



**LINFO 1121**  
**DATA STRUCTURES AND ALGORITHMS**



Algorithmes de tri et recherche dichotomique

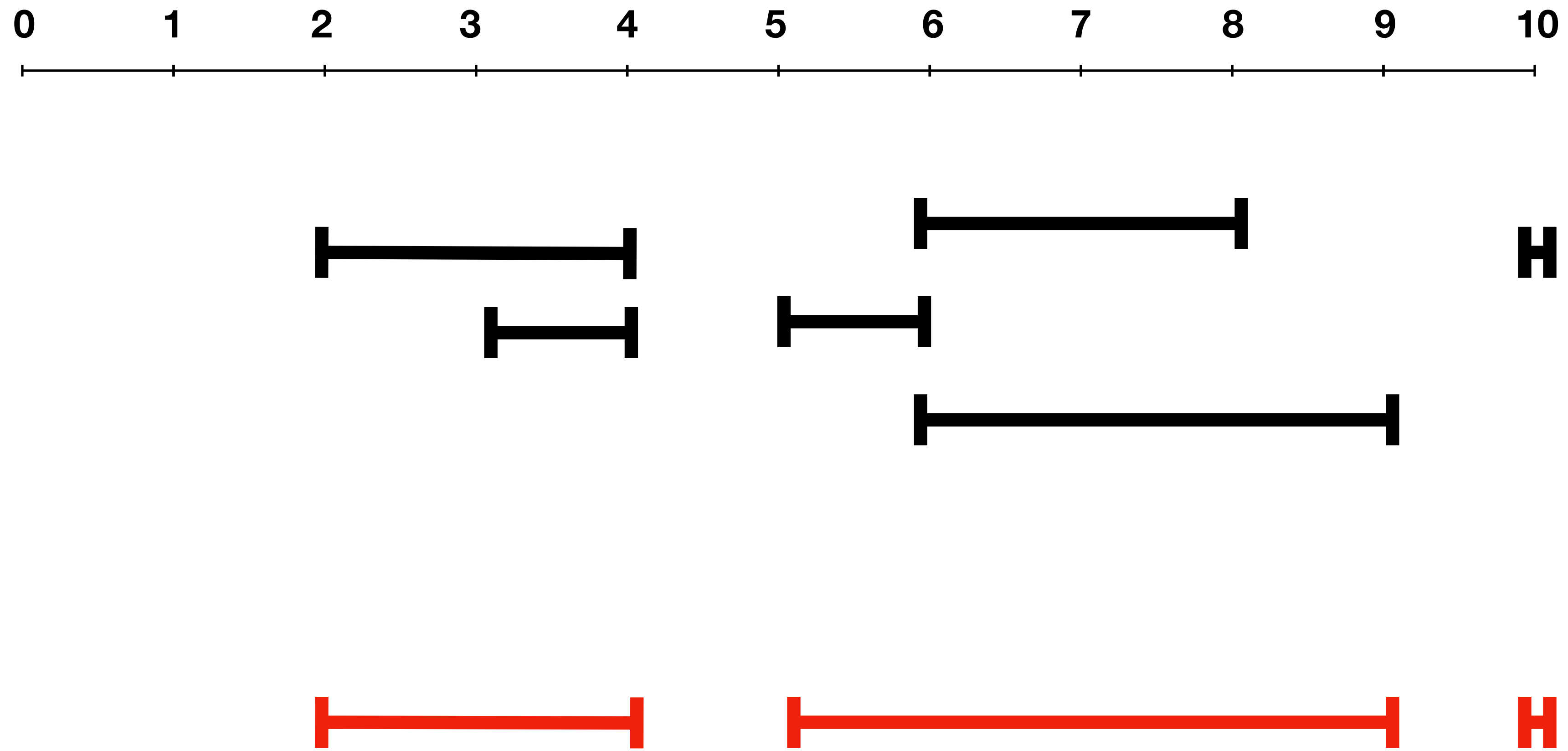
*Pierre Schaus*

# Union d'intervalles

Écrivez une méthode qui prend en entrée un tableau d'intervalles et qui retourne l'union de ces intervalles comme un tableau d'intervalles disjoints. On considère que les intervalles d'input sont donnés sous la forme de deux tableaux  $int[] \text{ min}$ ,  $int[] \text{ max}$ ; où le  $i$ ème intervalle est donné par  $(\text{min}[i], \text{max}[i])$ . Exemple d'entrée  $\text{min}=[5,0,1,6,2]$   $\text{max}=[7,2,2,8,3]$  donnerait en sortie  $\text{min}=[0,5]$ ,  $\text{max}=[3,8]$ .

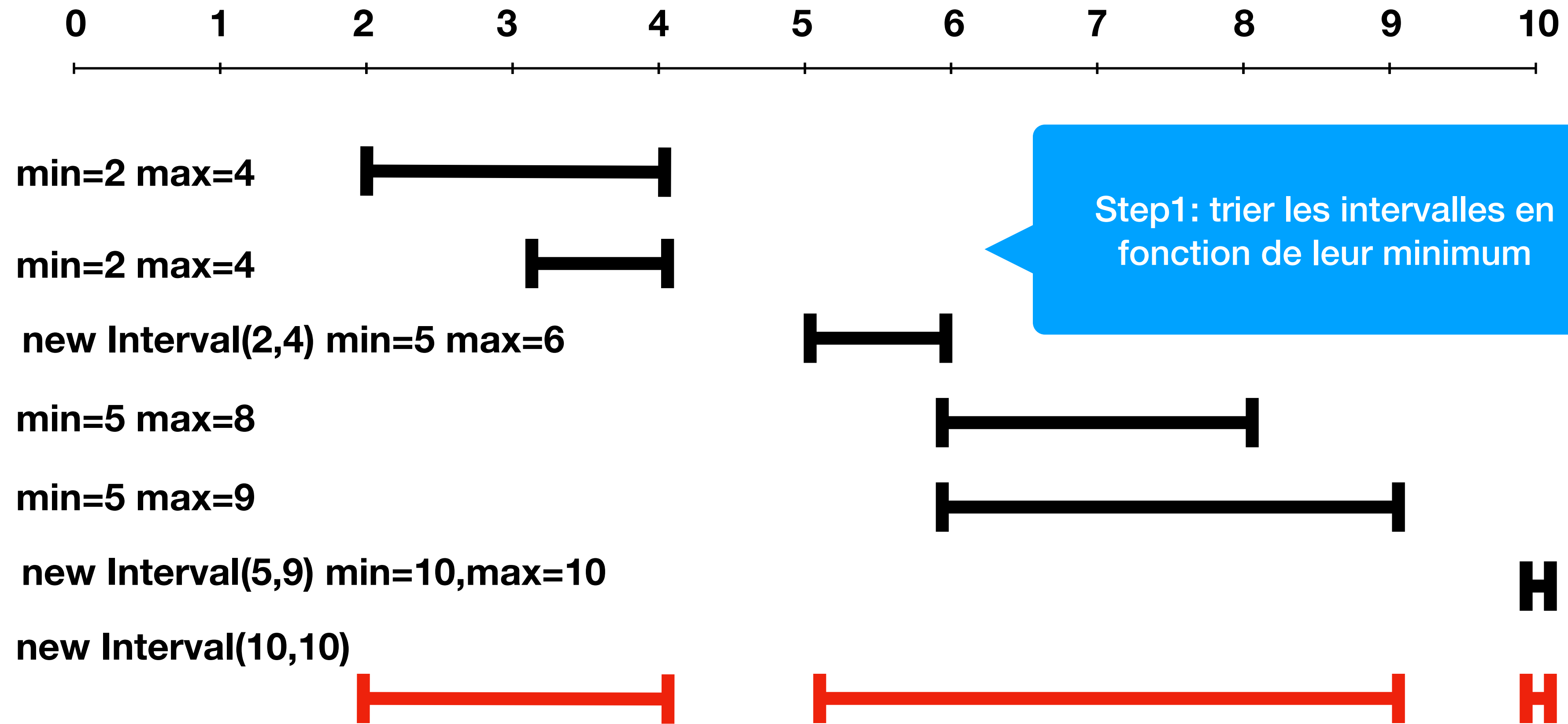
Ecrivez le pseudo-code. Quelle est la complexité de votre méthode ?

# Union of intervals



Input: [10,10],[2,4],[3,4],[5,6],[6,9],[6,8] Output: [2,4],[5,9],[10,10]

# Algo1



Step2: Considerer les intervalles triés un par un et mettre à jour les valeur min/max

# Algo1

```
public static Interval [] union(Interval [] intervals) {
    if (intervals.length == 0) return new Interval[0];
    ArrayList<Interval> res = new ArrayList<>();
    Arrays.sort(intervals);
    int min = intervals[0].min;
    int max = intervals[0].max;
    int i = 1;
    while (i < intervals.length) {
        if (intervals[i].min > max) {
            res.add(new Interval(min,max));
            min = intervals[i].min;
            max = intervals[i].max;
        } else {
            max = Math.max(max,intervals[i].max);
        }
        i++;
    }
    res.add(new Interval(min,max));
    return res.toArray(new Interval[0]);
}
```

! We change the input, not always considered a good practice

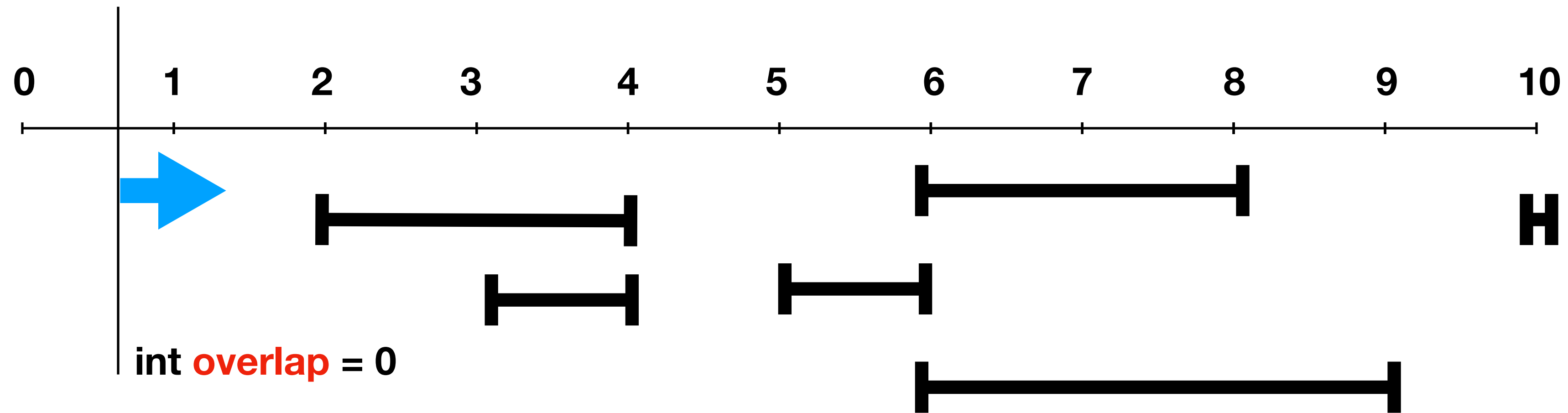
# Complexité temporelle ?

- $O(n)$  où  $n$  est le nombre d'intervalles en input
- $\Theta(n)$  où  $n$  est le nombre d'intervalles en input
- $O(n \log(n))$  où  $n$  est le nombre d'intervalles en input
- $\Theta(n \log(n))$  où  $n$  est le nombre d'intervalles en input

**wooclap**



# Autre approche: algorithme de Sweep (plus générique)



- On crée une séquence triée d'événements en début et fin d'intervalle:
  - +1 pour ouverture
  - -1 pour fermeture
- $(2,+1),(3,+1),(4,-1),(4,-1),(5,+1),(6,+1),(6,+1)(6,-1),(8,-1),(9,-1),(10,+1),(10,-1)$
- Un compteur "**overlap**" va être maintenu représentant le nombre d'intervalle intersection la ligne de sweep.
- Pour chaque nouveau pas de temps, on traite tous les événements un par un et on met à jour le compteur.
- Si lorsqu'on a traité tous les événements le compteur est à zero, on a détecté un nouvel intervalle.

# Sweep Line Algo: pseudo code

```
public static List<int[]> computeUnion(int[] min, int[] max) {
    int n = min.length;
    Point[] S = new Point[2 * n + 1];

    for (int i = 0; i < n; i++) {
        S[2 * i] = new Point(min[i], 1);
        S[2 * i + 1] = new Point(max[i] + 1, -1);
    }

    S[2 * n] = new Point(Integer.MAX_VALUE, 0);
    Arrays.sort(S);

    List<int[]> union = new ArrayList<>();
    int start = S[0].x;
    int overlap = S[0].delta;

    int i = 1;
    while (i < 2 * n) {
        overlap += S[i].delta;
        if (overlap == 0) {
            union.add(new int[]{start, S[i].x});
            start = S[i + 1].x;
        }
        i += 1;
    }

    return union;
}
```

Creation des "event points"

Process des "event points" un à un

Detection d'un intervalle

```
class Point implements Comparable<Point> {
    int x;
    int delta;

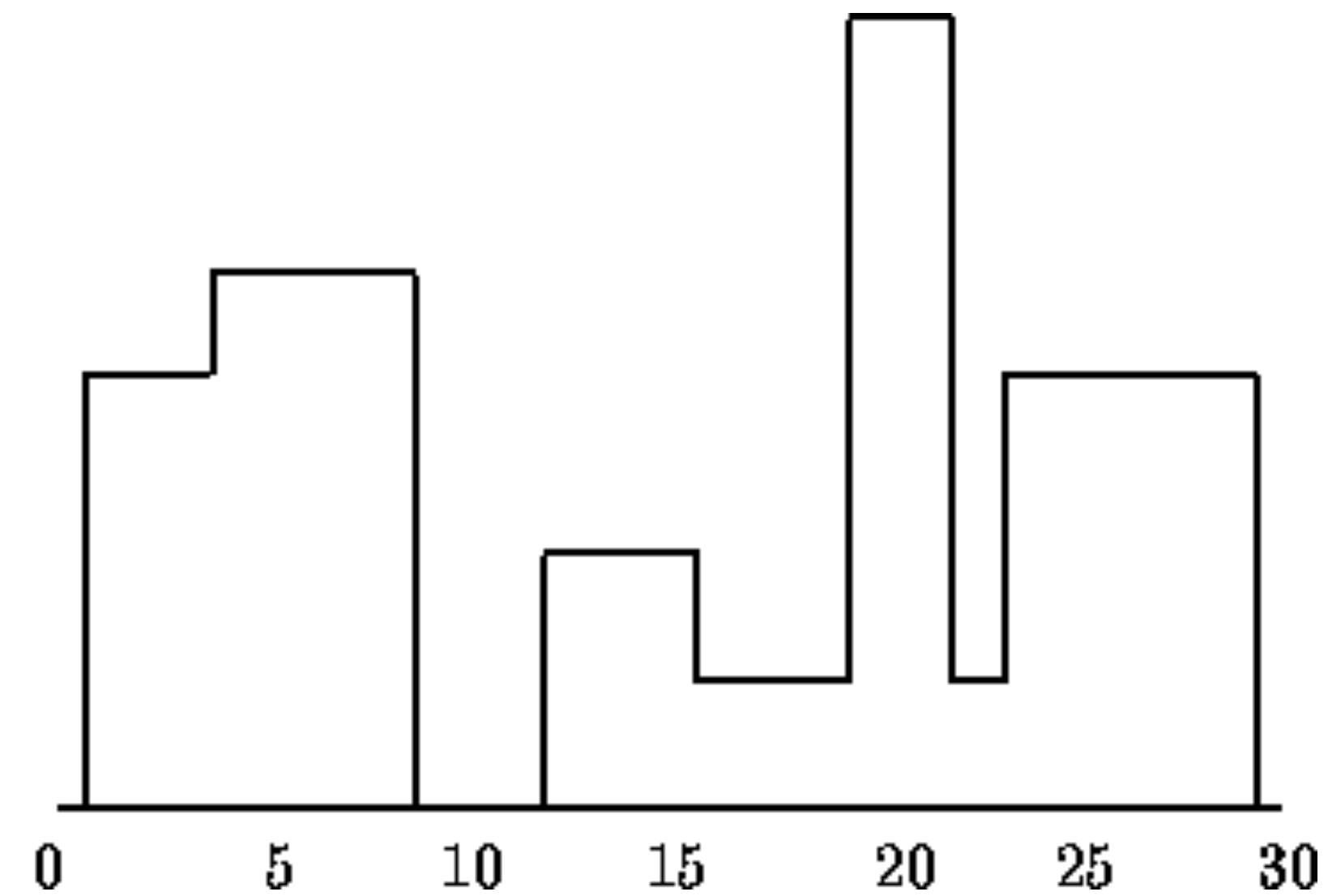
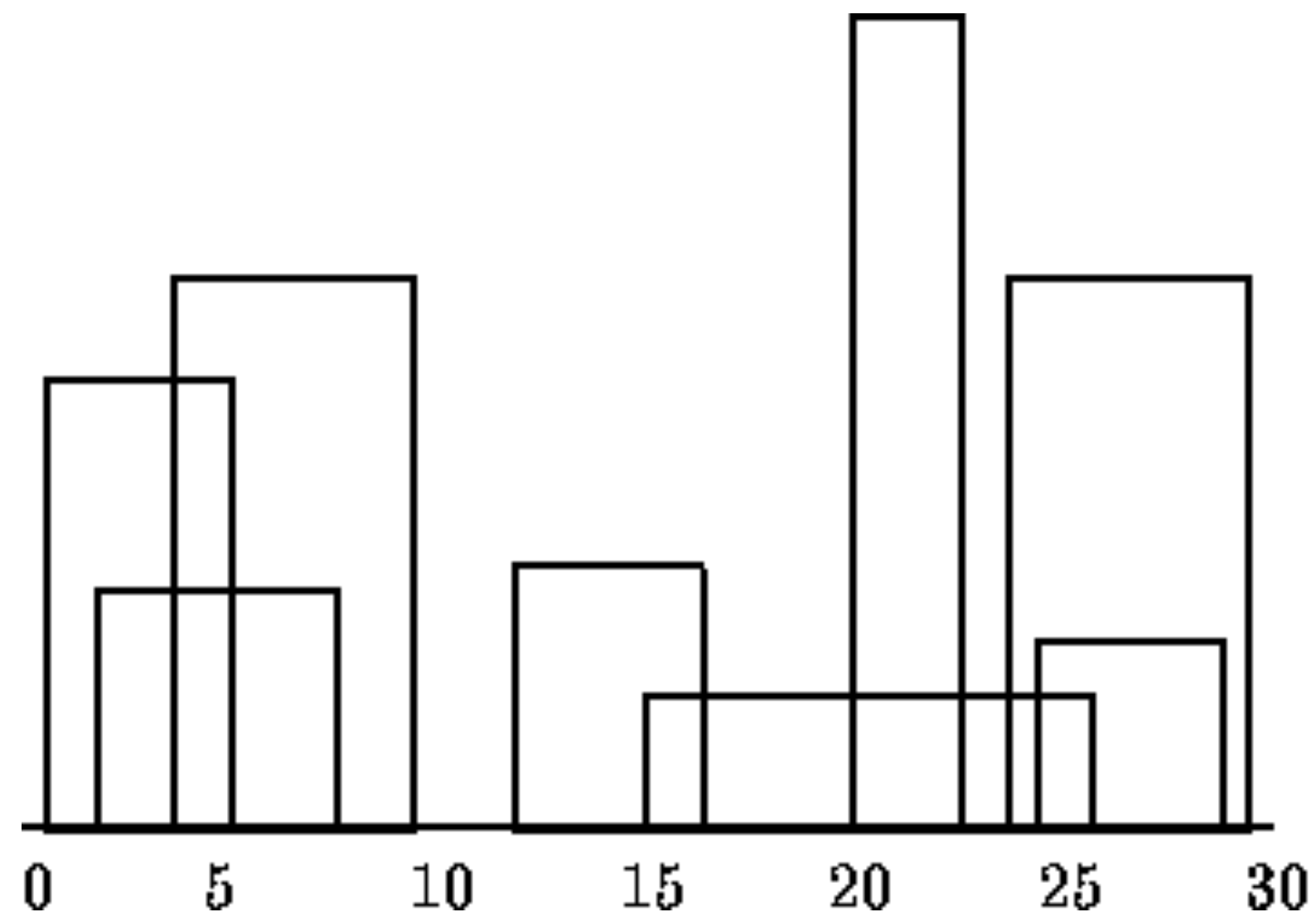
    Point(int x, int delta) {
        this.x = x;
        this.delta = delta;
    }

    @Override
    public int compareTo(Point o) {
        if (this.x != o.x) return Integer.compare(this.x, o.x);
        return -Integer.compare(this.delta, o.delta);
    }
}
```



# Algorithme de Sweep

- Fréquemment utilisé en “computational geometry”, par exemple pour calculer une “skyline” au départ d’un ensemble de rectangle (exercice).



# Question2

Vous devez trier un grand tableau qui a pour propriété qu'il ne contient que des valeurs dans l'ensemble  $\{0, 1, 2\}$ .

- Quel algorithme de tri suggérez-vous?
- Ecrivez le code.
- Quelle sera la complexité pour trier le tableau?
- Discutez cette complexité par rapport à la borne inférieure d'un algorithme de tri (Proposition 1 pages 280-281).

# Warmup

- Si je vous dis que je dois trier un ensemble qui contient
  - 1x0, 3x1, 2x5, est-ce que vous êtes capables de reconstruire l'input trié ?
    - Oui bien sûr: [0,1,1,1,5,5]
- L'algorithme "counting" sort consiste à reconstruire l'input trié sur base des compteurs

# Reconstruction de l'input

- Quelle est la complexité pour reconstruire l'input trié pour des valeurs de compteur  $k_1 \times 0$ ,  $k_2 \times 1$ ,  $k_3 \times 2$  ?
  - $O(k_1+k_2+k_3)$
  - $\Theta(k_1+k_2+k_3)$
  - $O(n \log(n))$  où  $n=k_1+k_2+k_3$
  - $\Theta(n \log(n))$  où  $n = k_1+k_2+k_3$

# Counting sort

```
int [] input =
    new int[]{0,1,2,0,0,2,0,1,0,2,1,0,1,1,0,0,2,2,2,2,0};
int [] counters = new int[3];
for (int i : input) counters[i]++;
int [] output = new int[input.length];
int j = 0;
for (int i = 0; i <= 2; i++) {
    while (counters[i] > 0) {
        counters[i]--;
        output[j] = i;
        j++;
    }
}
```

# Complexité Counting sort sur des valeurs $\{0,1,2\}$

- $n$  = nombre de valeurs à trier
  - $O(n)$
  - $\Theta(n)$
  - $O(n \log(n))$
  - $\Theta(n \log(n))$
  - $O(n^2)$
  - $\Theta(n^2)$

**wooclap**



# Est-ce que le counting sort fonctionne toujours ?

- Si je dois trier des valeurs entre 0 et k ? Quelle est la complexité de Counting-sort pour trier n valeurs ?

- $O(n)$
- $\Theta(n)$
- $O(n+k)$
- $\Theta(n+k)$
- $O(n \log(n))$
- $\Theta(n \log(n))$

wooclap

```
public static void countingSort(int[] values) {  
    int[] buckets = new int[100]; // k = 100  
    // Count occurrences of each value in the input array  
    for (int i : values) {  
        buckets[i]++;  
    }  
    int idx = 0;  
    for (int i = 0; i < 100; i++) {  
        for (int j = 0; j < buckets[i]; j++) {  
            values[idx] = i;  
            idx++;  
        }  
    }  
}
```

# More robust counting sort

- Mon input contient seulement trois valeurs possibles  $\{20007, 1368910, 900045\}$ , est-ce que je peux adapter counting-sort pour obtenir une complexité de  $\Theta(n)$  ?

- Discutez cette complexité par rapport à la borne inférieure d'un algorithme de tri (Proposition 1 pages 280-281).

**Prop1: Aucun algorithme basé sur les comparaisons ne peut garantir pouvoir trier  $N$  objets en moins que  $\sim N \lg N$  comparaisons.**

- Le counting sort n'est pas basé sur les comparaisons.
- Il fonctionne bien uniquement s'il y a peu de valeurs différentes à trier (= la différence entre la plus grande et la plus petite n'est pas trop grande  $\leq N$ ).

# Le mode d'un tableau

Le mode d'un tableau de nombres est le nombre qui apparaît le plus fréquemment dans le tableau.

Exemple  $\text{mode}([4,6,2,4,3,1]) = 4$ .

- Donnez un algorithme efficace pour calculer le mode d'un tableau de  $n$  nombres.
- Quid si on sait que le tableau ne contient que des valeurs de 0 à  $k$  ?

# Mode d'un tableau: Solution 1

1. Trier les éléments
2. Trouver l'élément qui se répète avec le plus de fois consécutivement en parcourant le tableau trié

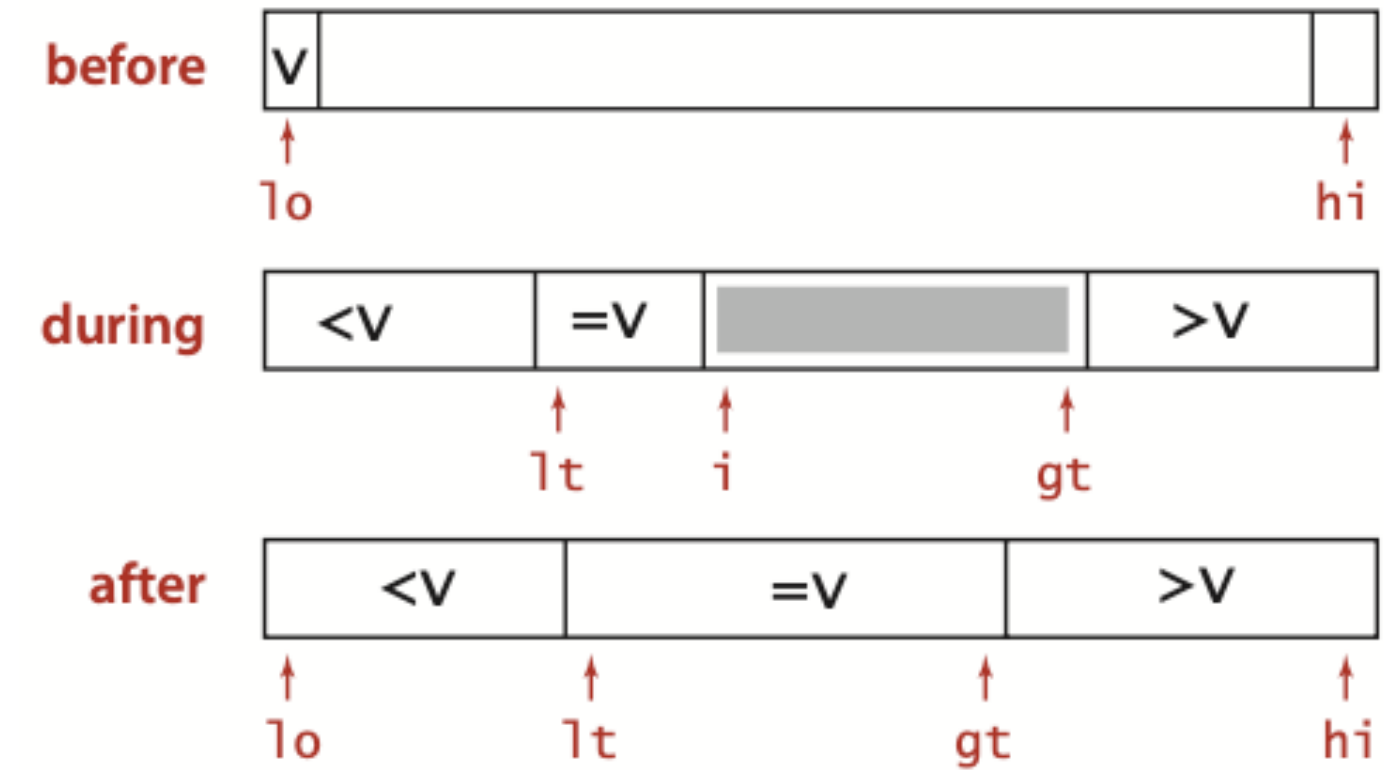
- Complexité:

- $O(n \log(n))$
- $\Theta(n \log(n))$
- $\Theta(n)$
- $O(n)$
- $\Theta(n^2)$

The logo for wooclap, with 'wooc' in dark blue and 'lap' in green.

# Recall 3-Way Partitioning

#v = gt-lt+1



3-way partitioning overview

```
public class Quick3way
{
    private static void sort(Comparable[] a, int lo, int hi)
    { // See page 289 for public sort() that calls this method.
        if (hi <= lo) return;
        int lt = lo, i = lo+1, gt = hi;
        Comparable v = a[lo];
        while (i <= gt)
        {
            int cmp = a[i].compareTo(v);
            if (cmp < 0) exch(a, lt++, i++);
            else if (cmp > 0) exch(a, i, gt--);
            else i++;
        } // Now a[lo..lt-1] < v = a[lt..gt] < a[gt+1..hi].
        sort(a, lo, lt - 1);
        sort(a, gt + 1, hi);
    }
}
```

Do not call if lt-lo < bestMode



# Mode d'un tableau: Solution QuickSort

- Lors de l'étape du pivoting (méthode partition), nous pouvons compter le nombre d'éléments qui sont égaux au pivot et maintenir le meilleur *candidat mode* ainsi que sa fréquence.
- Cette information peut être utilisée pour éviter des appels récursifs dans quick sort:
  - Un appel récursif est lancé seulement si le sous-tableau à partitionner est plus grand que la fréquence du mode courant.

## Et si le range des valeurs est entre 0 et k ?

- Dans ce cas simplement compter les valeurs dans un tableau de taille k. Si  $k < n$ , complexité  $\Theta(n)$

# Trouver une paire

Étant donné deux ensembles  $S1$  et  $S2$  (chacun de taille  $n$ ), et un nombre  $x$ .

Décrivez un algorithme efficace pour trouver s'il existe une paire  $(a,b)$  avec  $a \in S1, b \in S2$  telle que  $a+b=x$ .

Quelle est la complexité de votre algorithme?

Quid si les ensembles sont dans des tableaux déjà triés ?

# S1 et S2 sont non triés et taille n

- Trier le premier ensemble S1
- Pour chaque valeur  $v$  de S2, on cherche une valeur  $x-v$  dans S1 trié avec une recherche dichotomique.
- Complexité ?
  - $O(n \log(n))$
  - $\Theta(n \log(n))$
  - $\Theta(n)$
  - $O(n)$
  - $\Theta(n^2)$

wooclap

# Quid si les deux ensembles sont déjà triés

- Dans ce cas quelques optimisations peuvent être mises en place mais cela ne change pas la complexité.
- Si pour une valeur  $v$  de  $S1$ ,  $v + \min(S2) > x$ , il ne faut pas lancer la recherche dichotomique pour les valeurs suivantes plus grandes que  $v$ .
- Cela consiste à réduire à l'avance les bornes des deux tableaux
- Exemple:  $x = 100$
- $S1 = [10, 13, 46, 70, 80, 101, 108]$
- $S2 = [10, 22, 30, 70, 82, 104, 111]$

# Et pour un seul tableau ?

- Trouver deux entrées dont la somme fait  $x$  ?
- Exemple  $[5, 10, 1, 150, 151, 155, 18, 50, 30]$   $x=68$
- Quid si le tableau est déjà trié ?
  - $[1, 5, 10, 18, 30, 50, 150, 151, 155]$   $x=68$



# Pour un tableau trié

```
/**
 * Find a pair (i, j) in array T such that T[i] + T[j] = x.
 *
 * @param T An array of integers, sorted in non-decreasing order.
 * @param x The target sum.
 * @return An array of two integers i and j if a pair is found, or null otherwise.
 */
public static int[] findPair(int[] T, int x) {
    int i = 0;
    int j = T.length - 1;

    while (i < j) {
        if (T[i] + T[j] > x) {
            j--;
        } else if (T[i] + T[j] < x) {
            i++;
        } else {
            return new int[] {i, j}; // Pair found
        }
    }

    return null; // Pair not found
}
```

Complexité ?

# Pourquoi cet algorithme fonctionne-t-il ?

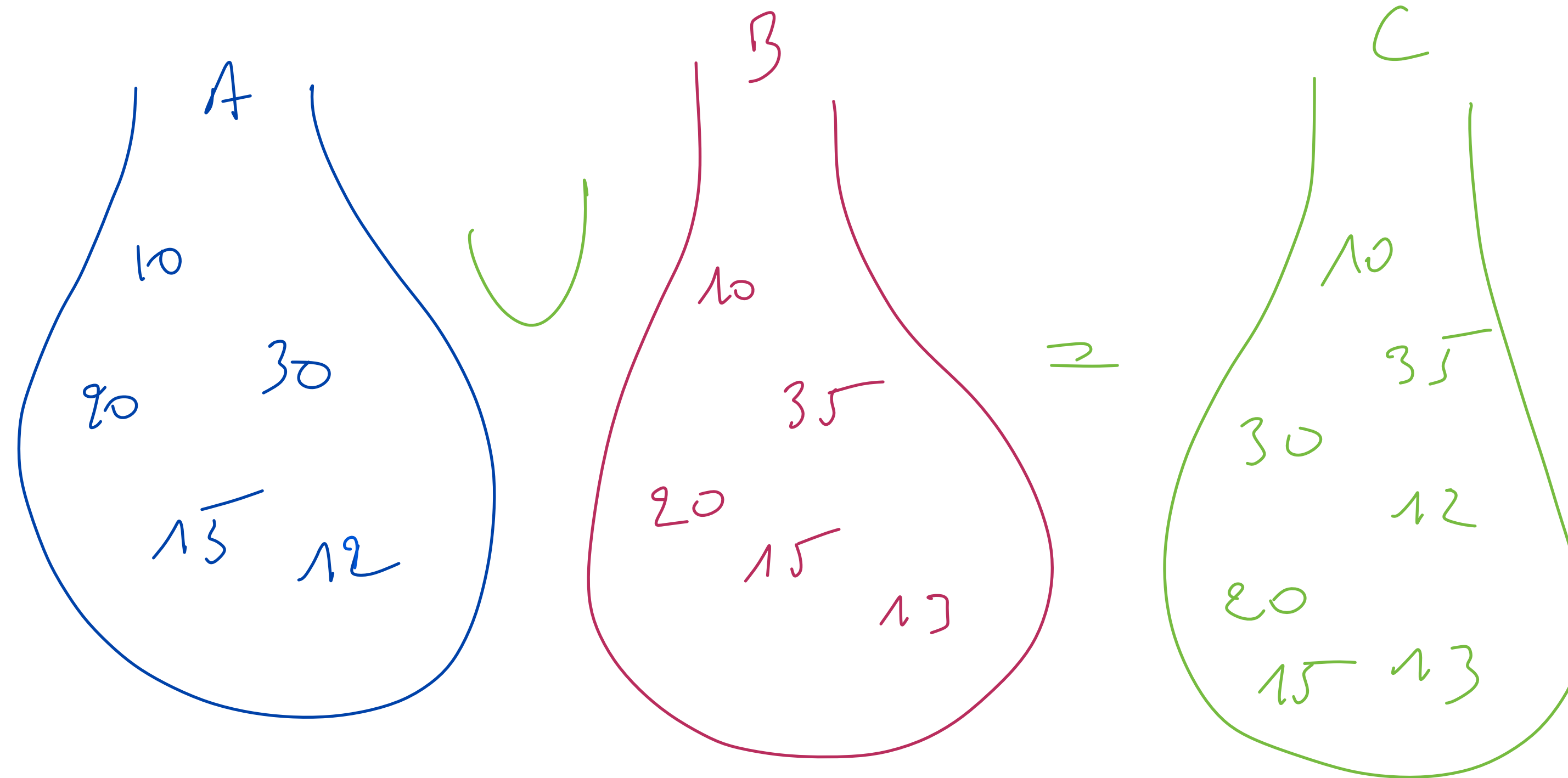
- Preuve par induction



- Hypothèse (n'importe quelle paire avec un element avant ( $<$ )  $i$  ou après ( $>$ )  $j$  ne fonctionne pas)
- Algo:  $T[i]+T[j] > x \Rightarrow j = j-1$ 
  - \* Il faut montrer qu'il n'existe pas de paire valable avec  $T[j]$ .
  - \*  $T[j]$  ne peut être apparié avec un element après  $i$  car les valeurs sont triées et ça ne ferait qu'éloigner davantage de  $x$ .
  - \* Par hypothèse,  $T[j]$  ne peut être apparié avec un élément avant  $i$  non plus.

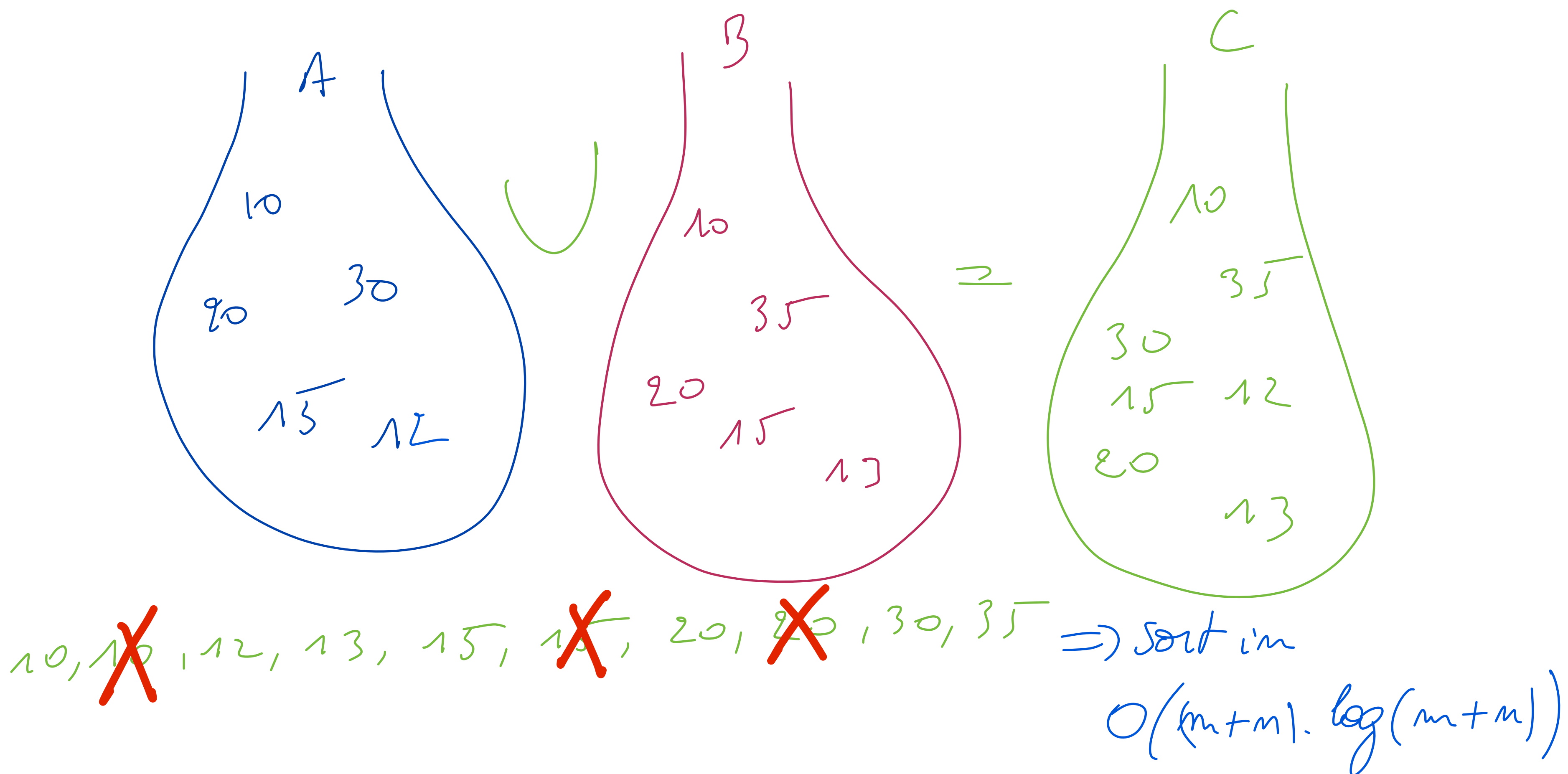
# Union de tableaux

- Donnez un algorithme pour calculer l'union de deux ensembles A et B.



# Union de deux tableaux: solution 1

- Soit  $m$  et  $n$  la taille des ensembles
- tout mettre dans un grand tableau et trier  $O((m+n) \log(m+n))$  et ensuite retirer les doublons.



# Solution 1

```
public static int[] union(int[] arr1, int[] arr2) {
    int n1 = arr1.length;
    int n2 = arr2.length;

    // Step 1: Concatenate the arrays
    int[] combined = new int[n1 + n2];
    System.arraycopy(arr1, 0, combined, 0, n1);
    System.arraycopy(arr2, 0, combined, n1, n2);

    // Step 2: Sort the array
    Arrays.sort(combined);

    // Step 3: Remove duplicates
    int[] temp = new int[n1 + n2];
    int j = 0;

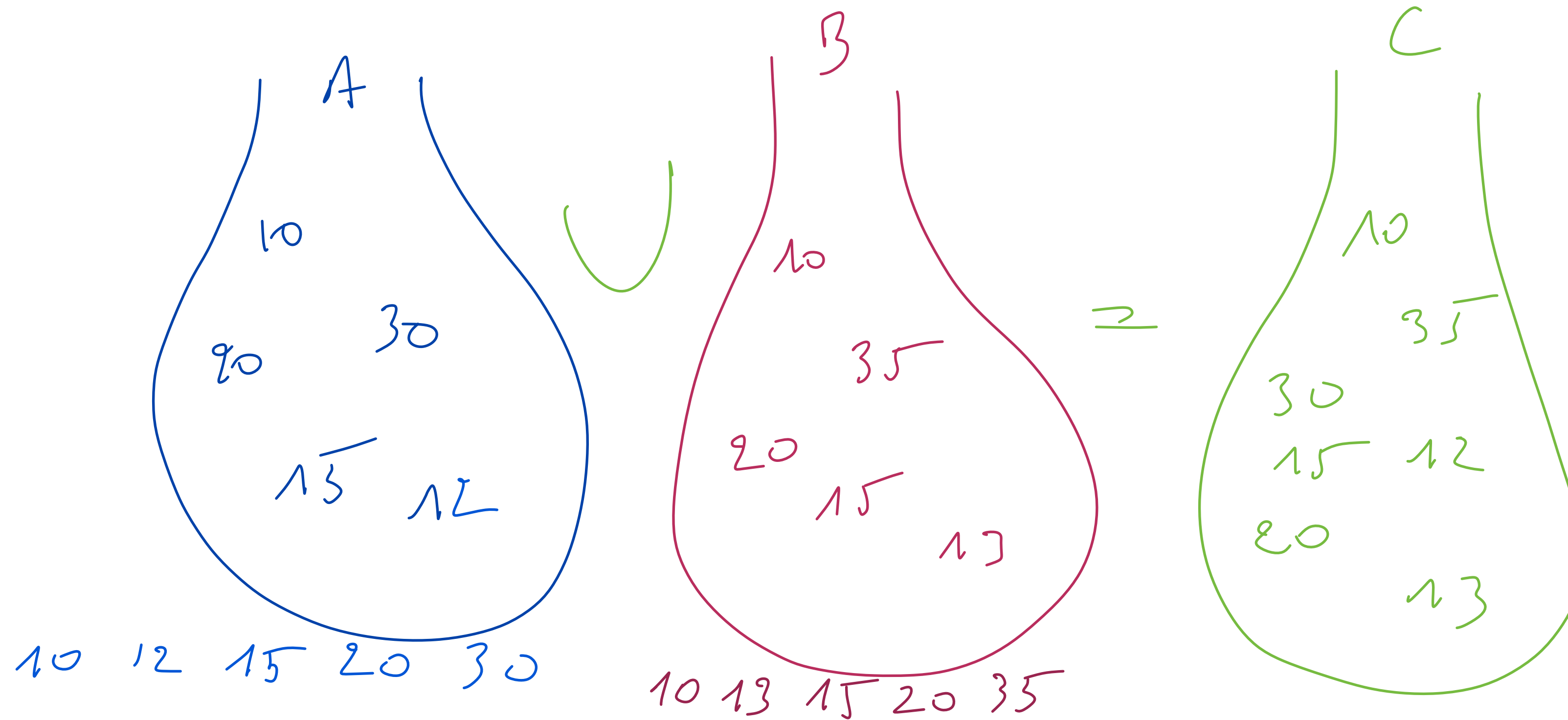
    for (int i = 0; i < n1 + n2 - 1; i++) {
        if (combined[i] != combined[i + 1]) {
            temp[j++] = combined[i];
        }
    }
    temp[j++] = combined[n1 + n2 - 1]; // Adding the last element

    // Trim temp array to actual size
    int[] result = new int[j];
    System.arraycopy(temp, 0, result, 0, j);

    return result;
}
```

# Union de deux tableaux: solution 2

- Trier chaque ensemble séparément et ensuite collecter les éléments en retirant les doublons.  $O(m \log(m) + n \log(n))$ .
- Si  $m = n$ , Solution1 a la même complexité temporelle que Solution2





# Solution 2

```
public static int[] union(int[] arr1, int[] arr2) {
    Arrays.sort(arr1);
    Arrays.sort(arr2);

    int[] temp = new int[arr1.length + arr2.length];
    int i = 0, j = 0, k = 0;

    while (i < arr1.length && j < arr2.length) {
        // Skip duplicates in arr1
        while (i < arr1.length - 1 && arr1[i] == arr1[i + 1]) {
            i++;
        }

        // Skip duplicates in arr2
        while (j < arr2.length - 1 && arr2[j] == arr2[j + 1]) {
            j++;
        }

        if (arr1[i] < arr2[j]) {
            temp[k++] = arr1[i++];
        } else if (arr1[i] > arr2[j]) {
            temp[k++] = arr2[j++];
        } else {
            temp[k++] = arr1[i];
            i++;
            j++;
        }
    }

    while (i < arr1.length) {
        if (i < arr1.length - 1 && arr1[i] == arr1[i + 1]) {
            i++;
            continue;
        }
        temp[k++] = arr1[i++];
    }

    while (j < arr2.length) {
        if (j < arr2.length - 1 && arr2[j] == arr2[j + 1]) {
            j++;
            continue;
        }
        temp[k++] = arr2[j++];
    }

    int[] result = new int[k];
    System.arraycopy(temp, 0, result, 0, k);

    return result;
}
```

# Union de deux tableaux: solution 3

- Trier seulement un des deux tableaux, pour chaque élément du tableau non trié, faire une recherche dichotomique sur le tableau trié pour vérifier s'il faut l'ajouter.
- Est-ce que le tableau qu'il faut trier a de l'importance ?
- Supposons une grosse différence de tailles sur les tableaux, par exemple de taille  $n$  et  $n^k$ 
  - Pour chaque élément du petit, on fait une recherche dichotomique sur le grand:  $O(n \log(n^k))$ . Mais il faut trier le grand:  $O(n^k \log(n^k))$ . Donc au total  $O(k n^k \log(n))$
  - Pour chaque élément du grand, on fait une recherche dichotomique sur le petit  $O(n^k \log(n))$ . Mais il faut trier le petit:  $O(n \log(n))$ . Donc au total  $O(n^k \log(n))$ .
  - Conclusion: si grosses différences de tailles, il vaut mieux trier le petit seulement.

# Solution 3

```
public static boolean binarySearch(int[] arr, int target) {
    int left = 0;
    int right = arr.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) return true;
        if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }

    return false;
}

public static int[] union(int[] arr1, int[] arr2) {
    Arrays.sort(arr1);

    int[] temp = new int[arr1.length + arr2.length];
    int k = 0, i;

    // Add unique elements from arr1 to result
    for (i = 0; i < arr1.length; i++) {
        if (i == 0 || arr1[i] != arr1[i - 1]) {
            temp[k++] = arr1[i];
        }
    }

    // Check arr2 elements against arr1
    for (int val : arr2) {
        if (!binarySearch(arr1, val)) {
            temp[k++] = val;
        }
    }

    int[] result = new int[k];
    System.arraycopy(temp, 0, result, 0, k);

    return result;
}
```

# Variante du problème

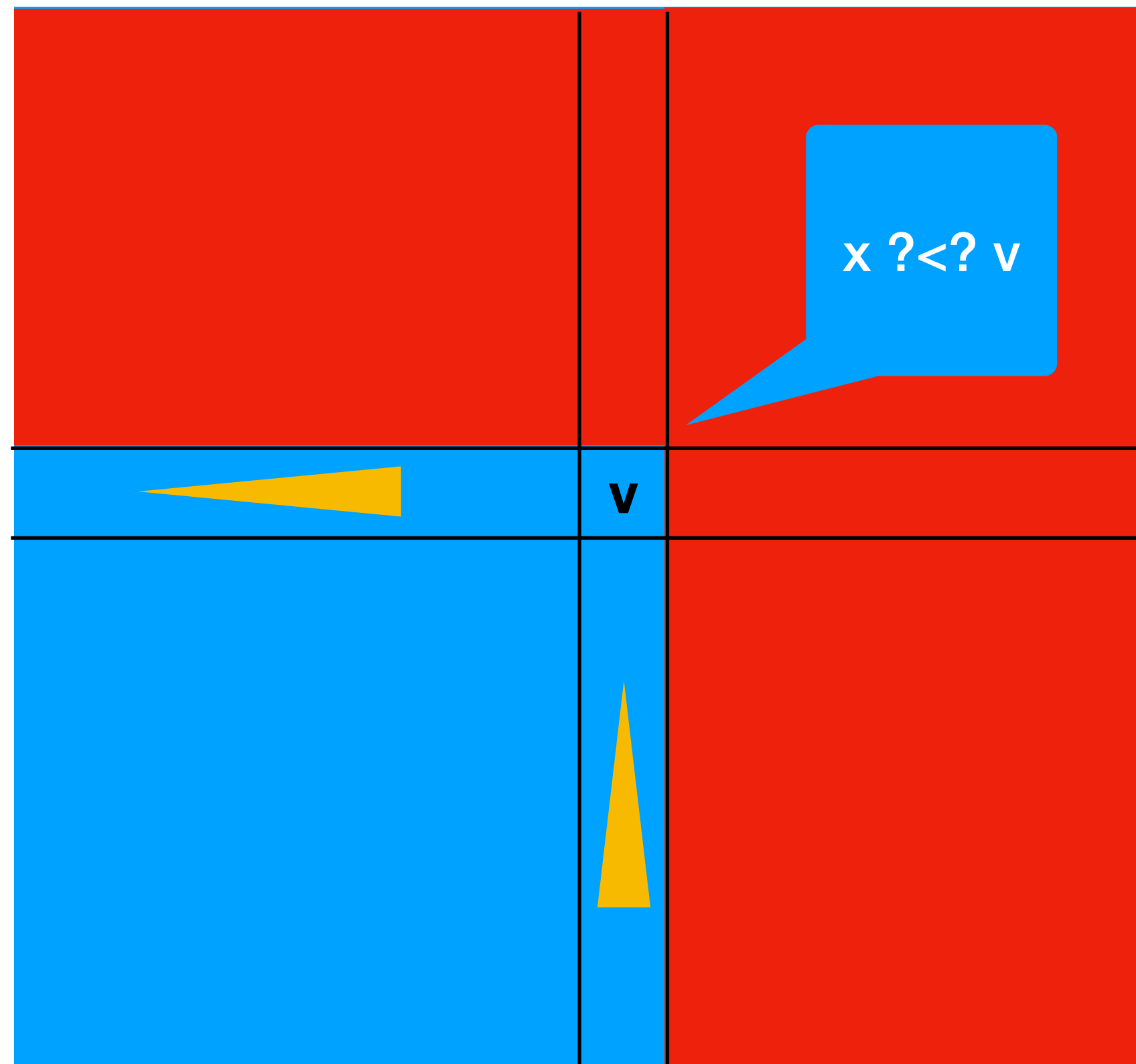
- Supposons dans un second temps, que l'ensemble A déjà trié a une taille  $n$  et l'ensemble B également trié a une taille  $n^2$ .
- Quelle seraient la complexité, est-ce que votre algorithme change (think about the merge in a merge sort) ?

# Recherche dans une matrice

- Étant donné une matrice de nombres entiers qui sont **triés le long des lignes et des colonnes**, comment **trouver un nombre donné** dans la matrice de manière efficace ?
- Indice: Il existe un algorithme en temps  $O(n+m)$  pour une matrice  $n \times m$ .

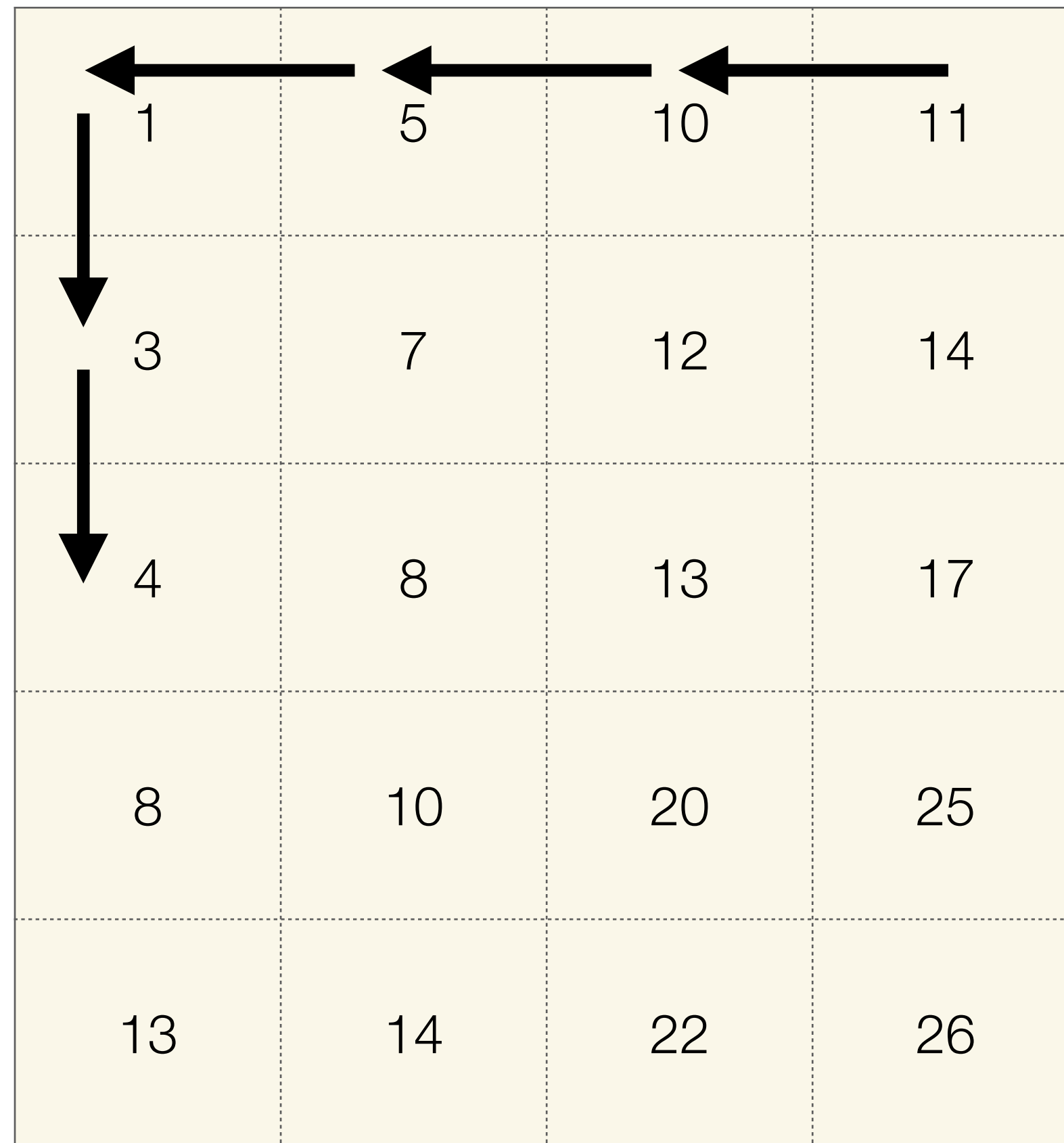
# Algo

- Pour cela commencez dans le coin supérieur droit et comparez avec le nombre recherché  $x$ .
- Quelles parties de la matrice pouvez-vous élaguer dans votre recherche en fonction du résultat?




- Recherche de 4

1	5	10	11
3	7	12	14
4	8	13	17
8	10	20	25
13	14	22	26



- Recherche de 20

1	5	10	11
3	7	12	14
4	8	13	17
8	10	20	25
13	14	22	26





# Pour encore s'entraîner

Réorganiser un tableau d'éléments identifiés par leur couleur, sachant que seules trois couleurs sont présentes (par exemple, rouge, blanc, bleu, dans le cas du drapeau des Pays-Bas).

Votre algorithme doit avoir deux propriétés:

- In place (ne peut créer de nouveau tableau)
- Stable (l'ordre relatif des éléments de mêmes couleurs doit être conservé)

# Solution 1: Quicksort partition 2 times

```
public static void flagSort(Couleur[] tableau) {
    int i = 0;

    // Partition pour ROUGE
    for (int j = 0; j < tableau.length; j++) {
        if (tableau[j] == Couleur.ROUGE) {
            // échanger tableau[i] et tableau[j]
            Couleur temp = tableau[i];
            tableau[i] = tableau[j];
            tableau[j] = temp;
            i++;
        }
    }

    // Partition pour BLANC
    for (int j = i; j < tableau.length; j++) {
        if (tableau[j] == Couleur.BLANC) {
            // échanger tableau[i] et tableau[j]
            Couleur temp = tableau[i];
            tableau[i] = tableau[j];
            tableau[j] = temp;
            i++;
        }
    }
}
```



**In place OK, stable KO**

# Solution 2

```
public static void flag(Couleur[] tableau) {
    int n = tableau.length;
    int indexRouge = 0; // Position to insert the next ROUGE color
    int indexBleu = n - 1; // Position to insert the next BLEU color
    int i = 0;
    while (i <= indexBleu) {
        if (tableau[i] == Couleur.ROUGE && i > indexRouge) {
            swap(tableau, i, indexRouge);
            indexRouge++;
        } else if (tableau[i] == Couleur.BLEU && i < indexBleu) {
            swap(tableau, i, indexBleu);
            indexBleu--;
        } else {
            i++;
        }
    }
}
```



```
public static void swap(Couleur[] tableau, int i, int j) {
    Couleur temp = tableau[i];
    tableau[i] = tableau[j];
    tableau[j] = temp;
}
```

**In place OK, stable KO**





**LINFO 1121**  
**DATA STRUCTURES AND ALGORITHMS**



Les arbres de recherches

*Pierre Schaus*

# Les tables de symboles

- Type abstrait de données permettant:
  - D'**insérer** une valeur avec une clef 
  - Etant donné la clef  de **retrouver** la valeur correspondante



# Quelques exemples

Application	Objectif de la recherche	clef	Valeur
Web search	Trouver des pages intéressantes	Mot-clefs	Liste de pages
DNS inversé	Trouver l'adresse IP	Nom de domaine	Adresse IP
Système de fichier	Trouver un fichier	Nom du fichier	Localisation sur le disque
Compilateur	Trouver les propriétés d'une variable	Nom de la variable	Type et valeur
Table de routage	Trouver un chemin de routage pour un paquet	Destination	Interface de sortie du routeur (meilleure route)

# Un tableau peut être vu comme une table de symboles

- Les clefs sont les indices
- Les valeurs sont les entrées du tableau
- Python adopte cette convention. `tab[i]` fonctionne pour les tableaux et les dictionnaires

# Convention du livre

- Une clef ne peut se retrouver qu'une seule fois dans la table de symbole.
- Les clefs sont différentes de null et les valeurs également.
  - En effet `get(key)` retourne null si la clef n'est pas présente
- Etant donné que l'égalité des clefs est basée sur la méthode "equals", il est plus prudent que les clefs soient des objets immuables.



# Rappel sur l'égalité et equals

- Propriétés demandées:
  - Réflexivité:  $x.equals(x)$  est vrai
  - Symétrique:  $x.equals(y) \iff y.equals(x)$
  - Transitivité:  $x.equals(y)$  et  $y.equals(z) \implies x.equals(z)$
  - Non null:  $x.equals(y)$  est faux

# Exemple

```
public final class Date implements Comparable<Date>
{
    private final int month;
    private final int day;
    private final int year;

    public boolean equals(Object y)
    {
        if (y == this) return true;
        if (y == null) return false;
        if (y.getClass() != this.getClass())
            return false;
        Date that = (Date) y;
        if (this.day != that.day ) return false;
        if (this.month != that.month) return false;
        if (this.year != that.year ) return false;
        return true;
    }
}
```

Contrat de base

La classe est final donc la question ne se pose pas

Le cast ne peut pas échouer

On teste tous les champs.  
!!! Ici ce sont des types primitifs !!!  
Sinon il aurait fallu utiliser equals et si c'était un tableau, il faudrait tester l'égalité sur chaque élément du tableau  
(deepEqual)

# Implémentation possible des tables de symboles

- Implémentations possibles pour une tables de symboles:
  - Sequential Search = simple liste chaînée qui contient les clefs et les valeurs
    - \* insert(key,value) en  $O(n)$ ,
    - \* get(key)/delete(key) en  $O(n)$
- En partie 4 nous verrons une autre structure (table de hashage) qui permet  $O(1)$  amortie pour toutes les opérations. Patience ...

# Relation d'ordre entre les clefs

- Il existe souvent une relation d'ordre entre les clefs. On parle alors généralement de dictionnaire

```
public class ST<Key extends Comparable<Key>, Value>
```

...

Key min()	<i>smallest key</i>
Key max()	<i>largest key</i>
Key floor(Key key)	<i>largest key less than or equal to key</i>
Key ceiling(Key key)	<i>smallest key greater than or equal to key</i>
int rank(Key key)	<i>number of keys less than key</i>
Key select(int k)	<i>key of rank k</i>
void deleteMin()	<i>delete smallest key</i>
void deleteMax()	<i>delete largest key</i>
int size(Key lo, Key hi)	<i>number of keys between lo and hi</i>
Iterable<Key> keys()	<i>all keys, in sorted order</i>
Iterable<Key> keys(Key lo, Key hi)	<i>keys between lo and hi, in sorted order</i>

# Utilisation de l'ordre

- Pour améliorer les complexités grâce à la recherche binaire (= dichotomique)

	sequential search	binary search
search	$N$	$\log N$
insert / delete	$N$	$N$
min / max	$N$	1
floor / ceiling	$N$	$\log N$
rank	$N$	$\log N$
select	$N$	1
ordered iteration	$N \log N$	$N$

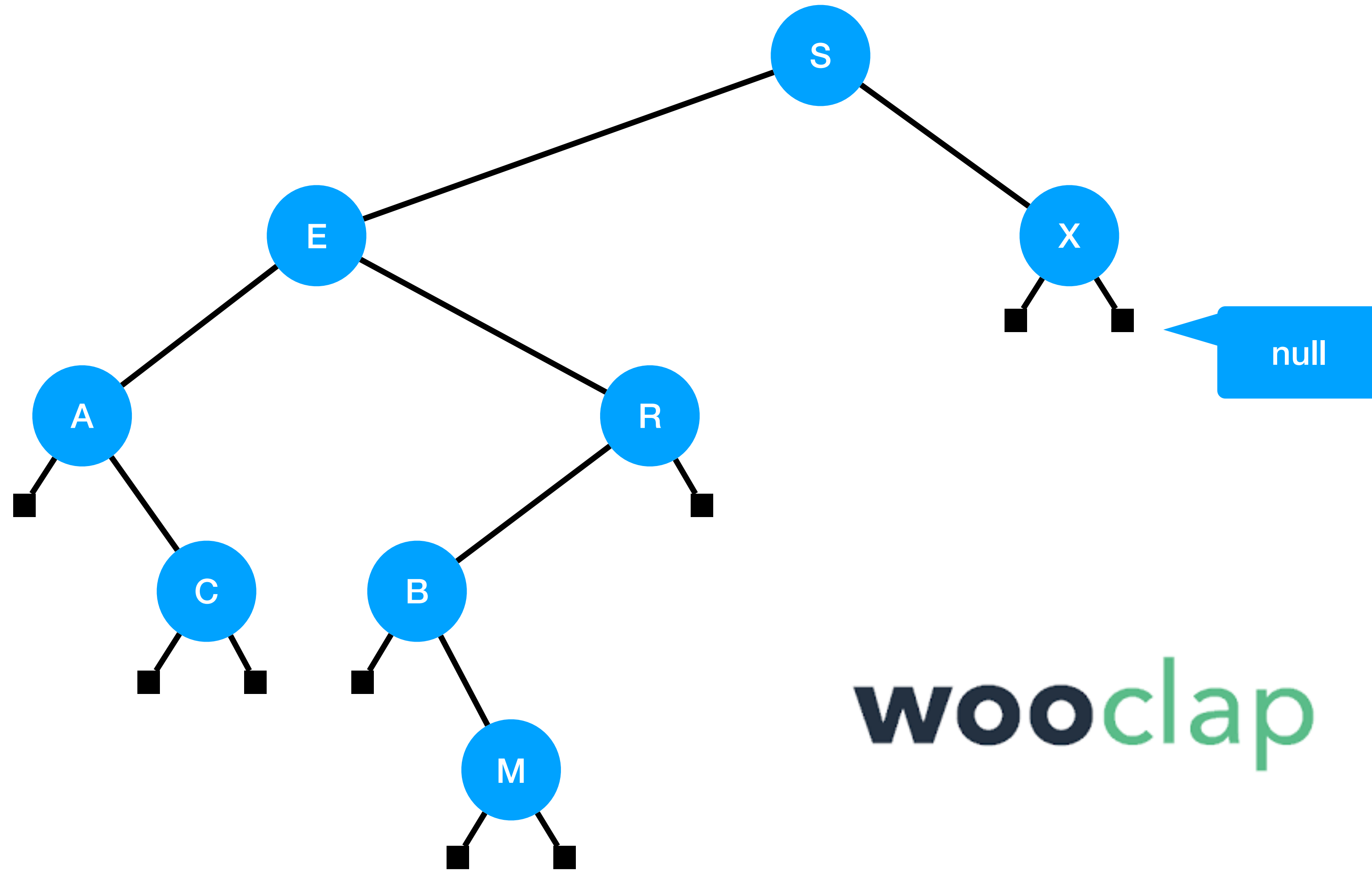
On peut encore améliorer grâce à une nouvelle structure de données: Les arbres de recherche

# Les arbres de recherche

- Arbre binaire:
  - \* Vide
  - \* Non-vide: Il contient deux arbres binaire (gauche et droit)
- Arbre binaire de recherche = arbre binaire avec une clef-valeur dans chaque noeud telle que
  - \* Cette clef est plus grande que **toutes** les clefs dans le sous arbre de gauche
  - \* Cette clef est plus petite que **toutes** les clefs dans le sous arbre de droite

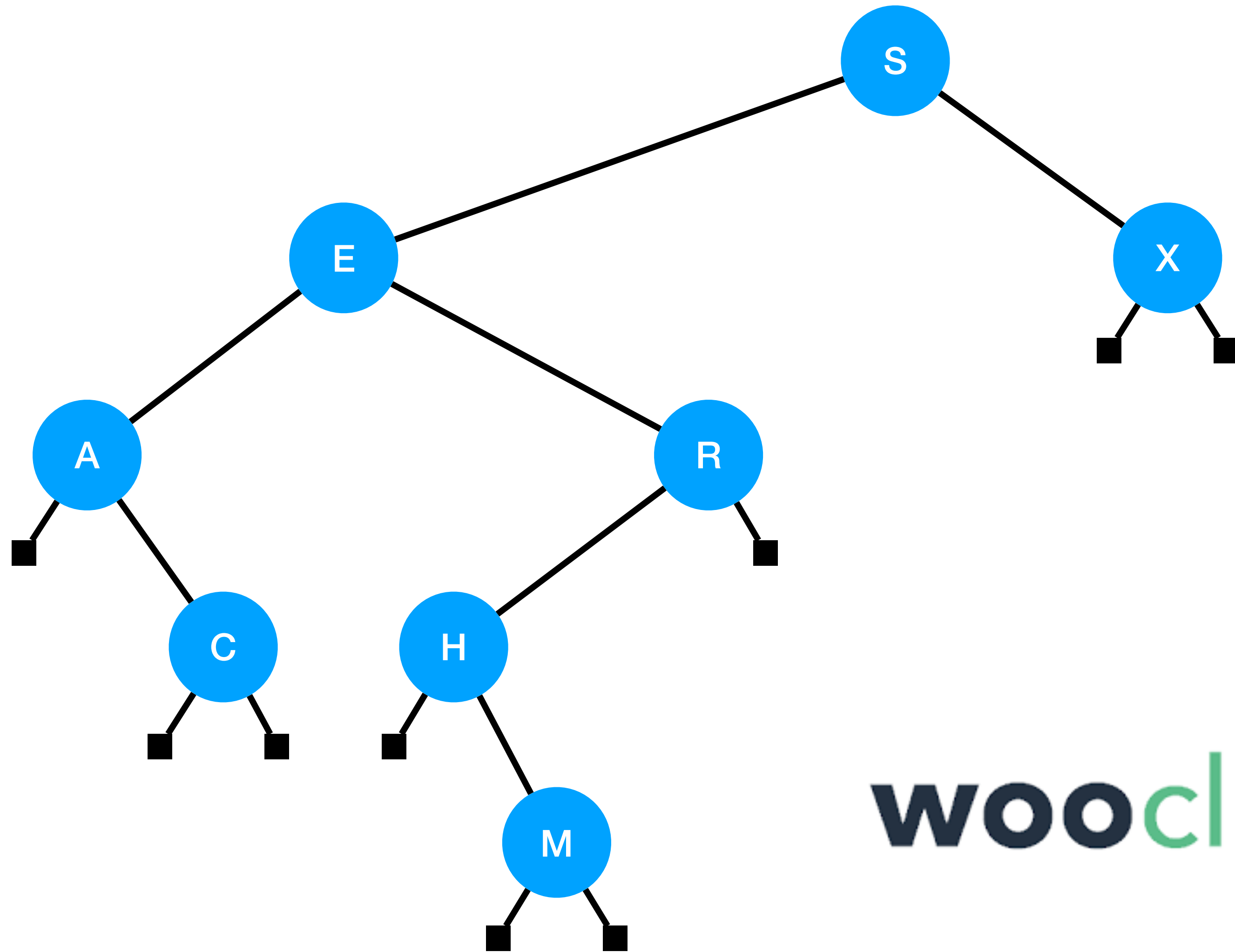
# Exemple

- Arbre de recherche non valide ?



# Exemple

- Arbre de recherche valide ?



wooclap



# Recherche dans un arbre de recherche

- Complexité

- $O(n)$  où  $n$  est le nombre de noeuds

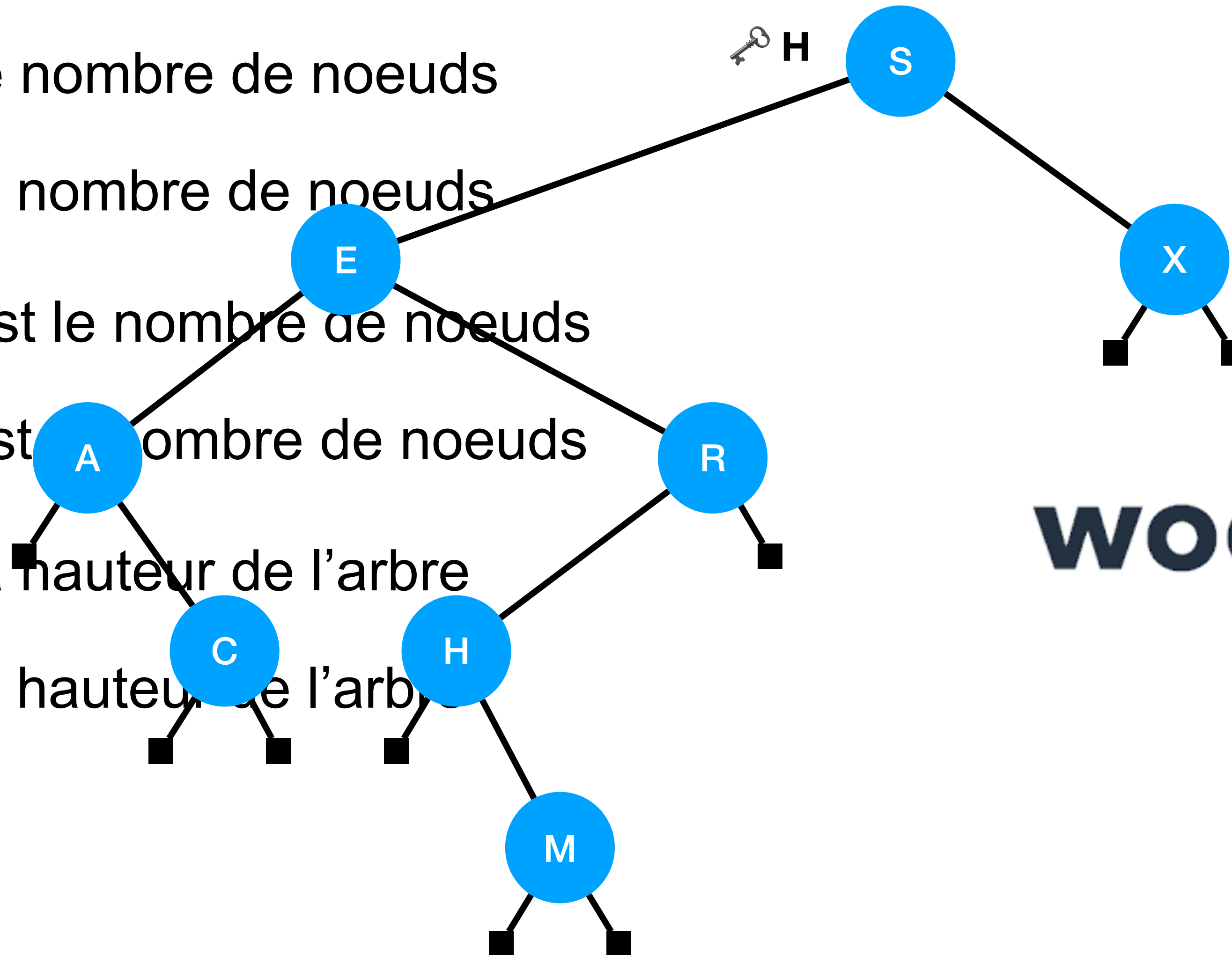
- $\Theta(n)$  où  $n$  est le nombre de noeuds

- $O(\log n)$  où  $n$  est le nombre de noeuds

- $\Theta(\log n)$  où  $n$  est le nombre de noeuds

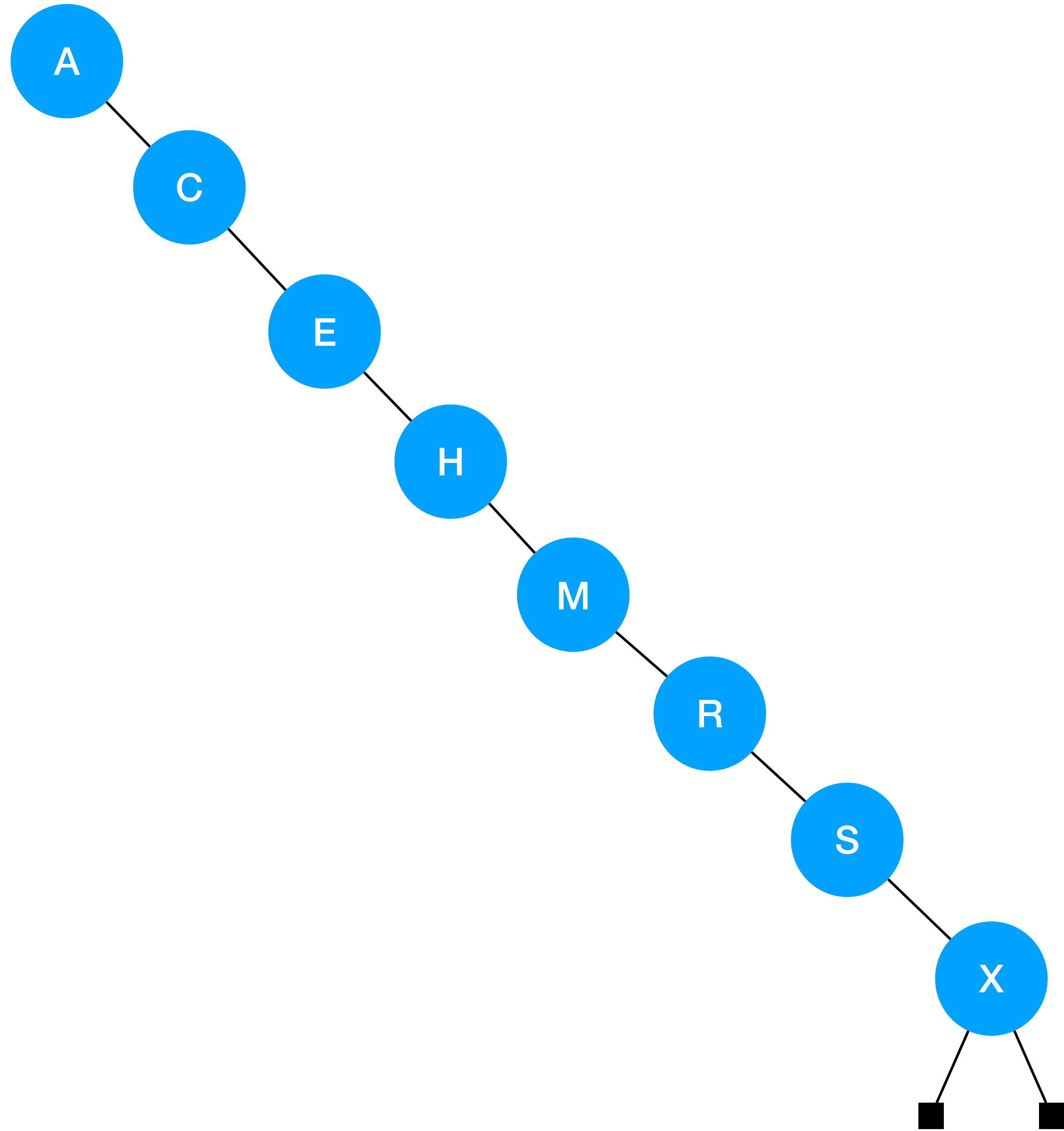
- $O(h)$  où  $h$  est la hauteur de l'arbre

- $\Theta(h)$  où  $h$  est la hauteur de l'arbre



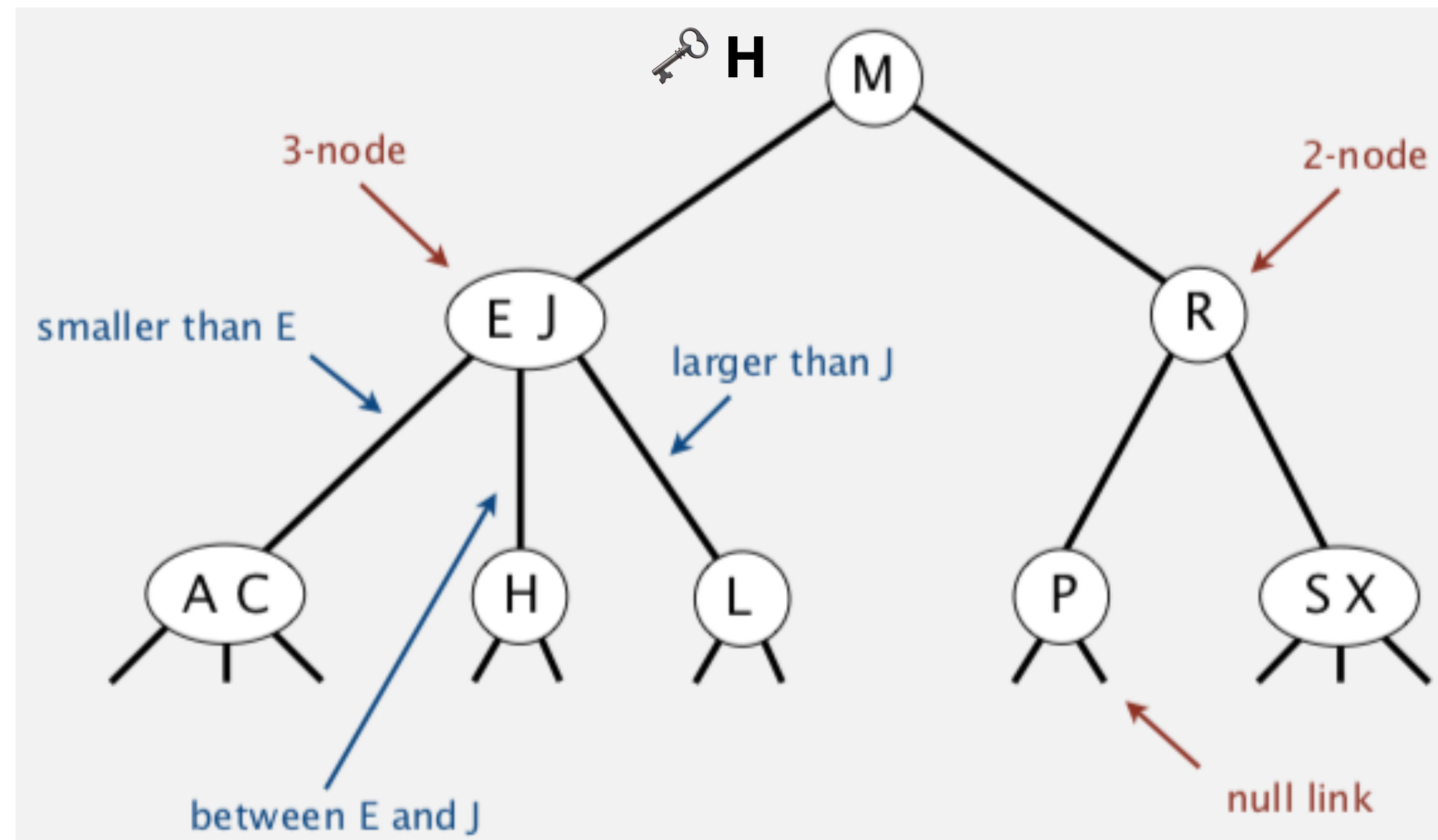
wooclap

# Le pire cas pour la hauteur



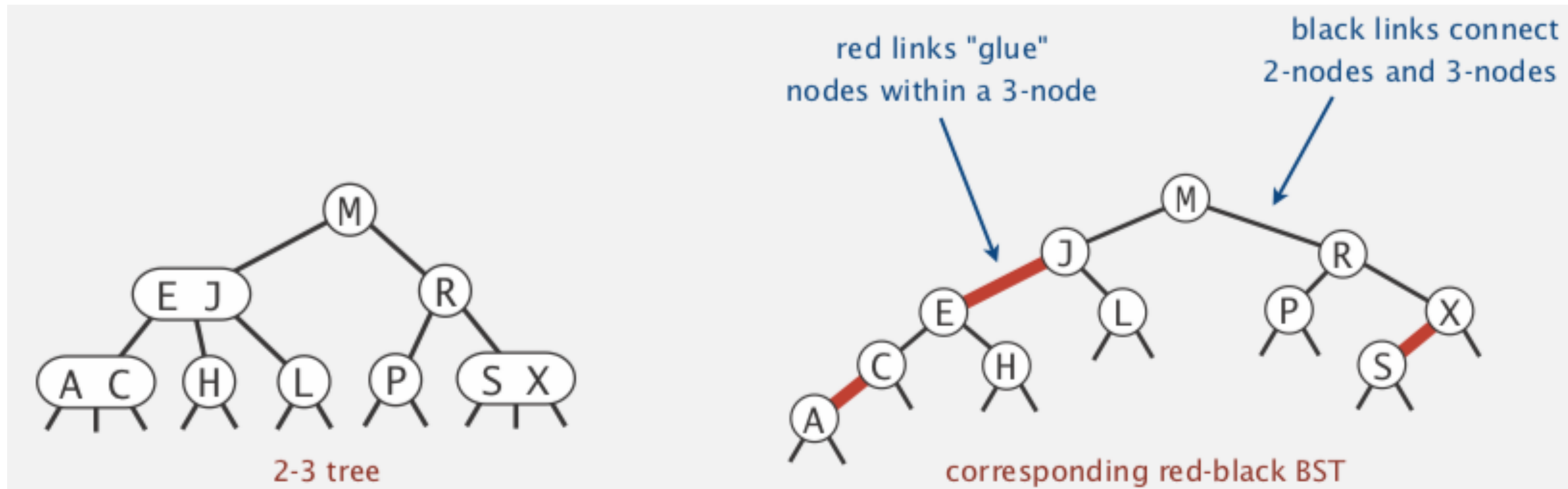
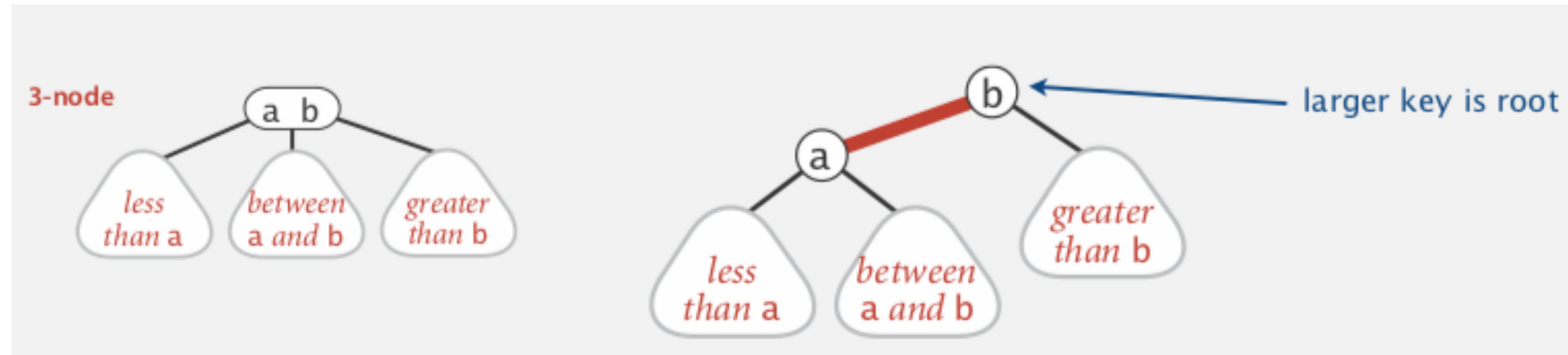
# Arbre équilibré $O(h) = O(\log(n))$

- Arbres 2-3, 1 ou 2 clefs par noeud
  - Equilibre parfait: Chaque chemin depuis la racine vers une feuille (lien null) a exactement la même longueur.
  - L'intérêt c'est que les complexités sont garanties  $O(\log(n))$ . Effet de la "liste chaînée" plus possible.



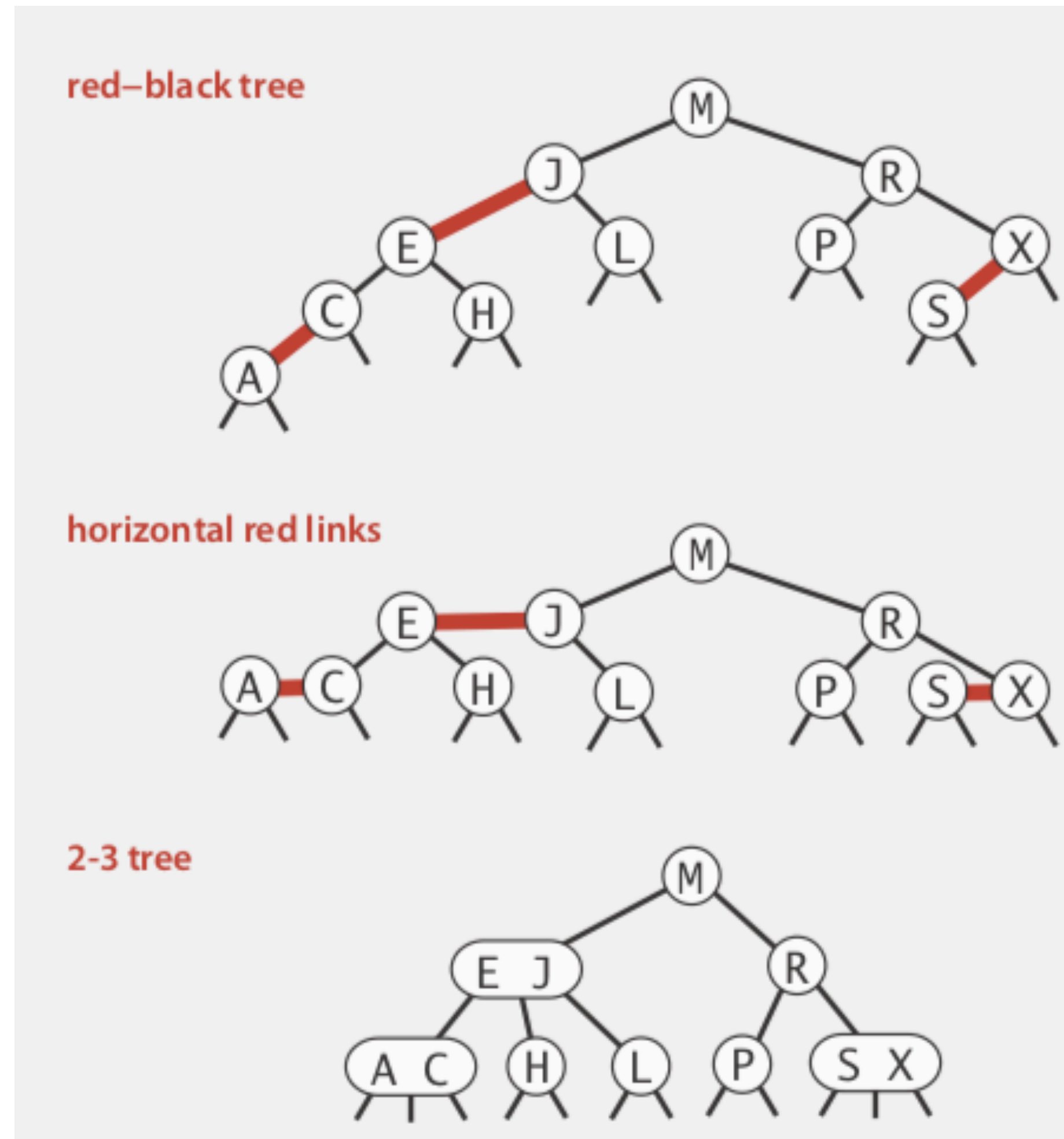
# Arbre équilibré binaire

- Arbre red-black penchant à droite
  - = représentation d'un arbre 2-3 mais avec un arbre binaire.



# Red-black vs 2-3

- La correspondance est parfaite
  - Il suffit de mettre les liens rouge à l'horizontal



# Résumé des complexité

implementation	guarantee		
	search	insert	delete
<b>sequential search (unordered list)</b>	$N$	$N$	$N$
<b>binary search (ordered array)</b>	$\lg N$	$N$	$N$
<b>BST</b>	$N$	$N$	$N$
<b>2-3 tree</b>	$c \lg N$	$c \lg N$	$c \lg N$
<b>red-black BST</b>	$2 \lg N$	$2 \lg N$	$2 \lg N$



# Dans les deux semaines qui viennent

- Lecture approfondie des chapitres 3.1, 3.2 et 3.3,
- Exercices théoriques pour maîtriser les arbres de recherche et les arbres de recherches équilibrés
- Exercices d'implémentation (part 1-3)
- Mid term test, ne compte dans la note finale que pour 2 points et uniquement s'il fait remonter la note.
  - Inginious le vendredi 3 novembre de 16h à 18h
  - Individuellement: **toute tentative de tricherie ou detection de plagiat sera sanctionnée d'un zero pour le cours.**

# Agenda Reminder

- Lecture next week (24h vélo mais on maintient)