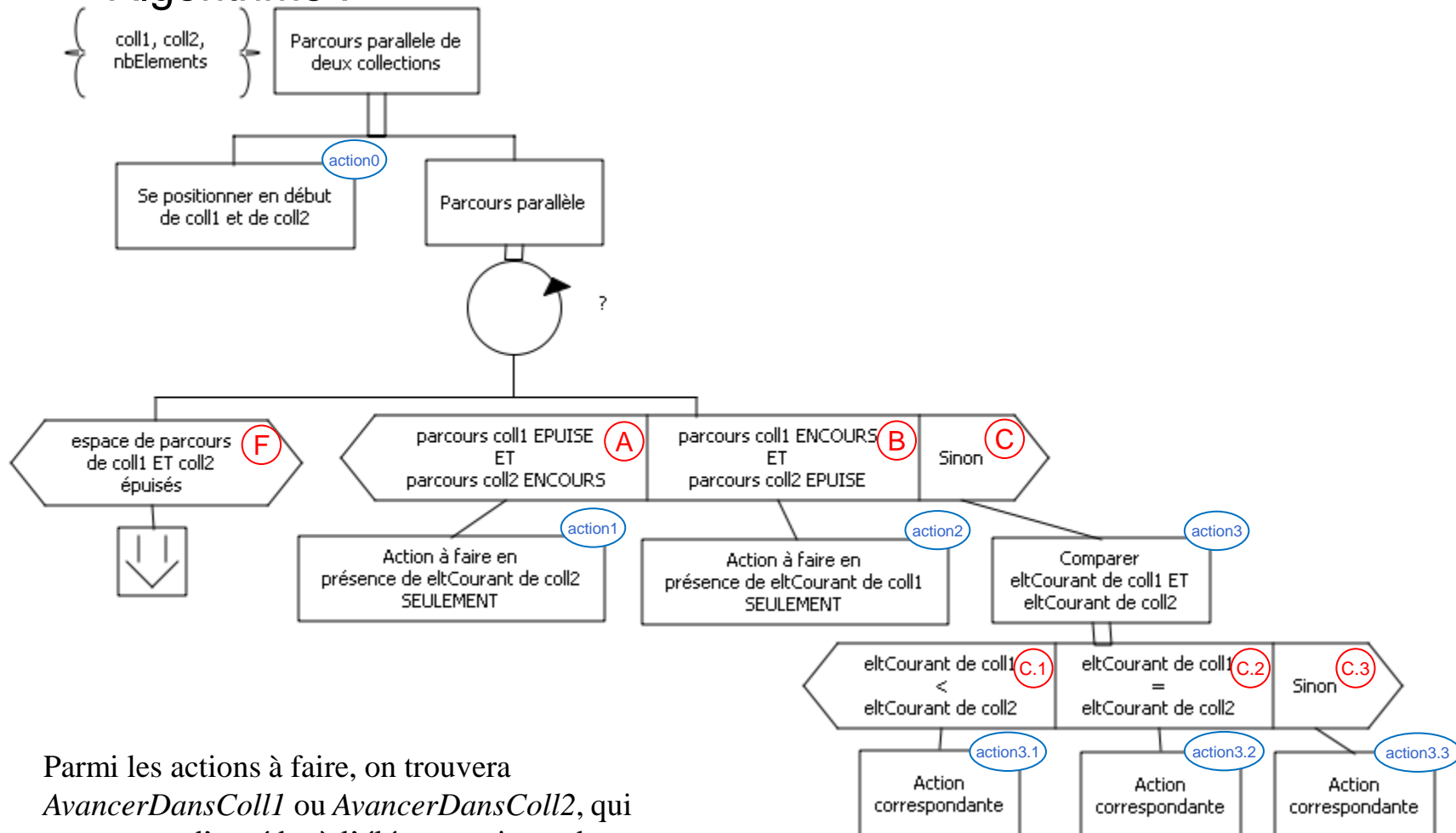
- nbCasesP et nbCasesQ représentent le nombre d'éléments de chaque tableau
- Ecrire l'algorithme permettant d'afficher à l'écran les valeurs de tabP **et** tabQ par ordre croissant de valeur, doublons compris.

Affichage attendu : -60 -44 -20 -20 -2 -2 -2 0 1 40 55

- Stratégie
 - modèle à appliquer : **parcours séquentiel complet parallèle des 2 tableaux**, en précisant :
 - sens de parcours :
de Gauche à Droite (indices croissants)
 - positionnement en début de structure :
 $iP \leftarrow 0 ; iQ \leftarrow 0$
 - condition de fin de parcours :
 $(iP > nbCasesP - 1)$ **ET** $(iQ > nbCasesQ - 1)$
 - accès à l'élément suivant (actions *avancerDansCollection*):
 $iP++$, $iQ++$
 - traitement à appliquer à chaque élément :
Afficher(tabP[iP]), Afficher(tabQ[iQ]), actions conditionnées

8.- Parcours parallèle de deux collections ordonnées (6/8)

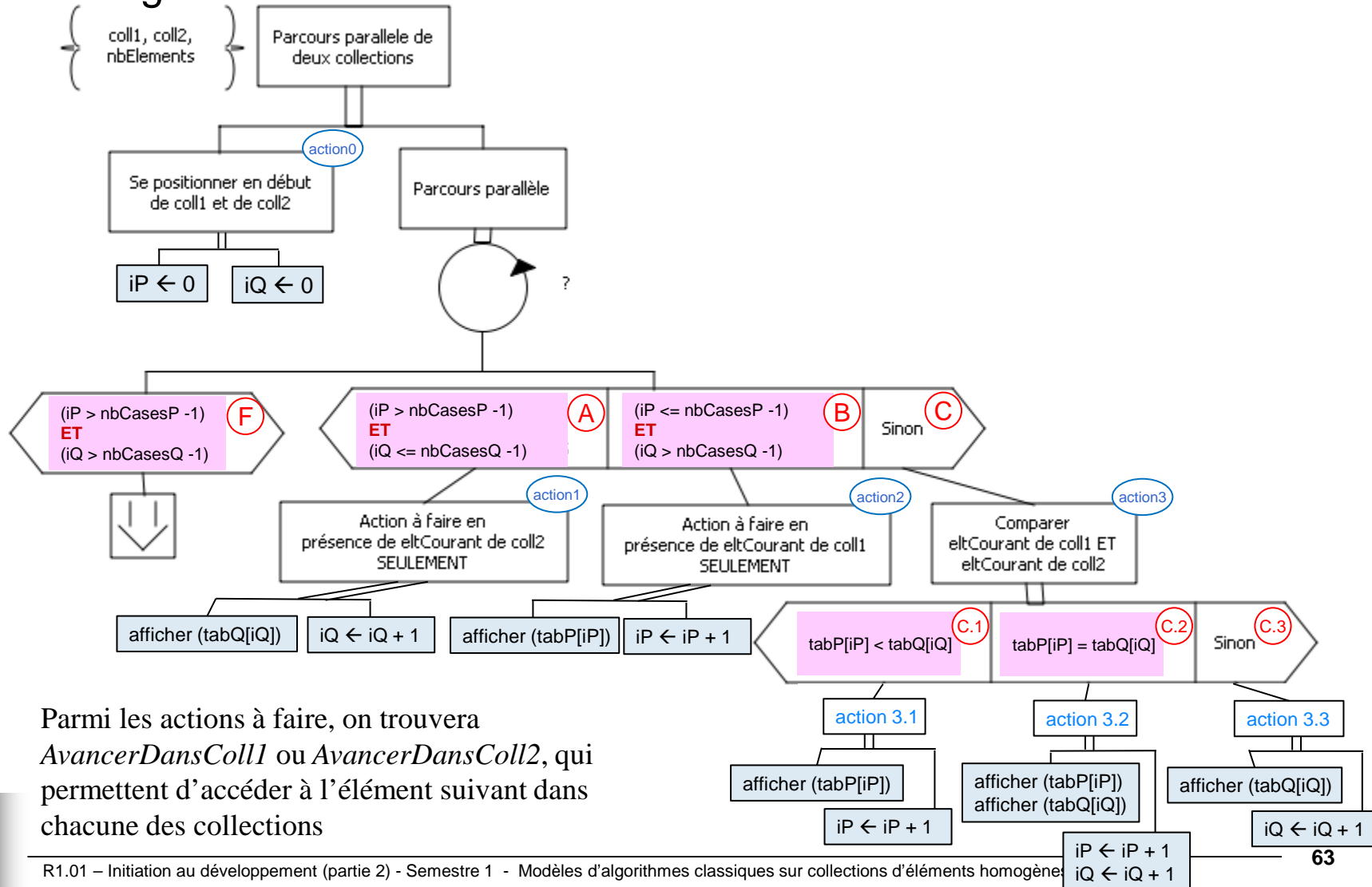
□ Algorithme :



Parmi les actions à faire, on trouvera *AvancerDansColl1* ou *AvancerDansColl2*, qui permettent d'accéder à l'élément suivant dans chacune des collections

8.- Parcours parallèle de deux collections ordonnées (7/8)

□ Algorithmme :



- Exemple d'application aux tableaux simples :
 - Tableau suivi exécution

R1.01 – Initiation au développement (partie 2) - Semestre 1 - Modèles d'algorithmes classiques sur collections d'éléments homogènes (suite et fin)

8.- Parcours parallèle de deux collections ordonnées (8/8)

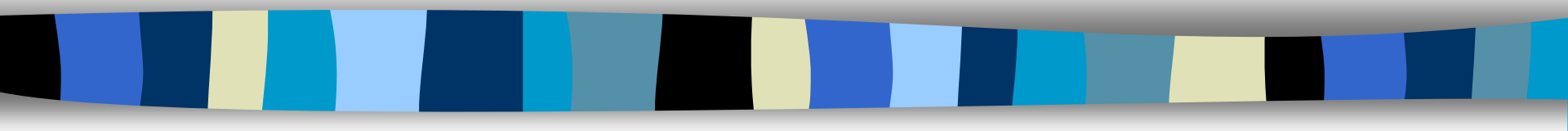
- Exemple d'application aux tableaux simples :
 - Tableau suivi exécution

ligne	instruction / action	iP	iQ	tabP								affichage écran
				0	1	2	3					
				-44	-20	-2	1					
				0	1	2	3	4	5	6		
				tabQ	-60	-20	-2	-2	0	40	55	
1	action 0	0	0									
2	F : tabP ET tabQ épuisés ?											
3	non --> cas C.3 --> action 3.3		1									-60
4	F : tabP ET tabQ épuisés ?											
5	non --> cas C.1 --> action 3.1	1										-44
6	F : tabP ET tabQ épuisés ?											
7	non --> cas C.2 --> action 3.2	2	2									-20 -20
8	F : tabP ET tabQ épuisés ?											
9	non --> cas C.2 --> action 3.2	3	3									-2 -2
10	F : tabP ET tabQ épuisés ?											
11	non --> cas C.3 --> action 3.3		4									-2
12	F : tabP ET tabQ épuisés ?											
13	non --> cas C.3 --> action 3.3		5									0
14	F : tabP ET tabQ épuisés ?											
15	non --> cas C.1 --> action 3.1	4										1
16	F : tabP ET tabQ épuisés ?											
17	non --> cas A --> action 1		6									40
18	F : tabP ET tabQ épuisés ?											
19	non --> cas A --> action 1		7									55
20	F : tabP ET tabQ épuisés ?											
	oui !											

chapitre 3.-

Modèles d'algorithmes sur collections d'éléments homogènes

à connaître par cœur



Ressource R1.01 : Initiation au développement - Partie 2

Merci pour votre attention !