

# R1.01

## INITIATION AU DÉVELOPPEMENT

---

### Cours 6, partie 3 : Vecteurs triés

✓ Recherche dichotomique

Hervé Blanchon & Anne Lejeune


Université Grenoble Alpes

IUT 2 – Département Informatique

# Sommaire

---

 Idée directrice de la recherche dichotomique

 Traiter  $v[m]$  par l'exemple

 Vers l'algorithme complet

 Différentes traces

# **IDÉE DIRECTRICE DE L'ALGORITHME SUR UN VECTEUR D'ENTIERS TRIÉ**

# Préambule

---

 La recherche dichotomique peut être implémenter de deux manières

 séquentielle (comme tous les algorithmes vus jusque ici)

 récursive (comme nous le verrons plus tard)

 Entête de la fonction séquentielle

```
private static
int indiceValDichoIterative(ArrayList<Integer> v, int val)
// {v trié croissant non vide} =>
// {résultat = indice le plus à gauche de val si val est dans v ;
//           -indice que val devrait occuper si val n'est pas dans v}
```

# Utiliser au mieux le fait que le vecteur est trié

Idée :

- disposer d'une zone de recherche utile, bornée par deux indices ( $\text{inf}$  et  $\text{sup}$ ), dans laquelle il est intéressant de chercher  $\text{val}$

Situation intermédiaire

- on peut proposer le dessin suivant qui permet d'obtenir l'invariant
  - la zone de recherche utile est l'intervalle  $[\text{inf} .. \text{sup}-1]$



Note :

- La littérature sur la recherche dichotomie propose d'autres dessins ou invariants, nous avons choisi celui-ci pour des raisons pédagogiques

# Invariant de la recherche dichotomique




$$v[0 .. \text{inf}-1] < \text{val} \leq v[\text{sup} .. v.\text{size}()-1]$$

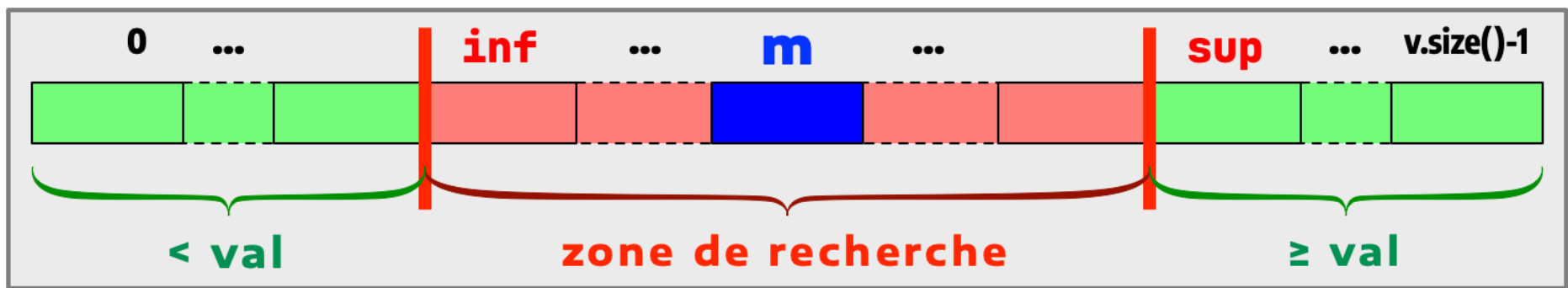
invariant

# Utiliser au mieux le fait que le vecteur est trié

## Traitement dans la zone de recherche

 idée (géniale !) consulter la valeur de  $v[m]$  avec  $m$  l'**indice milieu** de l'intervalle  $[inf .. sup-1]$

 
$$m = \frac{inf + sup}{2} \quad (c'est\ une\ division\ entière)$$



 traiter  $v[m]$  ?

 mettre à jour  $inf$  ou  $sup$


# TRAITER $V[M]$ PAR L'EXEMPLE



# Traiter v[m] ? exemples

 Soit v le vecteur suivant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 On va chercher l'indice le plus à gauche de 10 et de 8 avec les appels suivants :

```
int indice10 = indiceValDichoIterative(vectTrieInteger, 10);  
int indice8 = indiceValDichoIterative(vectTrieInteger, 8);
```

 On aura :

 indice10 = 9

 indice8 = 7

# Recherche indice le plus à gauche de 10

Une situation intermédiaire de l'algorithme sera :

0	1	2	3	4	5	<b>inf</b>		7	8	9	10	<b>sup</b>		12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

L'algorithme a détecté que

$v[0 .. 5] < 10 \rightarrow \text{inf} = 6$ , et

$v[11 .. 23] \geq 10 \rightarrow \text{sup} = 11$

la zone de recherche est donc l'intervalle  $[6 .. 10]$  ( $[\text{inf} .. \text{sup}-1]$ )

L'**indice milieu** de la zone de recherche est :

$$m = \frac{6+11}{2} = \frac{17}{2} = 8$$

# Recherche indice le plus à gauche de 10

étape i

0	1	2	3	4	5	inf		7	8	9	10	sup		11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85			

Traiter  $v[m]$  ? ( $m = 8$ )

$v[m] = 9 < 10$  (val)

10 ne peut pas se trouver dans l'intervalle  $[0 .. 8]$

inf va prendre la valeur de  $m+1$  (9)

inf =  $m + 1$ ;

retour à l'itération sur l'intervalle  $[9 .. 10]$

étape i+1

0	1	2	3	4	5	6	7	8	inf		sup		11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

# Recherche indice le plus à gauche de 8

Une situation intermédiaire de l'algorithme sera :

0	1	2	3	4	5	<b>inf</b>		7	<b>8</b>	9	10	<b>sup</b>		11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

L'algorithme a détecté que

$v[0 .. 5] < 8 \rightarrow \text{inf} = 6$ , et

$v[11 .. 23] \geq 8 \rightarrow \text{sup} = 11$

la zone de recherche est donc l'intervalle  $[6 .. 10]$  ( $[\text{inf} .. \text{sup}-1]$ )

L'**indice milieu** de la zone de recherche est :

$$m = \frac{6+11}{2} = \frac{17}{2} = 8$$

# Recherche indice le plus à gauche de 8

étape i

0	1	2	3	4	5	inf					sup		11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

Traiter  $v[m]$  ? ( $m = 8$ )

$v[m] = 9 \geq 8$  (val)

8 ne peut pas se trouver dans l'intervalle  $[8 .. 10]$

sup va prendre la valeur de  $m$  (8)

$sup = m$ ;

retour à l'itération sur l'intervalle  $[6 .. 7]$

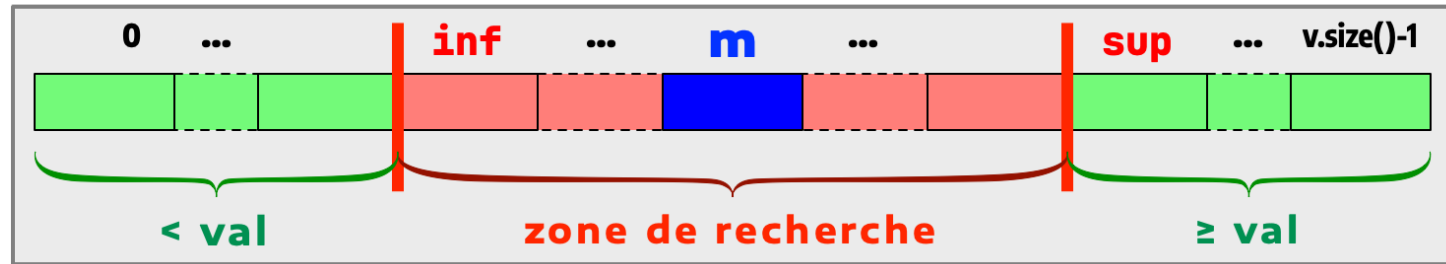
étape i+1

0	1	2	3	4	5	inf		sup		11	12	13	14	15	16	17	18	19	20	21	22	23	
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

# VERS L'ALGORITHME COMPLET

# Traiter $v[m]$ — récapitulons

Soit la situation intermédiaire :

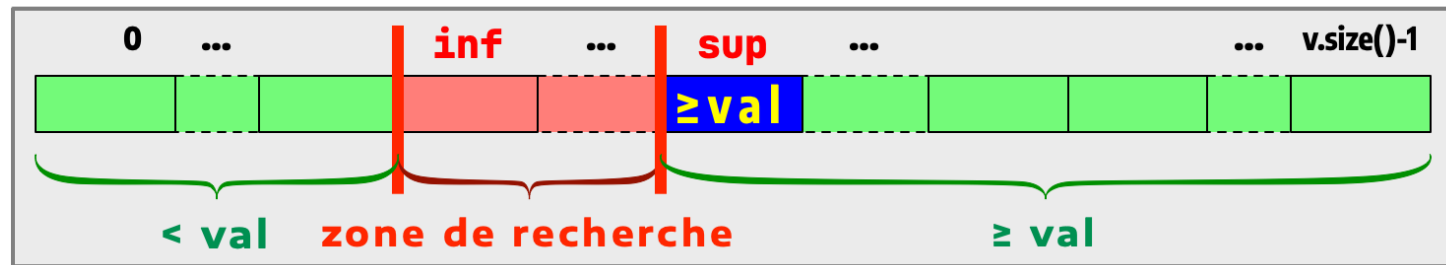


Si  $v[m] \geq \text{val}$  alors

la nouvelle zone de recherche est sur l'intervalle  $[inf .. m-1]$

poursuivre la recherche à gauche de  $m$

$\text{sup} = m;$

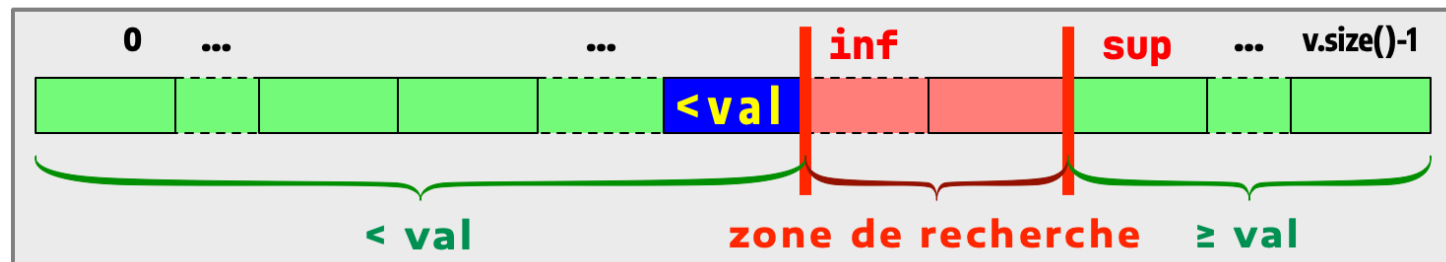


Sinon //  $v[m] < \text{val}$

la nouvelle zone de recherche est sur l'intervalle  $[m+1 .. sup]$

poursuivre la recherche à droite de  $m+1$

$\text{inf} = m + 1;$

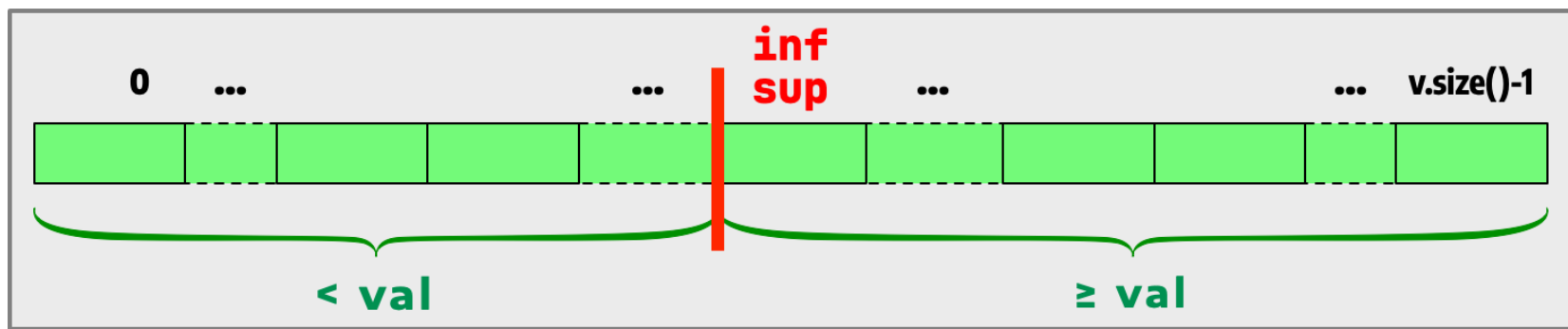


# Remarques et situation finale

## Remarques

- Les traitements successifs réduisent obligatoirement la zone de recherche  $[\text{inf} .. \text{sup}-1]$  en faisant...
  - ...soit croître **inf**
  - ...soit décroître **sup**
- inf** et **sup** vont donc forcément devenir égaux dans la situation finale (*zone de recherche vide*)

## Situation finale





# Itération

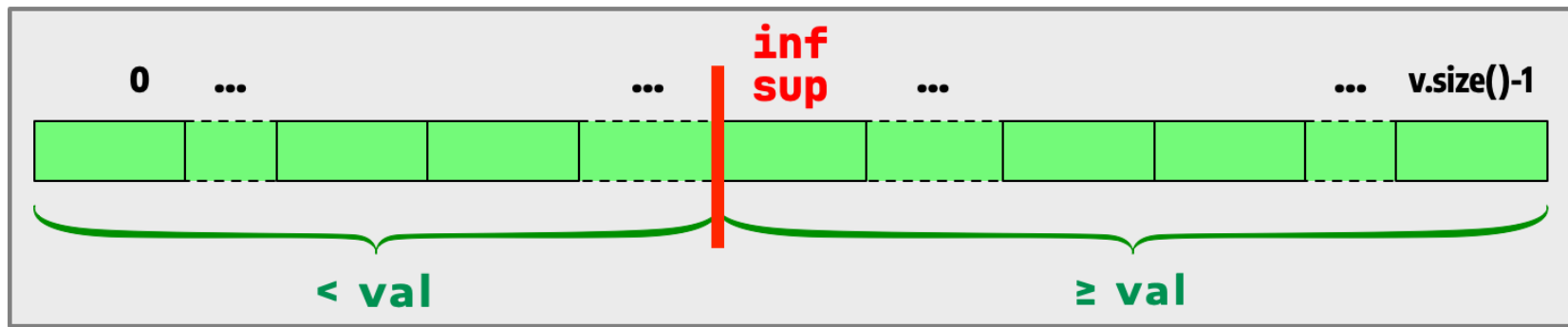
---

 Avec la situation finale et le traitement de  $v[m]$ , on obtient :

```
while (inf < sup) {  
    m = (inf + sup) / 2;  
    if (v.get(m) >= val) { // v[i] ≥ val  
        sup = m; // continuer de chercher à gauche sur [inf..m-1]  
    } else { // v[i] < val  
        inf = m + 1; // continuer de chercher à droite sur [m+1..sup-1]  
    }  
}
```

# Production du résultat

## Situation finale



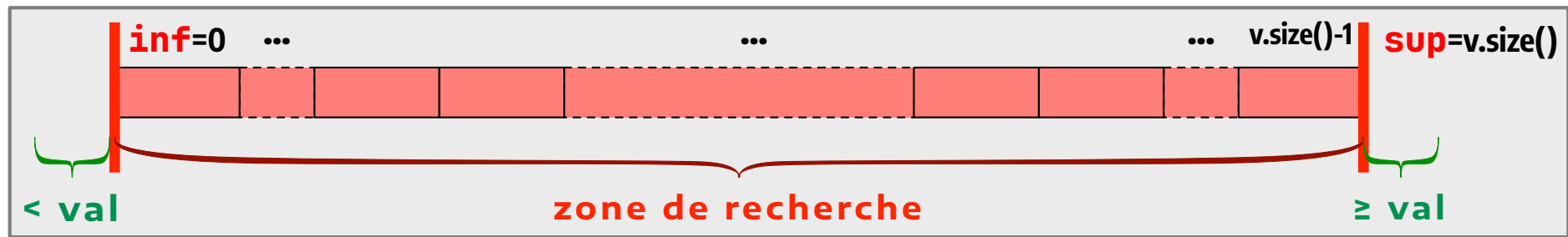
## Production du résultat

 il suffit de regarder la valeur de `v[sup]` (`v.get(sup)`)

```
if (v.get(sup) == val) {  
    return sup; // val trouvée  
} else {  
    return -sup; // val pas trouvée, val aurait été à l'indice sup  
}
```

# Initialisation — le mauvais choix

🧩 Dessin en réfléchissant vite :



🧩 on peut avoir envie de dire que tout le vecteur est la zone de recherche

🧩 Initialisation issue du dessin ci-dessus :

```
int inf = 0;  
int sup = v.size();
```

🧩 ça ne va pas marcher !

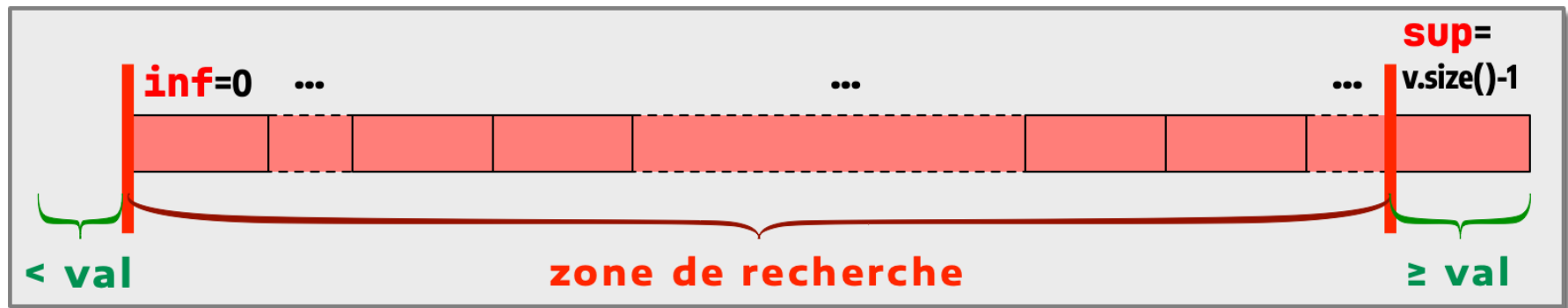
🧩 si `val > v[v.size()-1]` on aura `inf = sup = v.size()` à la sortie de l'itération

🧩 il est interdit de consulter `v[v.size()]`

🧩 il faut gérer le cas où `v[v.size()-1]` est strictement supérieur à `val`

# Initialisation — le bon choix

🧩 Dessin qui marche :



🧩 il va falloir gérer à part le cas où  $v[v.size()-1] < val$

🧩 Initialisation correcte :

```
if (v.get(v.size()-1) < val ) { // v.[v.size()-1] < val
    return -v.size(); // val devrait occuper l'indice v.size()
} else { // val ≥ v.[v.size()-1]
    int inf = 0;
    int sup = v.size()-1;
    // invariant vérifié
    // vers itération et production du résultat
}
```

```
private static int indiceValDichoIterative(ArrayList<Integer> v, int val) {
    // {v trié croissant non vide} =>
    // {résultat = indice le plus à gauche de val si val est dans v ;
    //           -indice que val devrait occuper si val n'est pas dans v}
    if (v.get(v.size()-1) < val) { // v.[v.size()-1] < val
        return -v.size();
    } else { // v.[v.size()-1] ≥ val
        int inf = 0;
        int sup = v.size()-1; // invariant vérifié
        int m;
        while (inf < sup) {
            m = (inf + sup) / 2;
            // invariant vérifié
            if (v.get(m) >= val) { // v[m] ≥ val
                sup = m; // continuer de chercher à gauche sur [inf..m-1]
            } else { // v[m] < val
                inf = m + 1; // continuer de chercher à droite sur [m+1..sup-1]
            }
            // invariant vérifié
        }
        // inf = sup
        if (v.get(sup) == val) {
            return sup; // val trouvée
        } else {
            return -sup; // val pas trouvée, val aurait été à l'indice sup
        }
    }
}
```

# Classe d'utilisation

## Classe

```
import java.util.ArrayList;
import java.util.Arrays;

public class RechercheDichotomique {
    private static int indiceValDichoIterative(ArrayList<Integer> v, int val) {
        // {v trié croissant non vide} =>
        // {résultat = indice le plus à gauche de val si val est dans v ;
        //          -indice que val devrait occuper si val n'est pas dans v}
        -> voir planche précédente
    }

    public static void main(String[] args) {
        // déclaration et initialisation à partir d'une liste d'entiers
        ArrayList<Integer> v = new ArrayList<>(Arrays.asList(1, ..., 85)); // voir trace
        System.out.println("Vecteur dans lequel on cherche : " + v);
        System.out.println("indice de 90 ? : " + indiceValDichoIterative(v, 90));
        System.out.println("indice de 10 ? : " + indiceValDichoIterative(v, 10));
        System.out.println("indice de 8 ? : " + indiceValDichoIterative(v, 8));
        System.out.println("indice de 16 ? : " + indiceValDichoIterative(v, 16));
    }
}
```

## Trace

```
Vecteur dans lequel on cherche : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 11,
12, 13, 14, 15, 20, 28, 37, 45, 63, 78, 85]
indice de 90 ? : -24
indice de 10 ? : 9
indice de 8 ? : 7
indice de 16 ? : -17
```

# **DIFFÉRENTES TRACES DE RECHERCHE**

# Situation

---

 Dans le vecteur  $v$  suivant :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 On propose la trace de la recherche de l'indice le plus à gauche de 90, 10, 8 et 16




**Indice le plus à gauche de 90**

# Trace

---

 Le vecteur `v` contient 24 valeurs indicées de 0 à 23

 `v.size()-1 = 23`

 `90 > v[v.size()-1]`


```
if (v.get(v.size()-1) < val) { // v.[v.size()-1] < val
    return -v.size();
} ...
```

 La fonction retournera : -24

**Indice le plus à gauche de 10**

# Trace – indice à gauche de 10

 Le vecteur  $v$  contient 24 valeurs indicées de 0 à 23

  $v.size()-1 = 23$


  $10 \leq v[v.size()-1]$

```
} else {                                     // v.[v.size()-1] ≥ val
    int inf = 0;
    int sup = v.size()-1;                     // invariant vérifié
    int m;
    while (inf < sup) {
        ...
    }
    // inf = sup
    ...
}
```


 La fonction retournera : 9

# Détail – indice à gauche de 10 (1/6)

 initialisation


  $\text{inf} = 0, \text{sup} = 23$

 condition d'itération vraie, **itération 1**

  $m = 11, v[m] = 10 \geq 10$  (val)


inf																							sup
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération


  $\text{sup} = 11$  (m)

# Détail – indice à gauche de 10 (2/6)

 valeurs des variables d'itération


  $\text{inf} = 0, \text{sup} = 11$

 condition d'itération vraie, **itération 2**

  $m = 5, v[m] = 6 < 10$  (val)


inf											sup												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération


  $\text{inf} = 6 (m + 1)$

# Détail – indice à gauche de 10 (3/6)

 valeurs des variables d'itération


  $\text{inf} = 6, \text{sup} = 11$

 condition d'itération vraie, **itération 3**

  $m = 8, v[m] = 9 < 10$  (val)


							<b>inf</b>				<b>sup</b>												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération


  $\text{inf} = 9 (m + 1)$

# Détail – indice à gauche de 10 (4/6)

 valeurs des variables d'itération


  $\text{inf} = 9, \text{sup} = 11$

 condition d'itération vraie, **itération 4**

  $m = 10, v[m] = 10 \geq 10$  (val)

									inf		sup												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85


 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 10$  (m)




# Détail – indice à gauche de 10 (5/6)

 valeurs des variables d'itération

  $\text{inf} = 9, \text{sup} = 10$

 condition d'itération vraie, **itération 5**

  $m = 9, v[m] = 10 \geq 10$  (val)


									inf														
										sup													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 9$  (m)

# Détail – indice à gauche de 10 (6/6)


 valeurs des variables d'itération

  $\text{inf} = 9, \text{sup} = 9$

 condition d'itération fausse

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85


 retour du résultat

  $v[\text{sup}] = 10 \text{ (val)} \rightarrow \text{return sup; (10)}$

**Indice le plus à gauche de 8**

# Trace – indice à gauche de 8

 Le vecteur  $v$  contient 24 valeurs indicées de 0 à 23

  $v.size()-1 = 23$


  $8 \leq v[v.size()-1]$

```
} else {                                     // v.[v.size()-1] ≥ val
    int inf = 0;
    int sup = v.size()-1;                     // invariant vérifié
    int m;
    while (inf < sup) {
        ...
    }
    // inf = sup
    ...
}
```

 La fonction retournera : 7

# Détail – indice à gauche de 8 (1/6)

 initialisation


  $\text{inf} = 0, \text{sup} = 23$

 condition d'itération vraie, **itération 1**

  $m = 11, v[m] = 10 \geq 8 \text{ (val)}$


inf																							sup
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 11 \text{ (m)}$

# Détail – indice à gauche de 8 (2/6)

 valeurs des variables d'itération


  $\text{inf} = 0, \text{sup} = 11$

 condition d'itération vraie, **itération 2**

  $m = 5, v[m] = 6 < 8$  (val)


inf											sup												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération

  $\text{inf} = 6 (m + 1)$

# Détail – indice à gauche de 8 (3/6)

 valeurs des variables d'itération

  $\text{inf} = 6, \text{sup} = 11$

 condition d'itération vraie, **itération 3**

  $m = 8, v[m] = 9 \geq 8$  (val)


0	1	2	3	4	5	<b>inf</b>		7	8	9	10	<b>sup</b>		12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 8$  (m)

# Détail – indice à gauche de 8 (4/6)

 valeurs des variables d'itération

  $\text{inf} = 6, \text{sup} = 8$

 condition d'itération vraie, **itération 4**

  $m = 7, v[m] = 8 \geq 8$  (**val**)

						<b>inf</b>		<b>sup</b>															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85


 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 7$  (**m**)



# Détail – indice à gauche de 8 (5/6)

 valeurs des variables d'itération


  $\text{inf} = 6, \text{sup} = 7$

 condition d'itération vraie, **itération 5**

  $m = 6, v[m] = 7 < 8 \text{ (val)}$


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération

  $\text{inf} = 7 (m + 1)$

# Détail – indice à gauche de 8 (6/6)


 valeurs des variables d'itération

  $\text{inf} = 7, \text{sup} = 7$

 condition d'itération fausse

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85


 retour du résultat

  $v[\text{sup}] = 8 \text{ (val)} \rightarrow \text{return sup; (7)}$

**Indice le plus à gauche de 16**

# Trace – indice à gauche de 16

 Le vecteur  $v$  contient 24 valeurs indicées de 0 à 23

  $v.size()-1 = 23$


  $16 \leq v[v.size()-1]$

```
} else {                                     // v.[v.size()-1] ≥ val
    int inf = 0;
    int sup = v.size()-1;                     // invariant vérifié
    int m;
    while (inf < sup) {
        ...
    }
    // inf = sup
    ...
}
```


 La fonction retournera : -17

# Détail – indice à gauche de 16 (1/5)

 initialisation


  $\text{inf} = 0, \text{sup} = 23$

 condition d'itération vraie, **itération 1**

  $m = 11, v[m] = 10 < 16 \text{ (val)}$

inf																							sup
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85

 traiter  $v[m]$  et retour à l'itération


  $\text{inf} = 11 (m + 1)$

# Détail – indice à gauche de 16 (2/5)

 valeurs des variables d'itération


  $\text{inf} = 12, \text{sup} = 23$

 condition d'itération vraie, **itération 2**

  $m = 17, v[m] = 20 \geq 16$  (val)


												inf												sup	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23		
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

 traiter  $v[m]$  et retour à l'itération

  $\text{sup} = 17$  (m)

# Détail – indice à gauche de 16 (3/5)

 valeurs des variables d'itération


  $\text{inf} = 12, \text{sup} = 17$

 condition d'itération vraie, **itération 3**

  $m = 14, v[m] = 13 < 16$  (val)


0	1	2	3	4	5	6	7	8	9	10	11	<b>inf</b>		13	14	15	16	<b>sup</b>		19	20	21	22	23
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85	

 traiter  $v[m]$  et retour à l'itération


  $\text{inf} = 15$  ( $m + 1$ )

# Détail – indice à gauche de 16 (4/5)

 valeurs des variables d'itération


  $\text{inf} = 15, \text{sup} = 17$

 condition d'itération vraie, **itération 4**

  $m = 16, v[m] = 15 < 16$  (val)

															<b>inf</b>		<b>sup</b>															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23									
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85									

 traiter  $v[m]$  et retour à l'itération

  $\text{inf} = 17$  ( $m + 1$ )



# Détail – indice à gauche de 16 (5/5)

 valeurs des variables d'itération

  $\text{inf} = 17, \text{sup} = 17$

 condition d'itération fausse

																inf	sup								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23		
1	2	3	4	5	6	7	8	9	10	10	10	11	12	13	14	15	20	28	37	45	63	78	85		

 retour du résultat

  $v[\text{sup}] \neq 16 \text{ (val)} \rightarrow \text{return } -\text{sup}; (-17)$