Files de priorité

Christophe Damas
José Vander Meulen

File de priorité

```
public interface PriorityQueue{
    void insert (Comparable v);
    Comparable max();
    Comparable delMax();
    boolean isEmpty();
    int size();
}
```

Implémentation

- Pour implémenter les files de priorité, on pourrait utiliser certaines des structures vues jusqu'à présent.

complexité	Liste non triée	Liste triée
insert		
delMax		

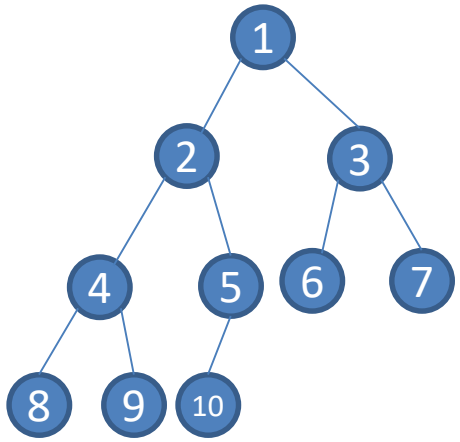
Implémentation: Heap

- Il existe une implémentation où les opérations se font en O(log(n)).
  - Donc, l'insertion et le traitement de n éléments se feront en O(n*log(n)).

compléxité	Liste non triée	Liste triée	Tas
insert	O(1)	O(n)	O(log(n))
delMax	O(n)	O(1)	O(log(n))

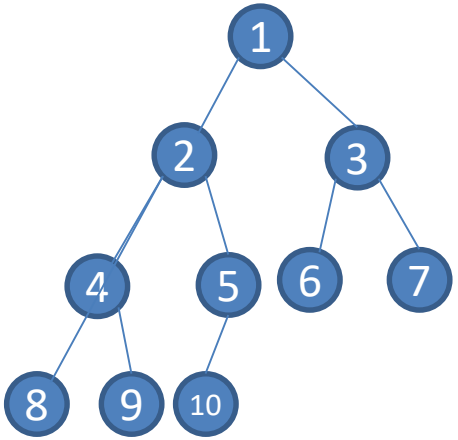
Arbre binaire complet

- Arbre dont toutes les feuilles sont au même niveau ou sur deux niveaux adjacents et si toutes les feuilles situées au niveau le plus bas sont le plus à gauche possible.



Implémentation arbre binaire complet

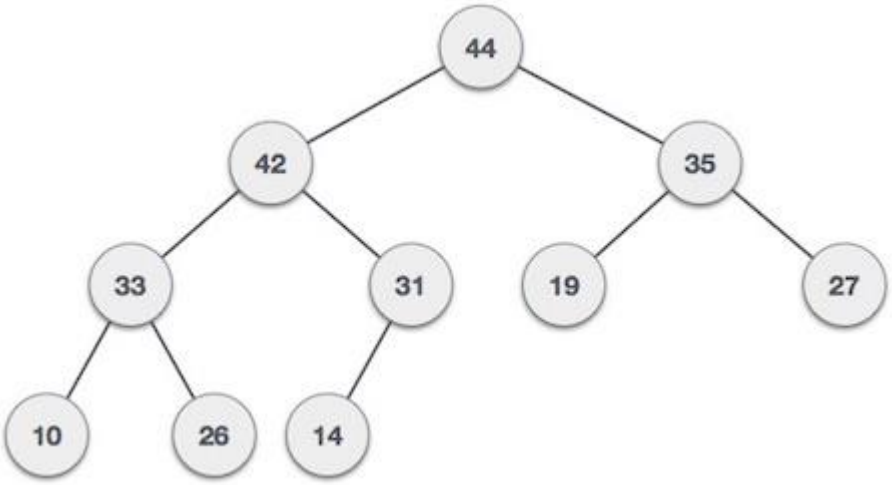
- Il est facile d'implémenter un arbre binaire complet à l'aide d'un tableau.
  - La racine est en position 1
  - Les enfants d’un nœud sont en position 2k et 2k+1



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

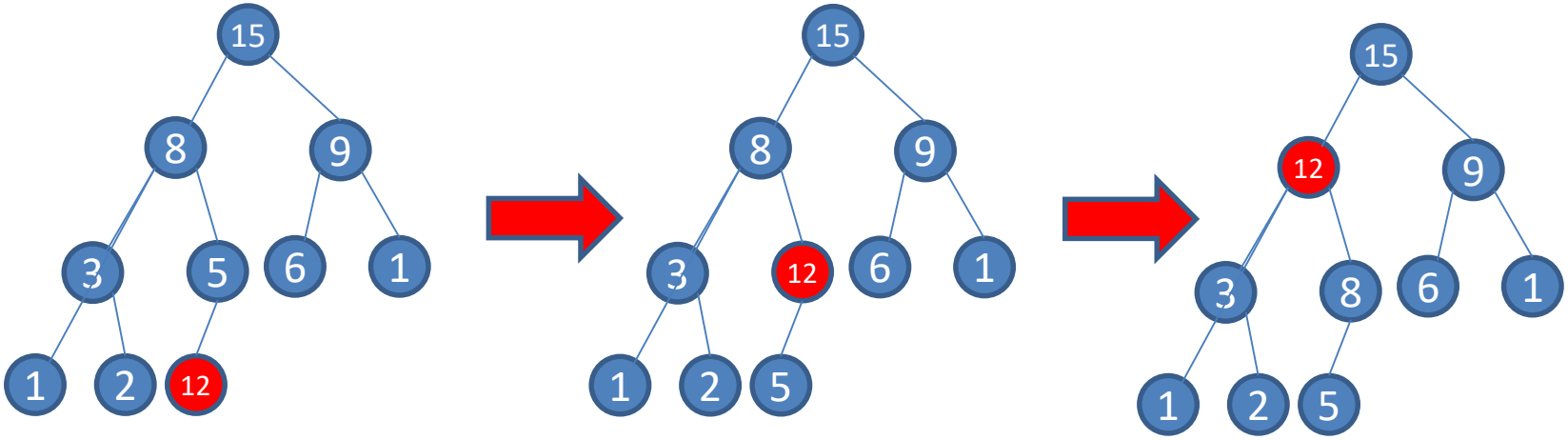
Tas

- Un tas est un arbre binaire complet pour lequel la valeur de chaque noeud est supérieure ou égale à celle de ses fils



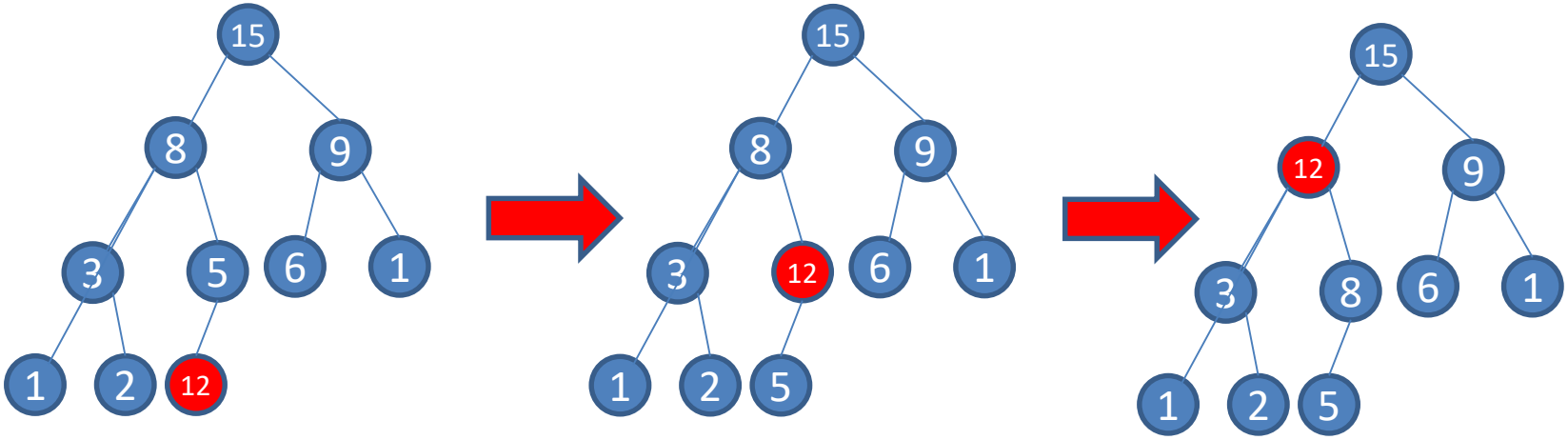
Algorithme sur les tas: swim

- Si la propriété du tas est violé car un noeud est plus grand que son parent, on peut régler le problème en échangeant ce noeud avec son parent



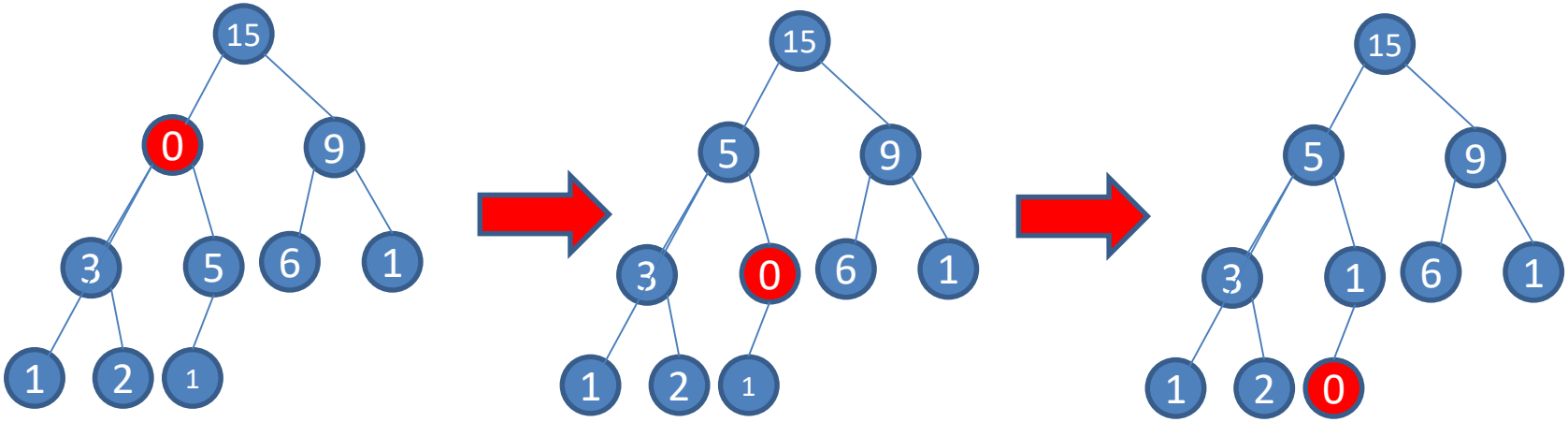
swim: pseudo-code

```
private void swim(int k){
      while (k>1 && less(k/2,k)){
            exchange(k/2,k);
            k=k/2;
      }
}
```



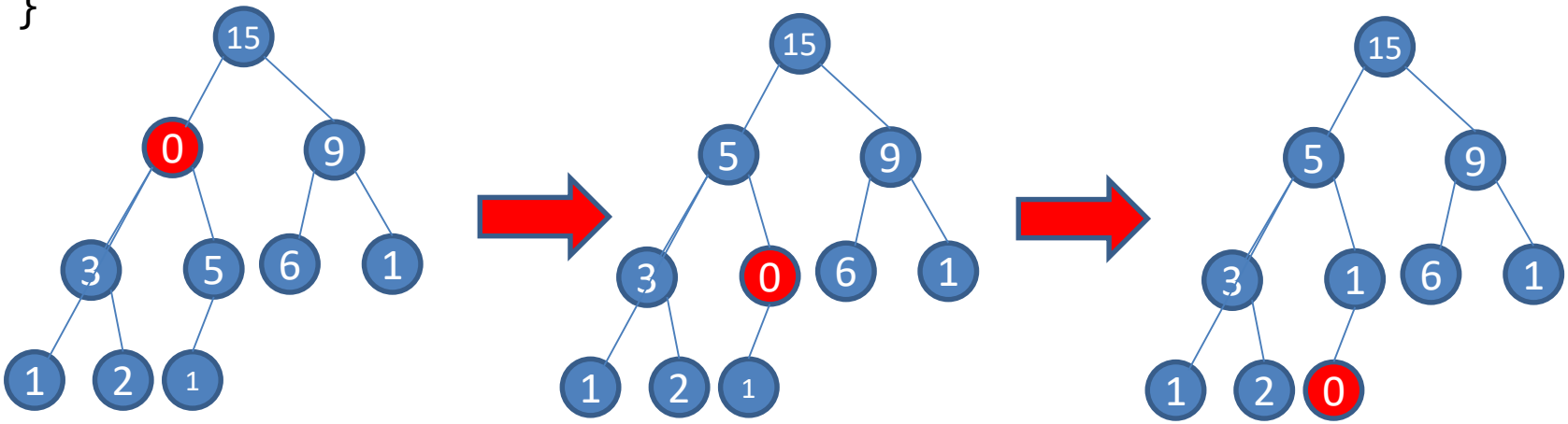
Algorithme sur les tas: sink

- Si la propriété du tas est violé car un noeud est plus petit que ses enfants, on peut régler le problème en échangeant ce noeud avec le plus grand de ses enfants



sink: pseudo-code

```
private void sink(int k){
        while (2*k<=N){
                int j=2*k;
                if(j<N && less(j,j+1)) j++:
                if (!less(k,j)) break;
                exchange(k,j);
                k=j;
        }
}
```

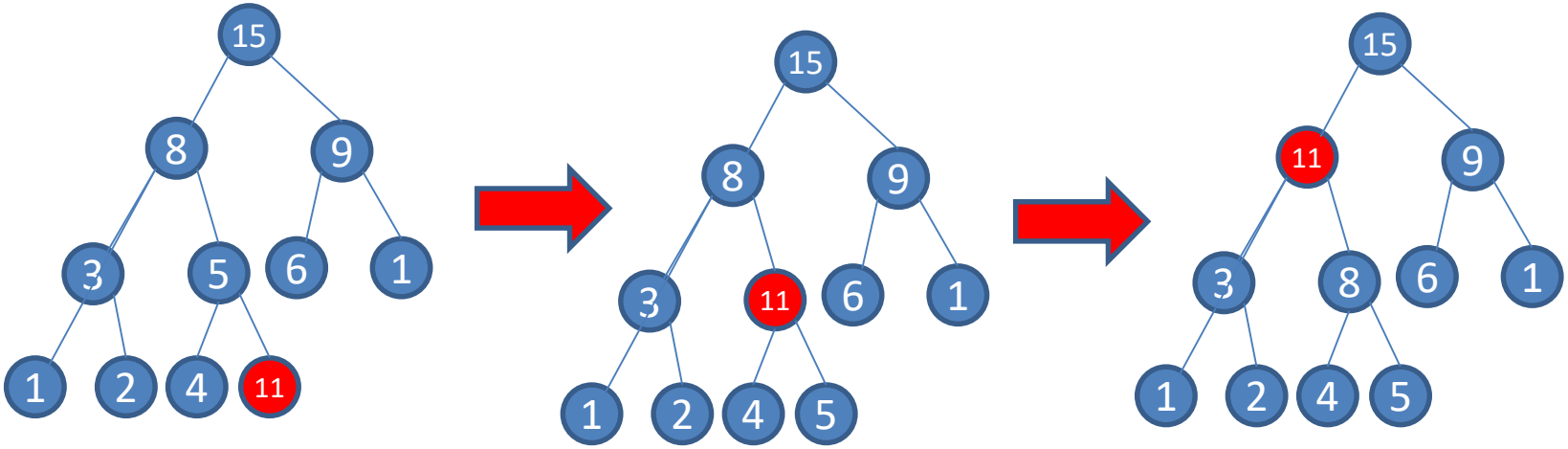


Retour aux méthodes

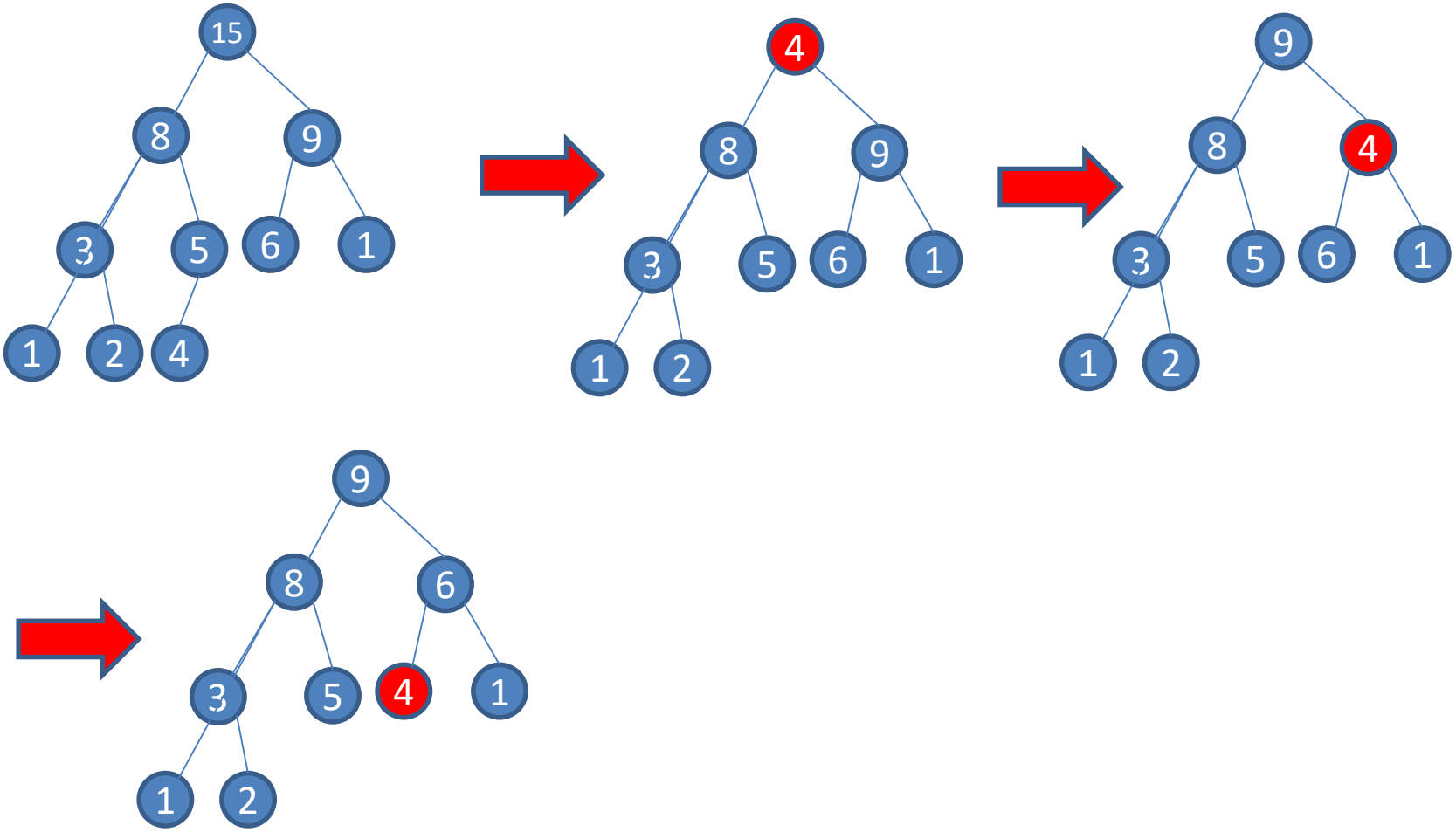
```
public interface PriorityQueue{
	void insert (Comparable v);
	Comparable max();
	Comparable delMax();
	boolean isEmpty();
	int size();
}
```

- insert: on ajoute la valeur à la fin du tableau, on incrémente la taille du tas et on exécute swim sur ce noeud pour rétablir la propriété du tas
- delMax: On enlève l'élément maximum (la racine) et le remplace par le dernier élément du tas sur lequel on exécute sink pour rétablir la propriété du tas

Insertion de la valeur 11



Suppression du maximum



Complexité

- Les 2 opérations en O(log(n)) car l'arbre binaire est complet

Heap Sort

- Si on insère n éléments dans une file de priorité, puis qu'on supprime le maximum, la dernière case du tableau devient libre. On peut donc y stocker le maximum. En continuant de la sorte, le tableau sera trié.
- Le gros avantage de Heapsort est d'être en O(n*log(n)) même dans le pire des cas (contrairement à Quick Sort qui est $O(n^2)$).
- Heapsort utilise une technique légèrement modifiée pour construire le tas initial contenant tous les éléments du tableau. Elle utilise uniquement le sink().

En java

- class PriorityQueue
- Voir Javadoc

Exercices

- Sur papier
  - Insérez les éléments suivants dans un tas dans cet ordre: 57 85 44 21 23 52 17 7 95
  - Donnez le tableau représentant ce tas après ces 9 insertions
  - Enlevez deux fois le maximum et représenter le tableau après ces opérations
- Sur machine
  - Implémentez les méthodes insert et delMax()