INF2010 – ASD

Monceaux

Plan

- Définition de monceau et implémentation
- Insertion et retrait d'éléments
- Construction d'un monceux
- Tri par monceau

Plan

- Définition de monceau et implémentation
- Insertion et retrait d'éléments
- Construction d'un monceux
- Tri par monceau

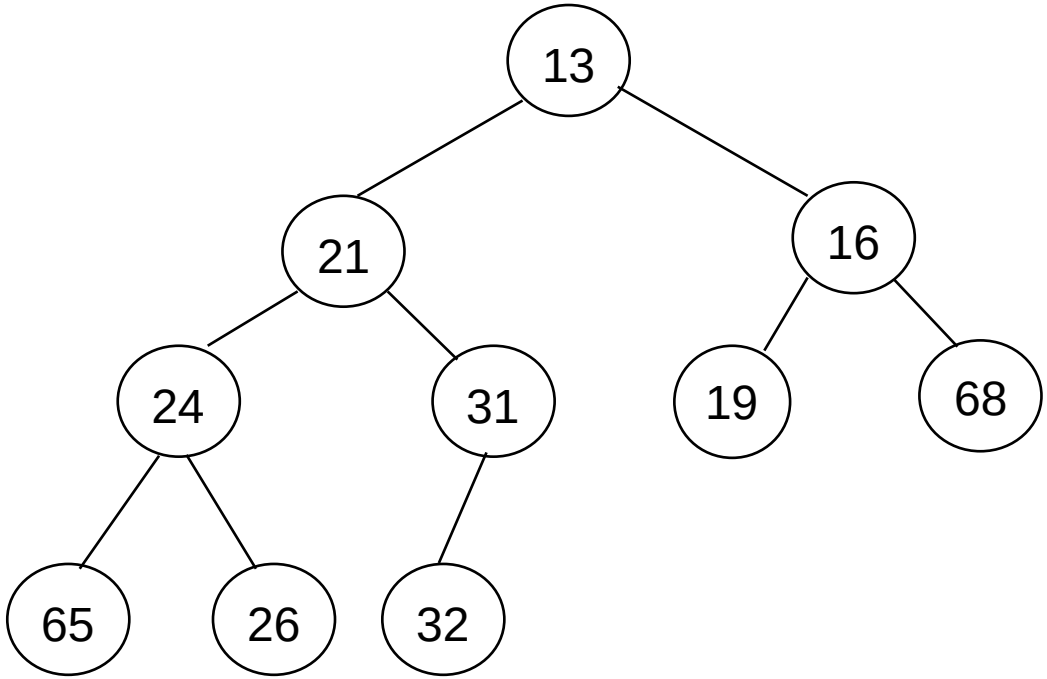
Monceau - définition

- Un arbre complet tel que pour chaque noeud N dont le parent est P, la clé de P est plus petite que celle de N
- Normalement implémenté dans un tableau
- Insertion et retrait sont O(lg N) dans le pire cas
- Recherche du plus petit élément est O(1) dans le pire cas

Monceau – implémentation

- On utilise un tableau pour implémenter l'arbre
- On met la racine de l'arbre à la position 1, plutôt que la position 0, ce qui facilite les opérations
- Pour chaque nœud i, les indices sont donc 2*i pour le fils gauche et 2*i + 1 pour le fils droit

Monceau - exemple

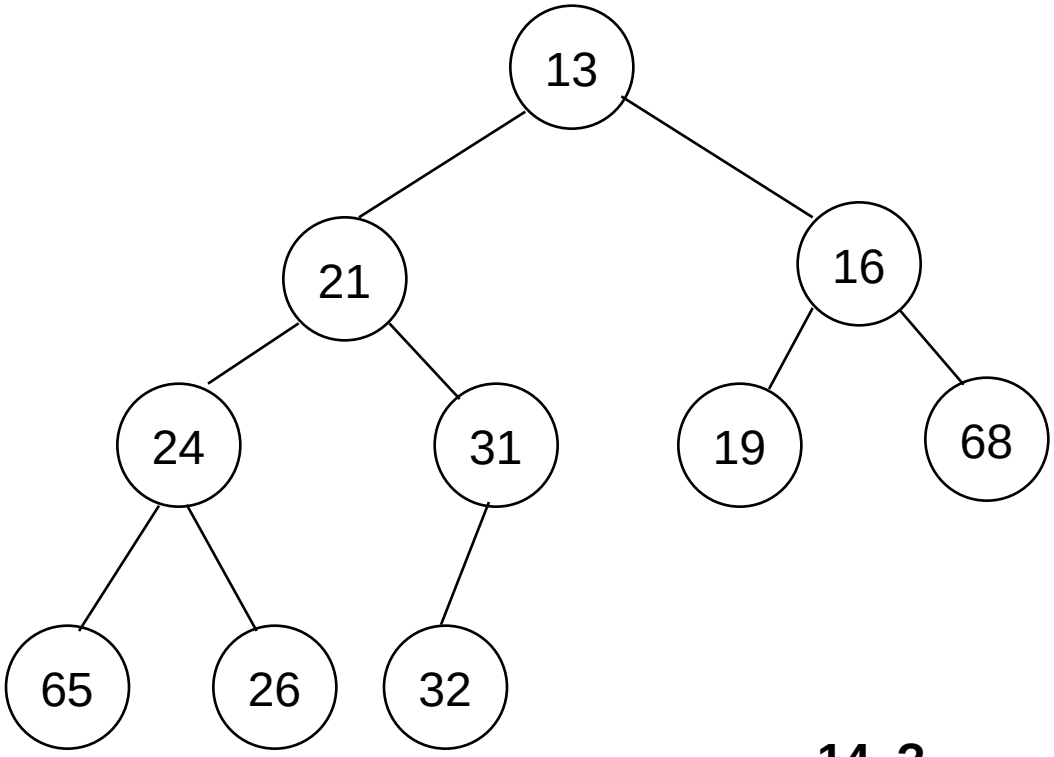


0	1	2	3	4	5	6	7	8	9	10	11
	13	21	16	24	31	19	68	65	26	32	

Plan

- Définition de monceau et implémentation
- Insertion et retrait d'éléments
- Construction d'un monceux
- Tri par monceau

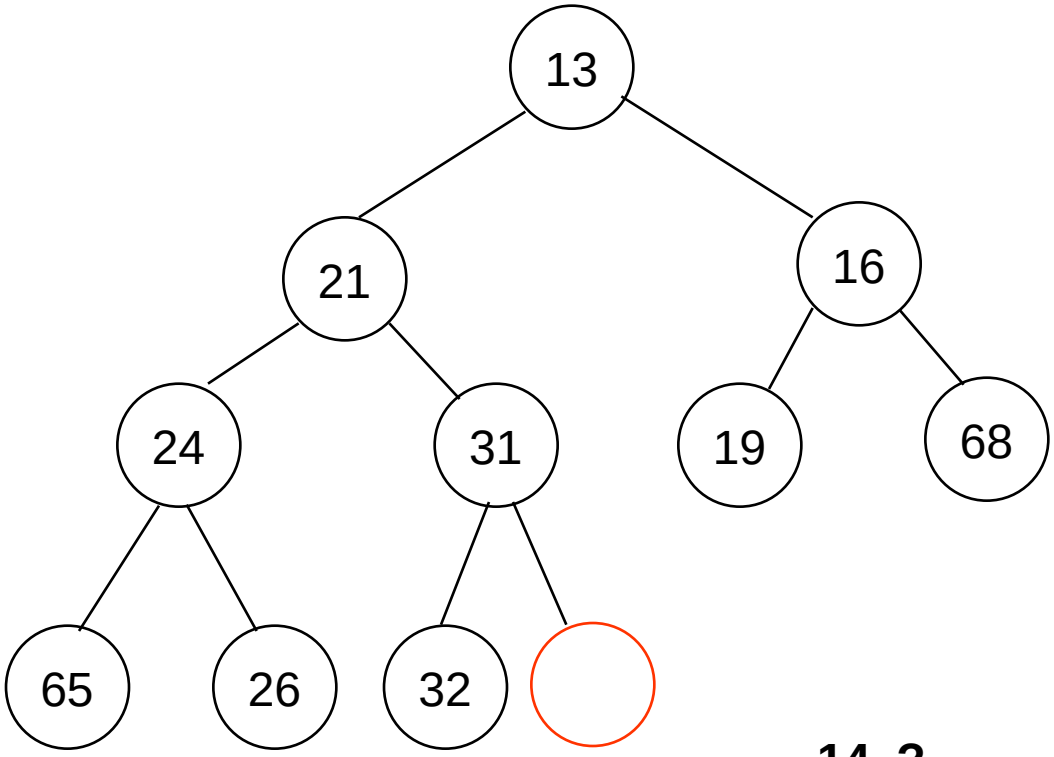
Monceau – insertion de 14



14 ?

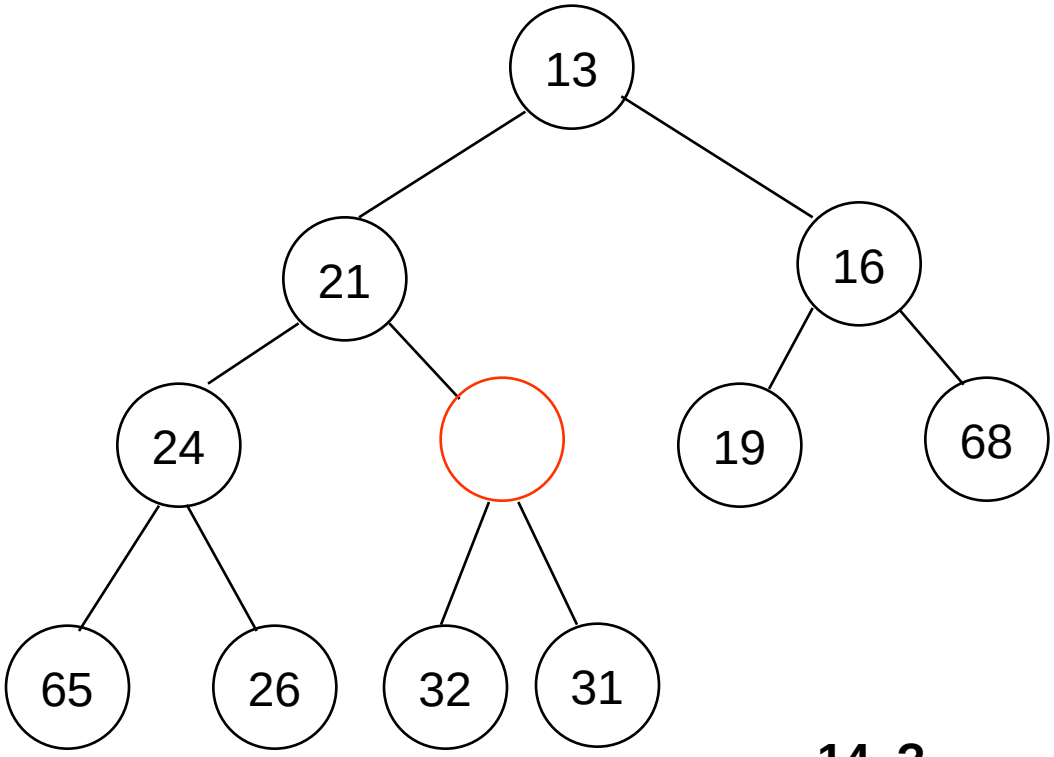
0	1	2	3	4	5	6	7	8	9	10	11
	13	21	16	24	31	19	68	65	26	32	

Monceau - insertion de 14



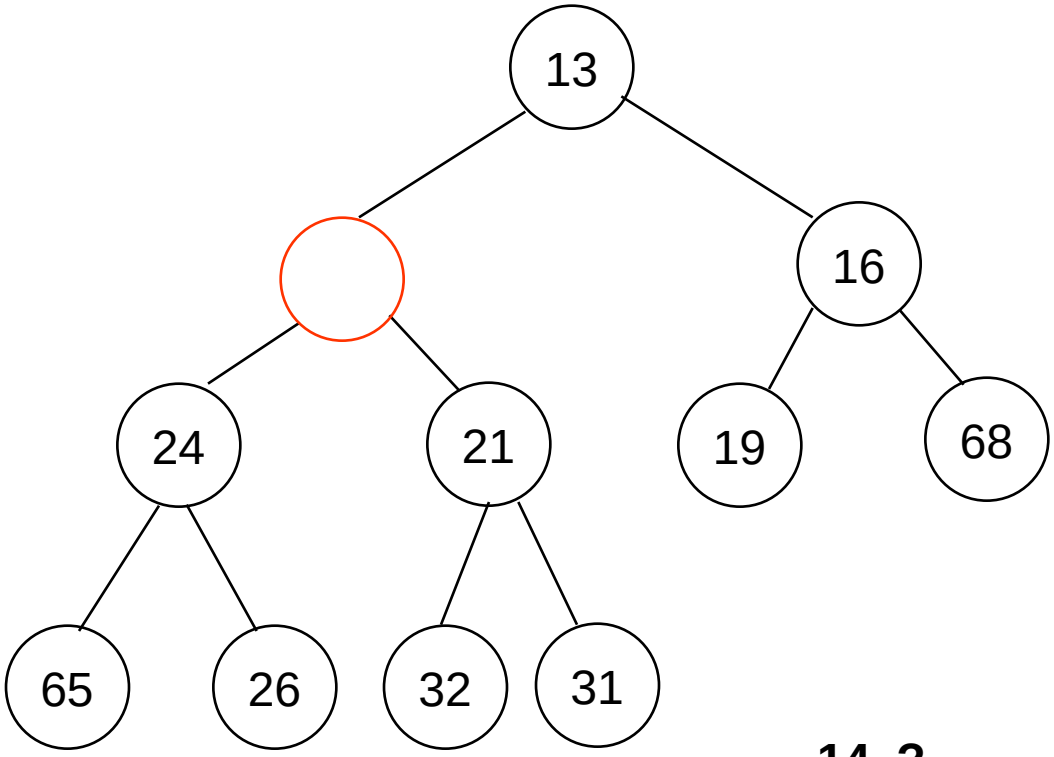
0	1	2	3	4	5	6	7	8	9	10	11
	13	21	16	24	31	19	68	65	26	32	

Monceau - insertion de 14



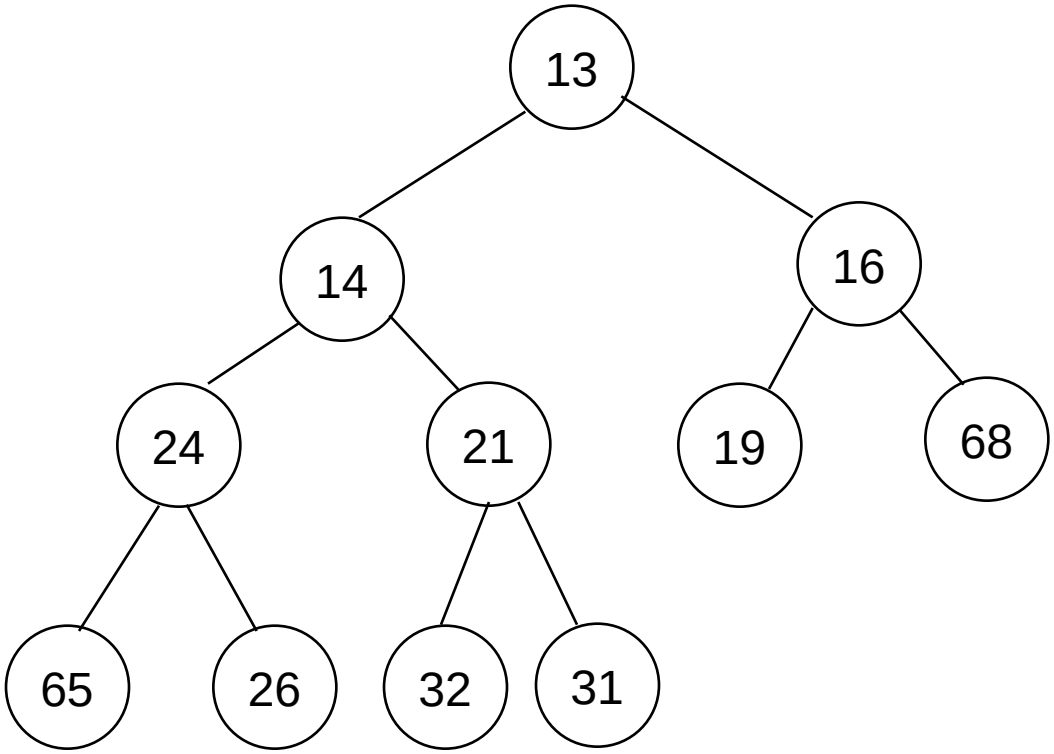
0	1	2	3	4	5	6	7	8	9	10	11
	13	21	16	24		19	68	65	26	32	31

Monceau - insertion de 14



0	1	2	3	4	5	6	7	8	9	10	11
	13		16	24	21	19	68	65	26	32	31

Monceau - insertion de 14

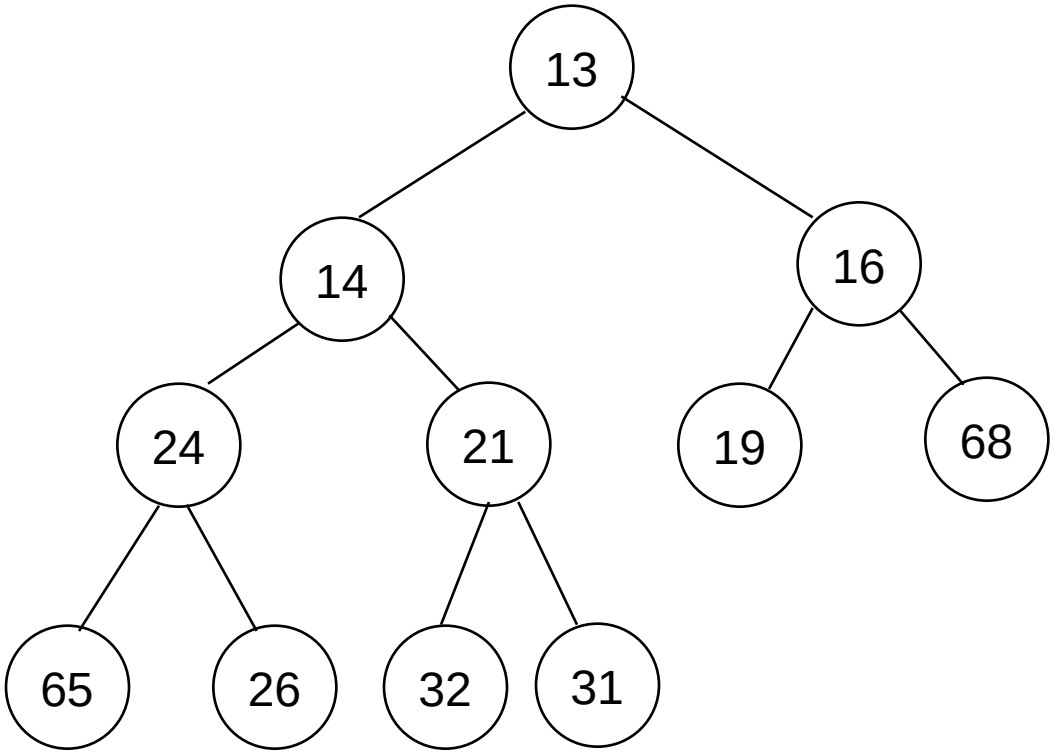


0	1	2	3	4	5	6	7	8	9	10	11
	13	14	16	24	21	19	68	65	26	32	31

Insertion

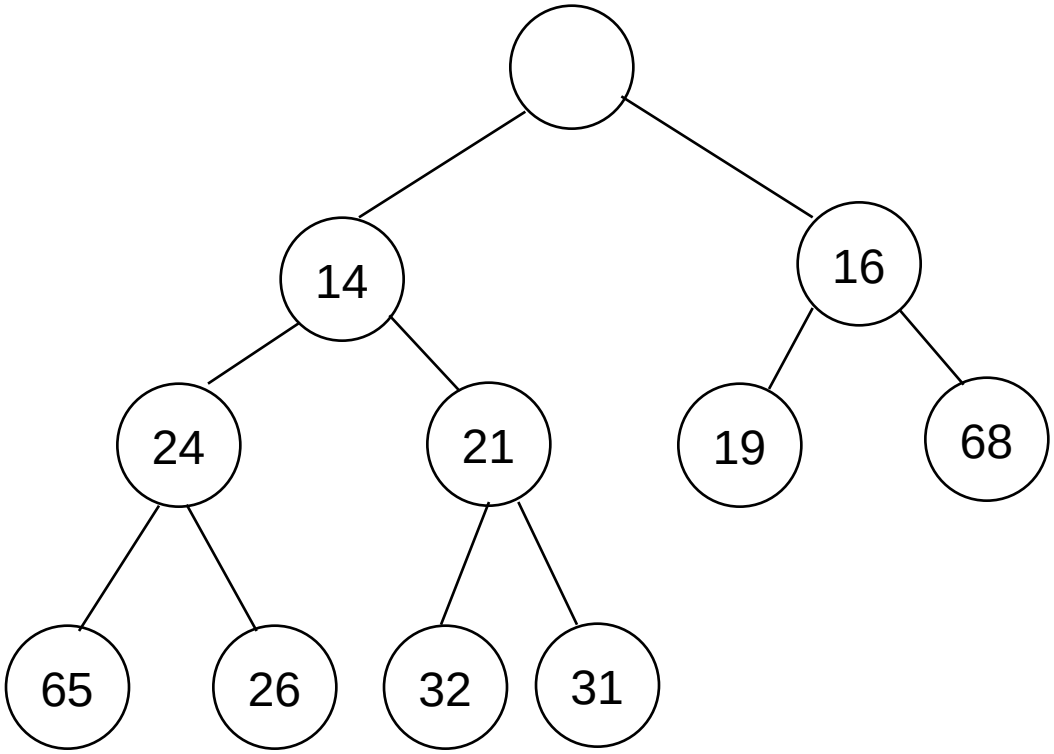
```
/**
  * Insert into the priority queue, maintaining heap order.
  * Duplicates are allowed.
  * @param x the item to insert.
  */
 public void insert( AnyType x )
 {
     if( currentSize == array.length - 1 )
         enlargeArray( array.length * 2 + 1 );

     // Percolate up
     int hole = ++currentSize;
     for( ; hole > 1 && x.compareTo( array[ hole / 2 ] ) < 0; hole /= 2)
         array[ hole ] = array[ hole / 2 ];
     array[ hole ] = x;
 }
```

Monceau - retrait



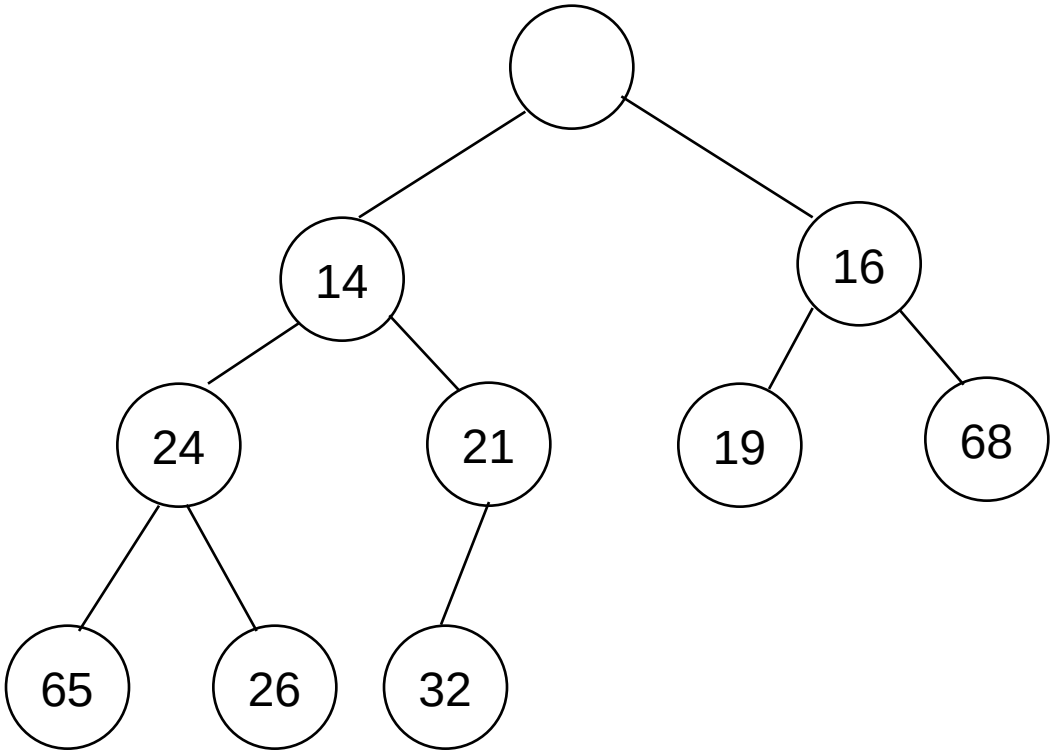
0	1	2	3	4	5	6	7	8	9	10	11
	13	14	16	24	21	19	68	65	26	32	31

Monceau - retrait



0	1	2	3	4	5	6	7	8	9	10	11
		14	16	24	21	19	68	65	26	32	31

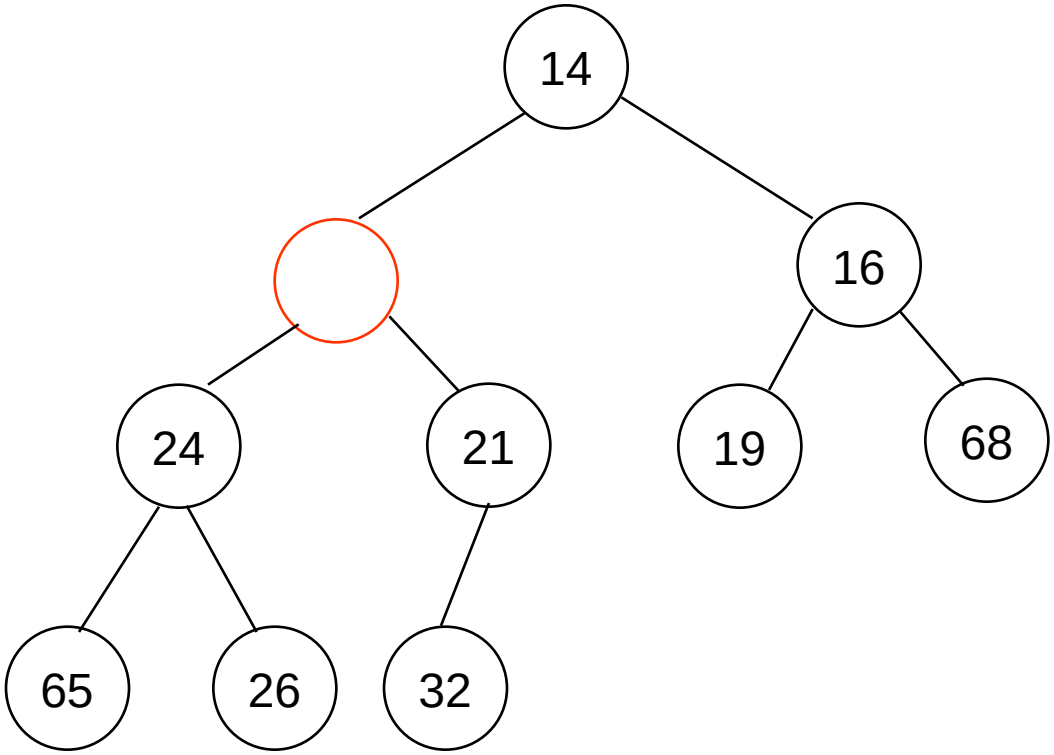
Monceau - retrait



31 ?

0	1	2	3	4	5	6	7	8	9	10	11
		14	16	24	21	19	68	65	26	32	

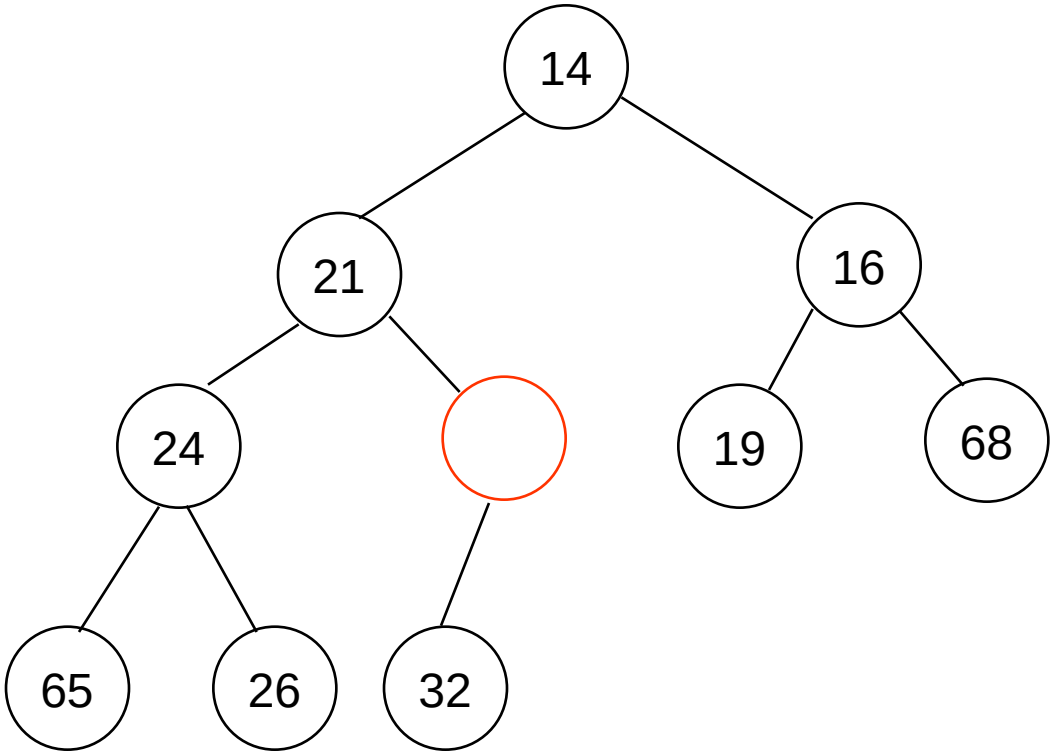
Monceau - retrait



31 ?

0	1	2	3	4	5	6	7	8	9	10	11
	14		16	24	21	19	68	65	26	32	

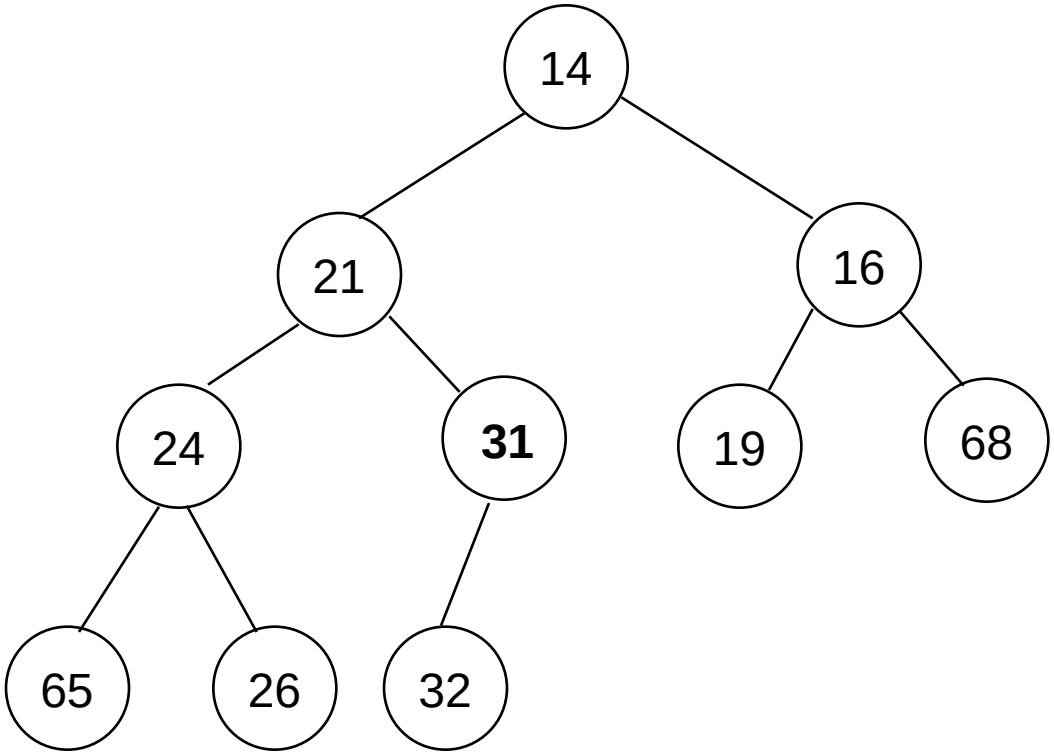
Monceau - retrait



31 ?

0	1	2	3	4	5	6	7	8	9	10	11
	14	21	16	24		19	68	65	26	32	

Monceau - retrait



0	1	2	3	4	5	6	7	8	9	10	11
	14	21	16	24	31	19	68	65	26	32	

Monceau - retrait

```
/**
  * Remove the smallest item from the priority queue.
  * @return the smallest item, or throw UnderflowException, if empty.
  */
 public AnyType deleteMin( )
 {
     if( isEmpty( ) )
         throw new UnderflowException( );

     AnyType minItem = findMin( );
     array[ 1 ] = array[ currentSize-- ];
     percolateDown( 1 );

     return minItem;
 }
```

Monceau - retrait

```
/**
  * Internal method to percolate down in the heap.
  * @param hole the index at which the percolate begins.
  */
 private void percolateDown( int hole )
 {
     int child;
     AnyType tmp = array[ hole ];

     for( ; hole * 2 <= currentSize; hole = child )
     {
         child = hole * 2; //Considérer fils de gauche

         if( child != currentSize && // il y a deux fils
         array[ child + 1 ].compareTo( array[ child ] ) < 0 ) //et fils droit<fils gauche
             child++; //Considérer fils droit
         if( array[ child ].compareTo( tmp ) < 0 )//fils considéré< élément à percoler
             array[ hole ] = array[ child ];//Remonter le fils courrent de un niveau
         else
             break; //sortir de la boucle. L'élément à percoler sera inséré à position hole
     }

     array[ hole ] = tmp; // Insérer l'élément à percoler à la position hole

 }
```

Monceau – insertion et retrait

- La complexité en pire cas d'un retrait et d'une insertion est *O*(log(n))

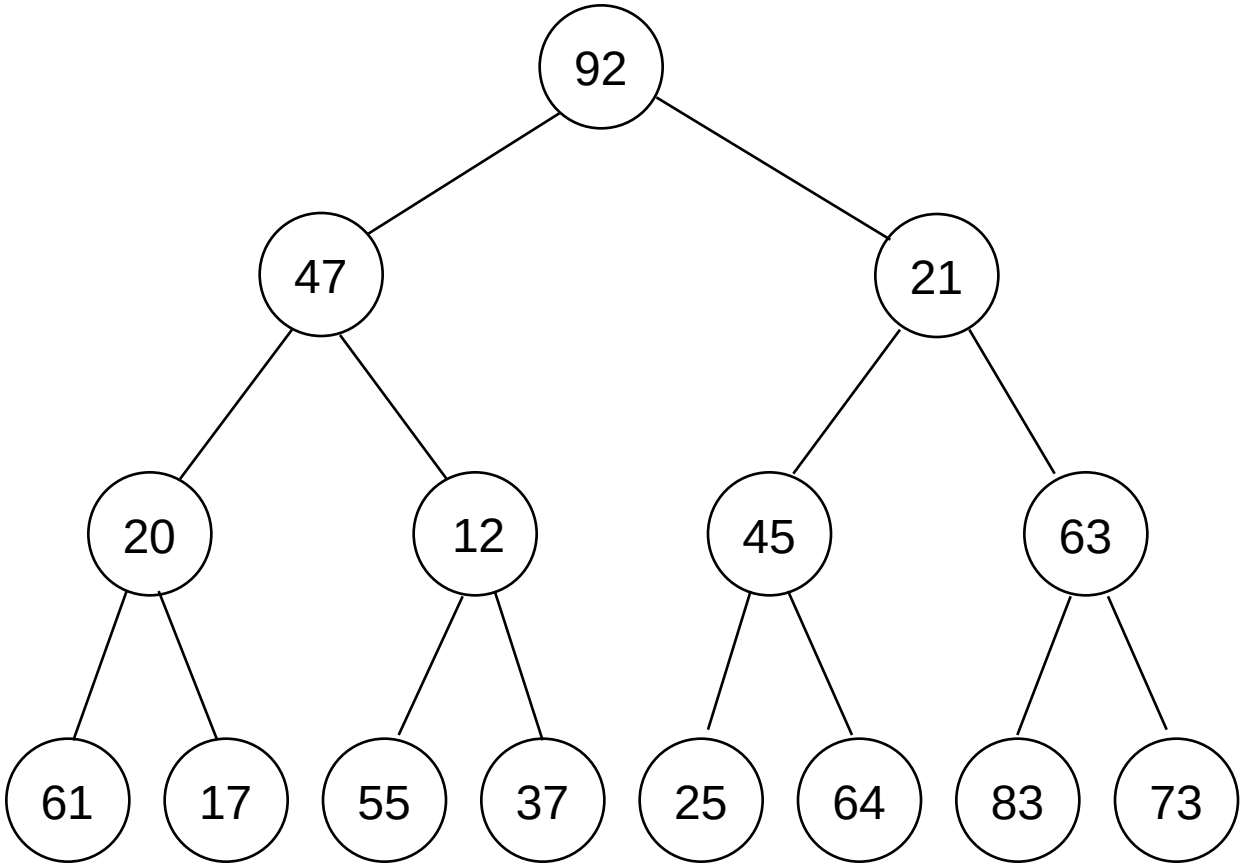
- *La complexité* en meilleur cas d'un retrait et d'une insertion est *O*(1)

- La complexité moyenne d'un retrait : *O*(log(n))
- La complexité moyenne d'une insertion est : O(1)

Plan

- Définition de monceau et implémentation
- Insertion et retrait d'éléments
- Construction d'un monceux
- Tri par monceau

Monceau - construction

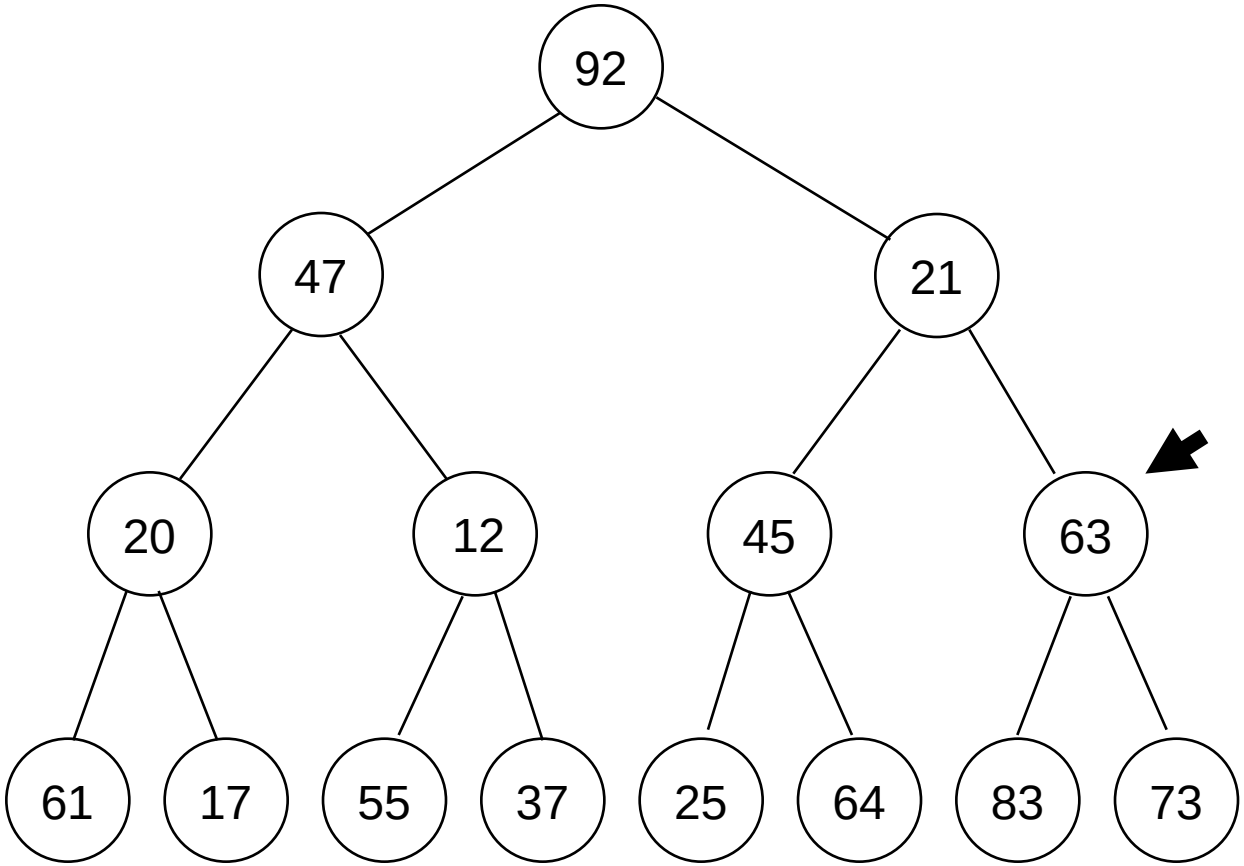


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	45	63	61	17	55	37	25	64	83	73

Construction

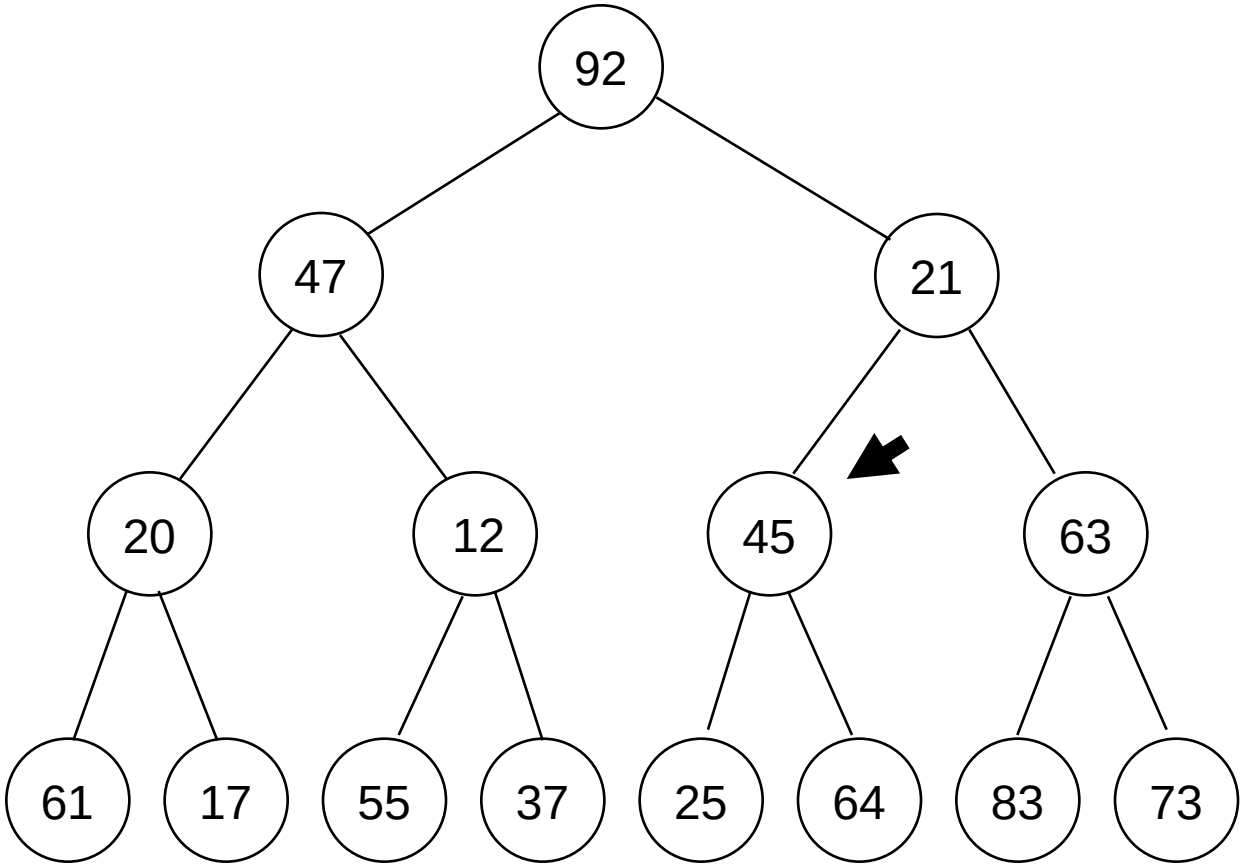
```
/**
  * Establish heap order property from an arbitrary
  * arrangement of items. Runs in linear time.
  */
 private void buildHeap( )
 {
     for( int i = currentSize / 2; i > 0; i-- )
         percolateDown( i );
 }
```

Monceau - construction



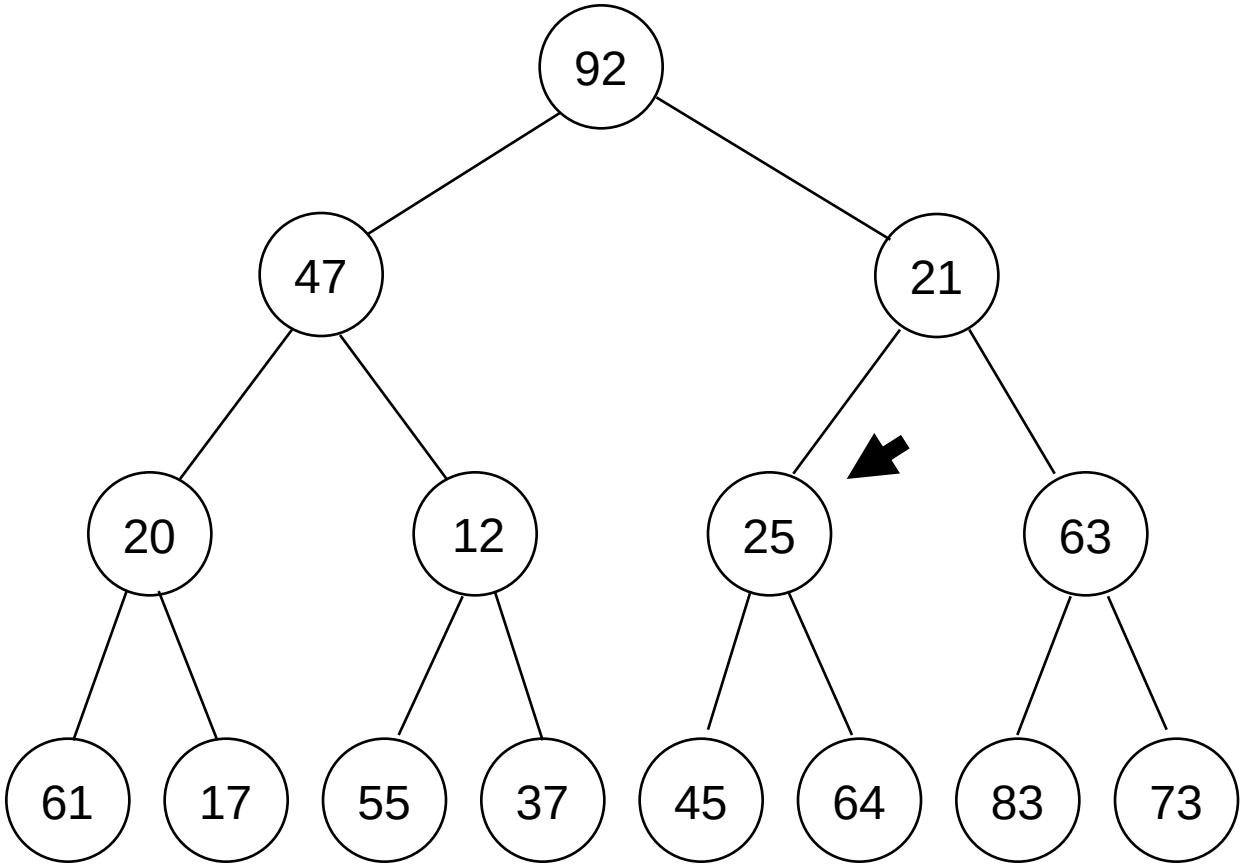
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	45	63	61	17	55	37	25	64	83	73

Monceau - construction



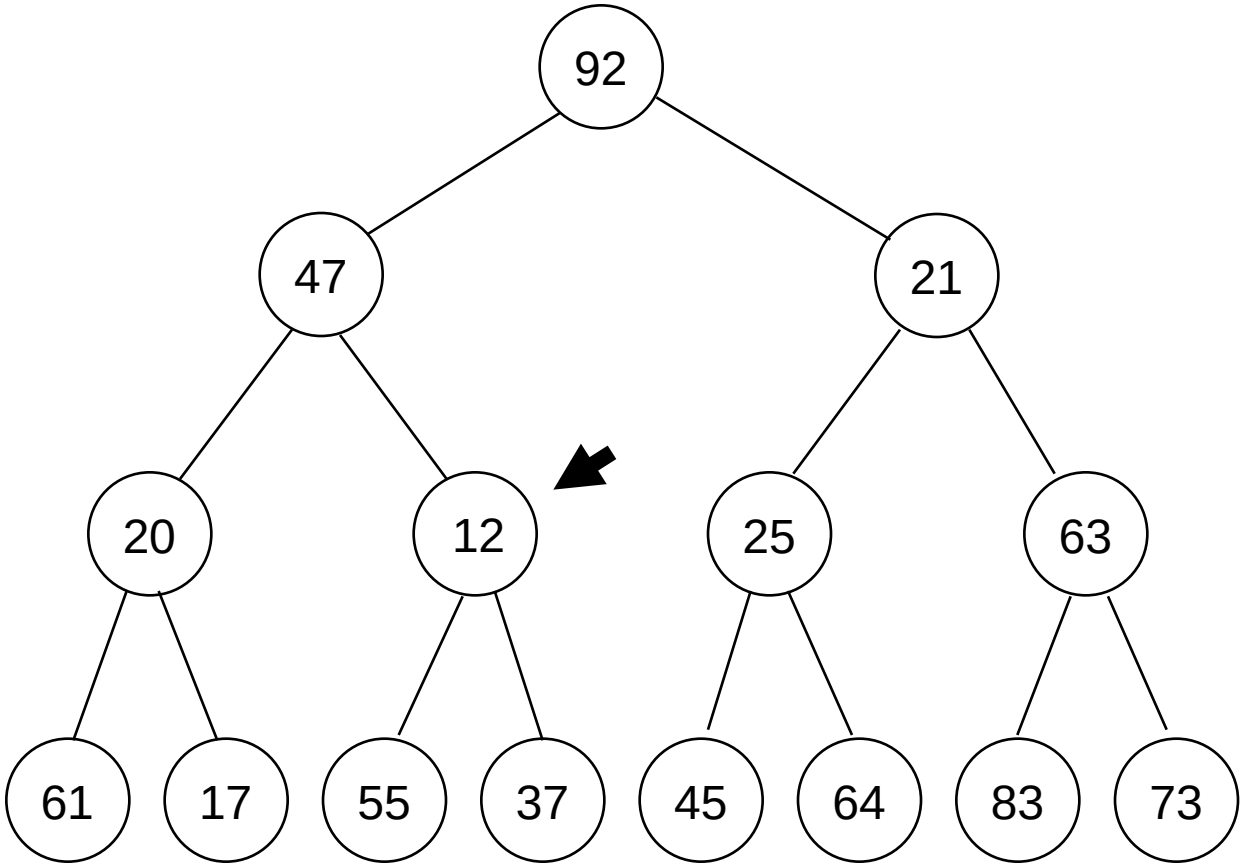
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	45	63	61	17	55	37	25	64	83	73

Monceau - construction



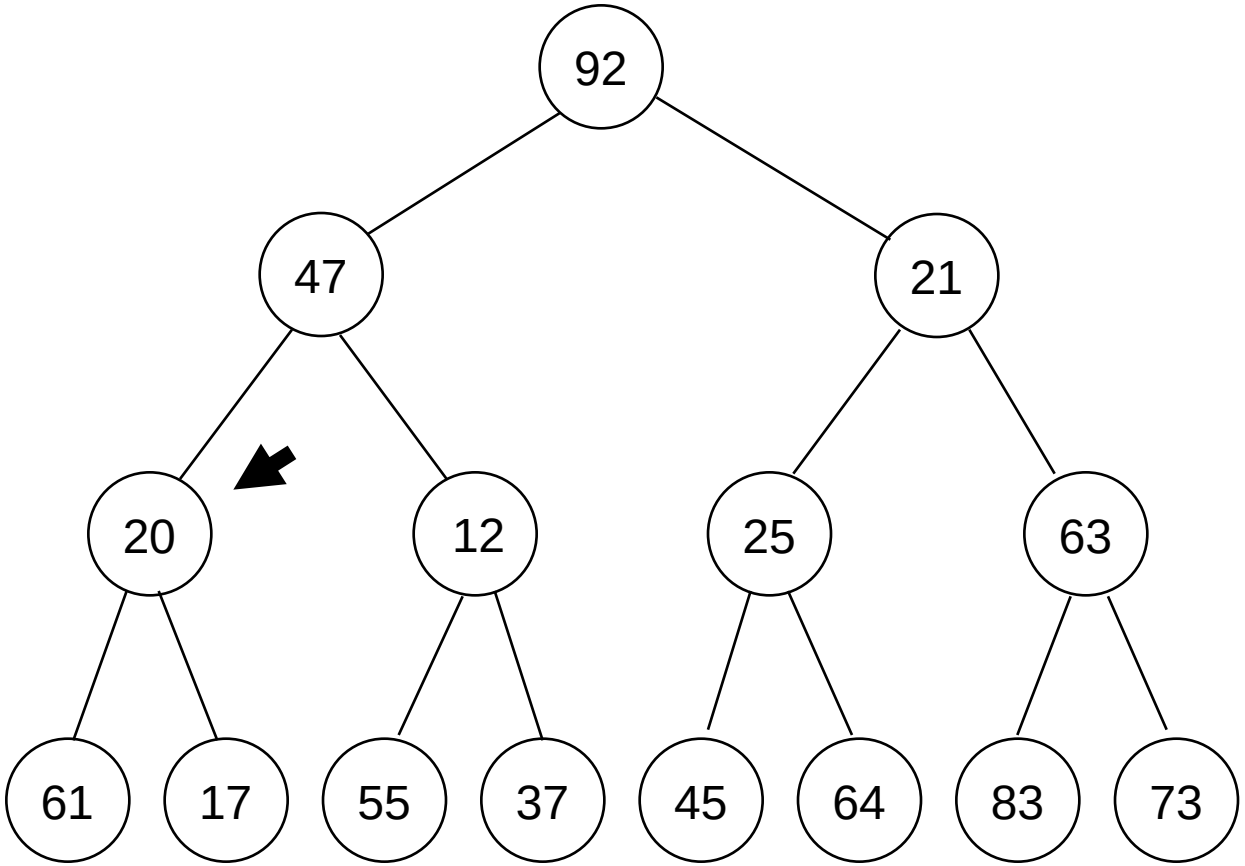
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	25	63	61	17	55	37	45	64	83	73

Monceau - construction



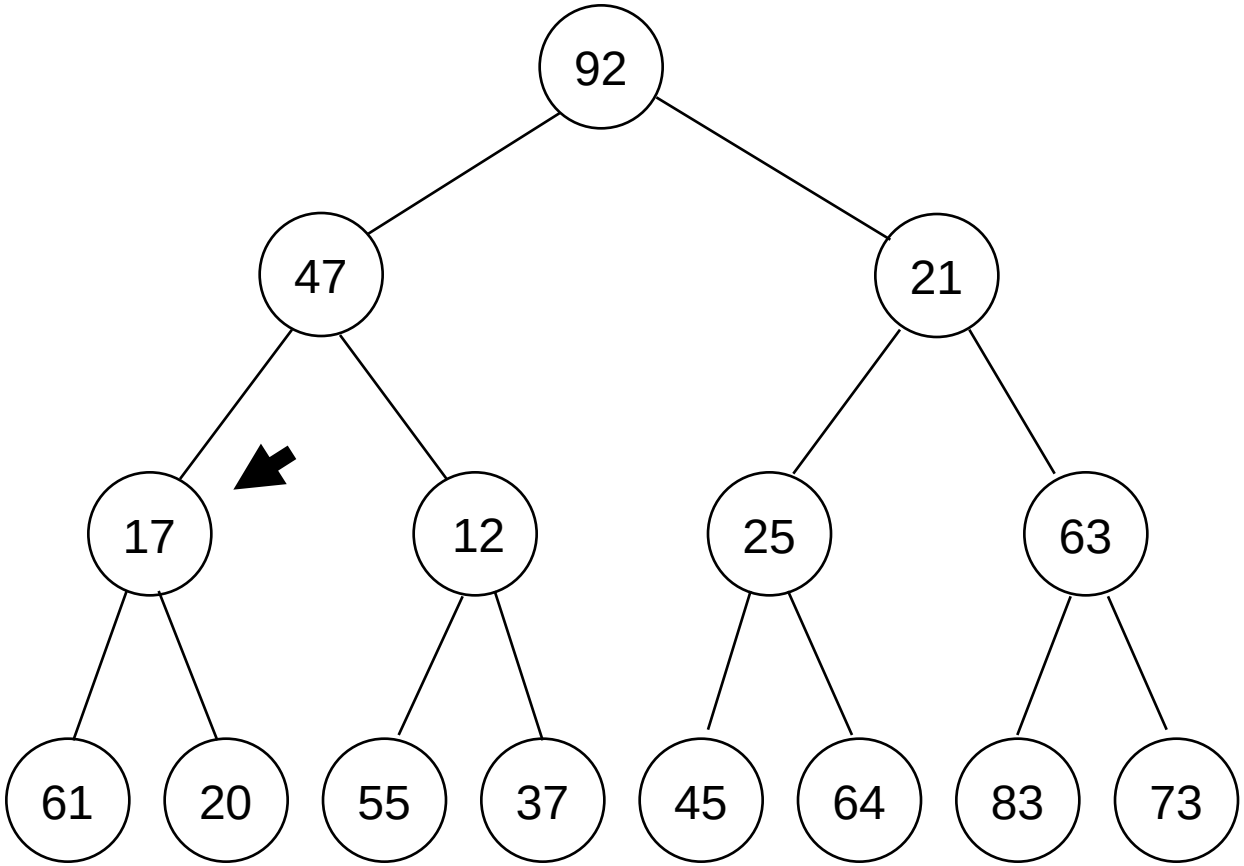
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	25	63	61	17	55	37	45	64	83	73

Monceau - construction



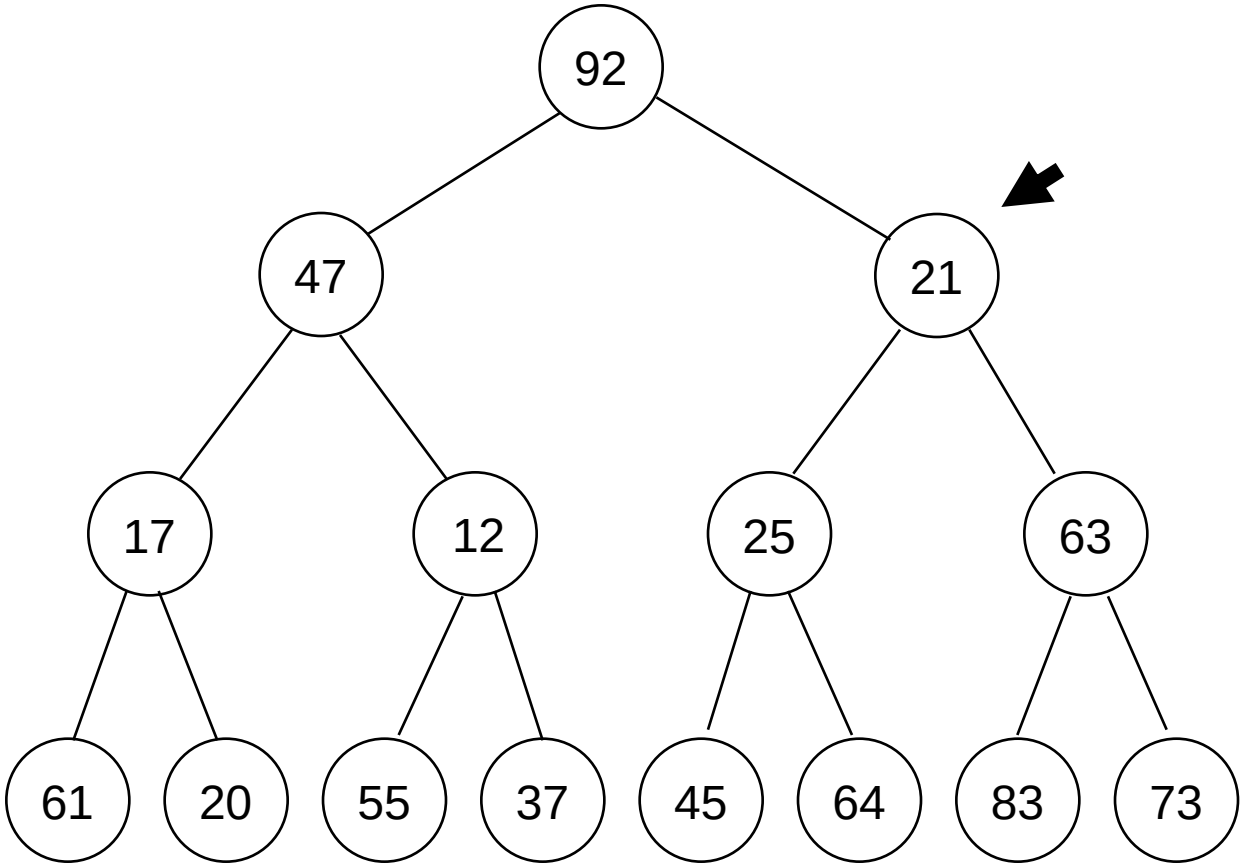
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	20	12	25	63	61	17	55	37	45	64	83	73

Monceau - construction



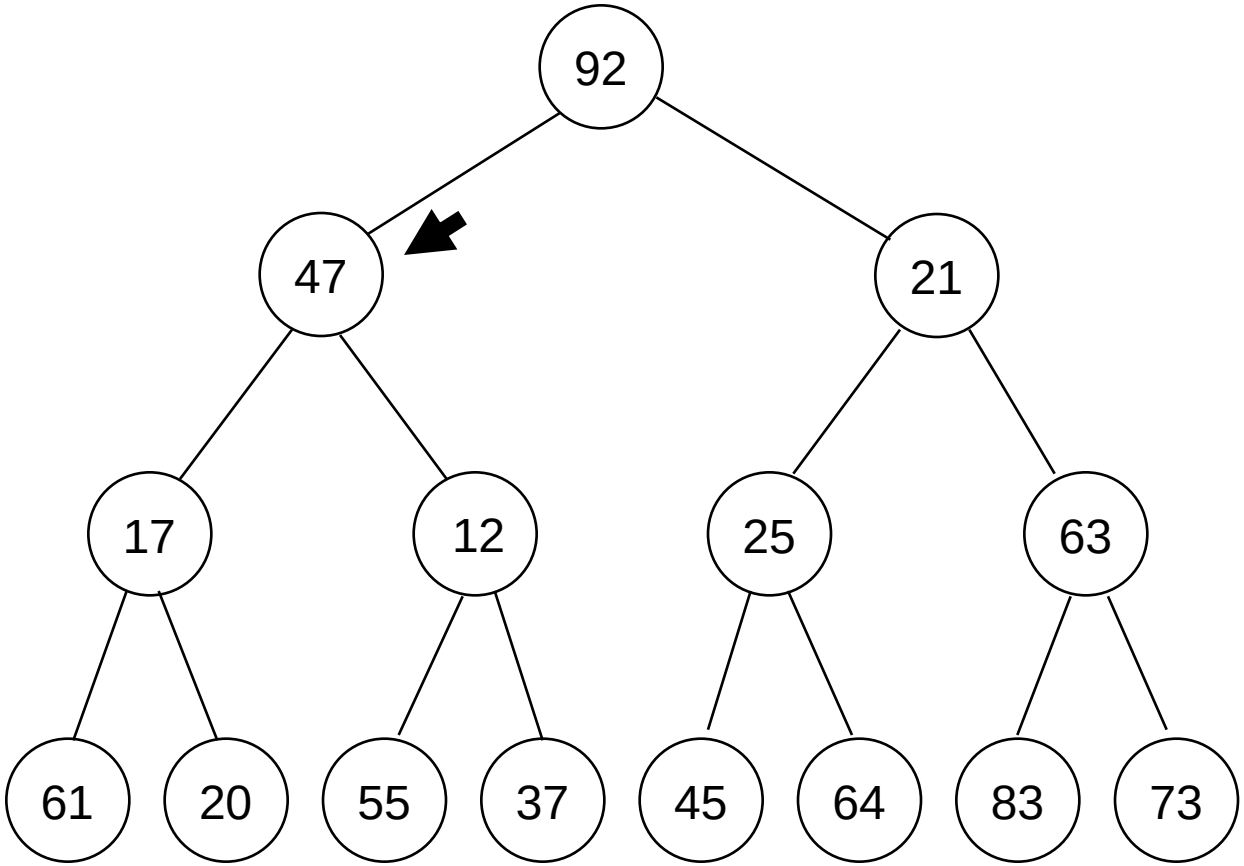
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	17	12	25	63	61	20	55	37	45	64	83	73

Monceau - construction



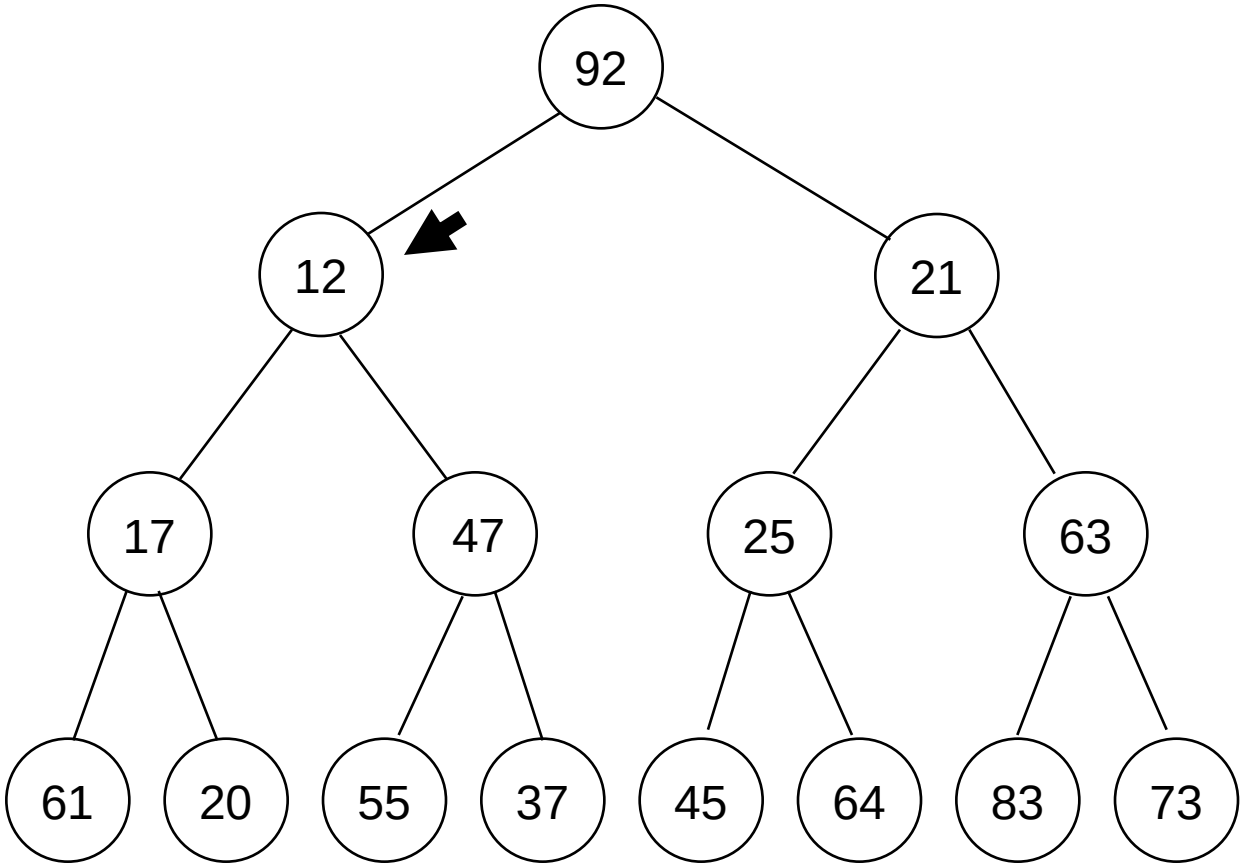
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	17	12	25	63	61	20	55	37	45	64	83	73

Monceau - construction



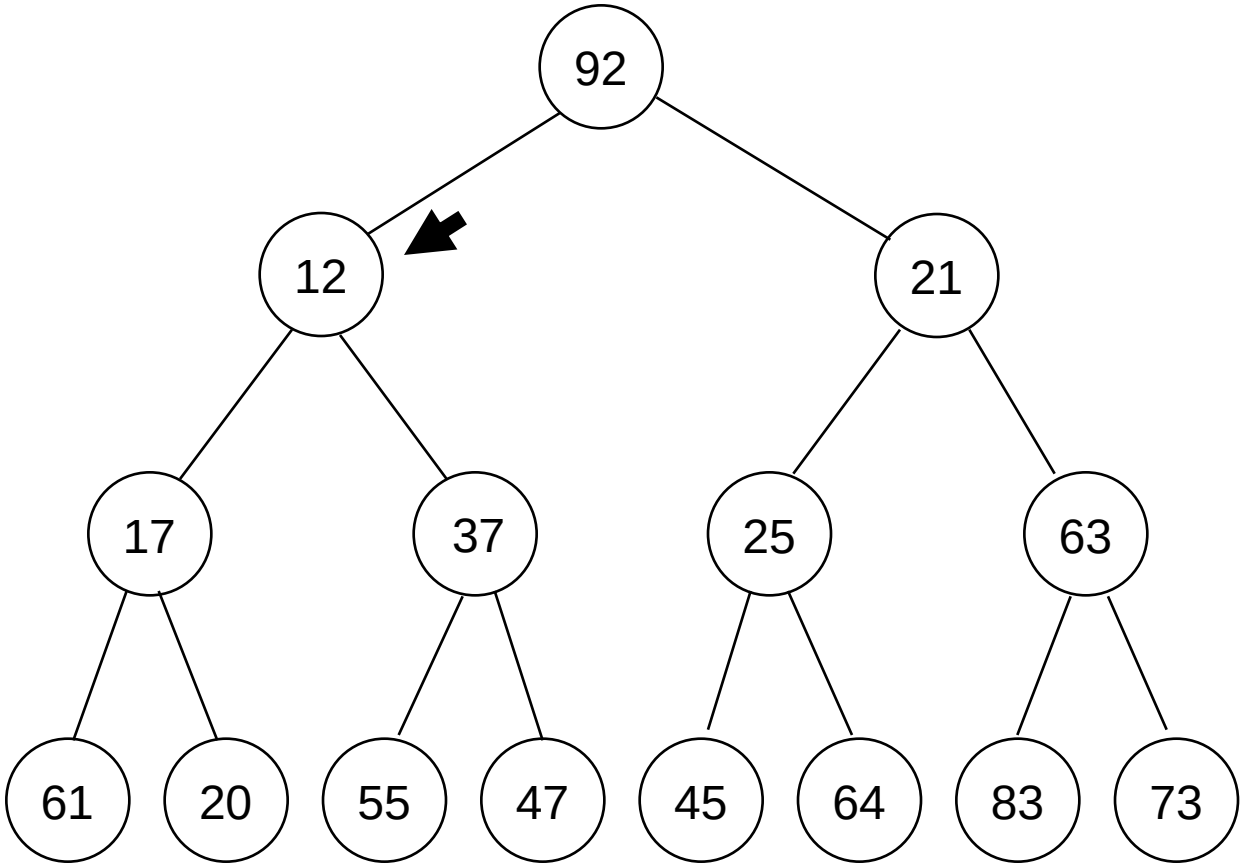
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	47	21	17	12	25	63	61	20	55	37	45	64	83	73

Monceau - construction



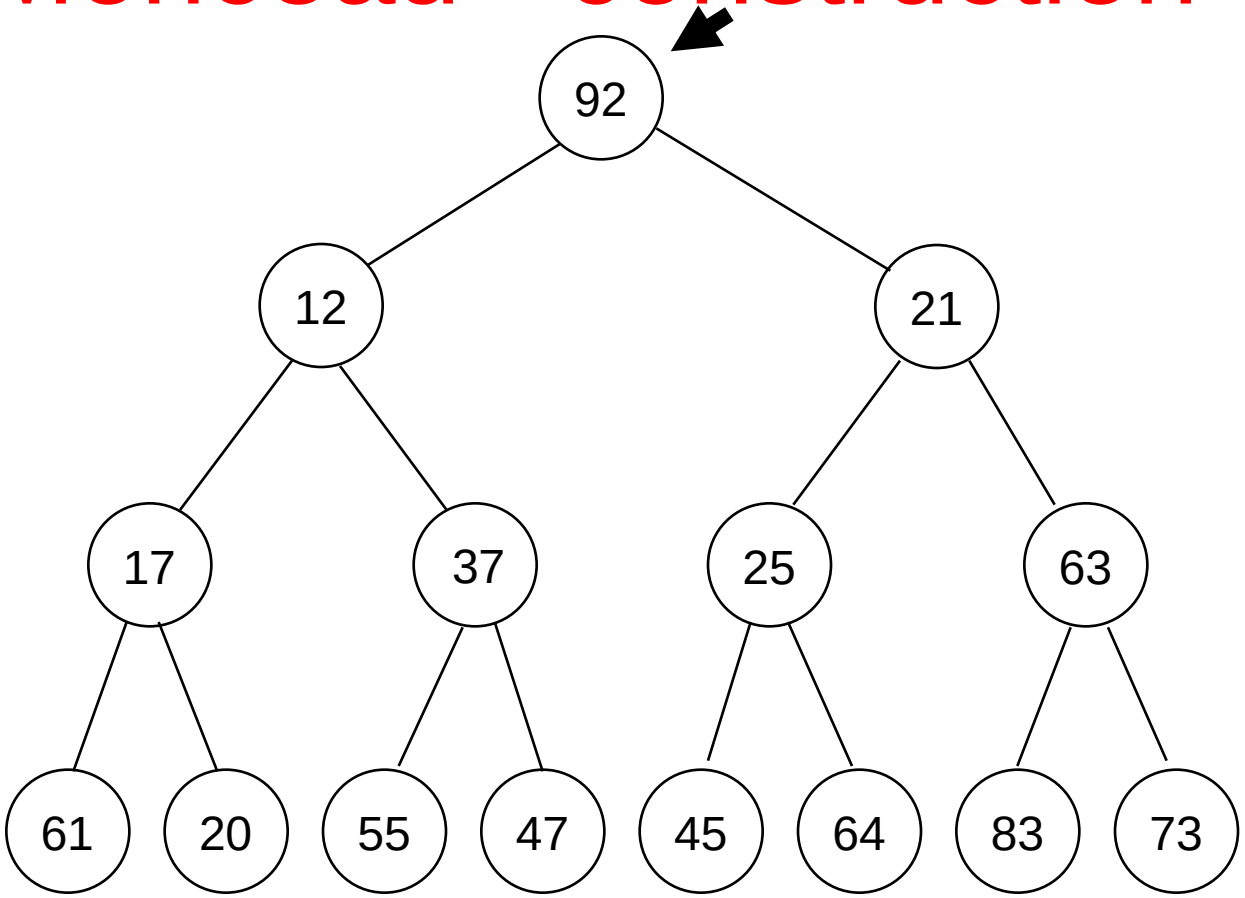
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	12	21	17	47	25	63	61	20	55	37	45	64	83	73

Monceau - construction



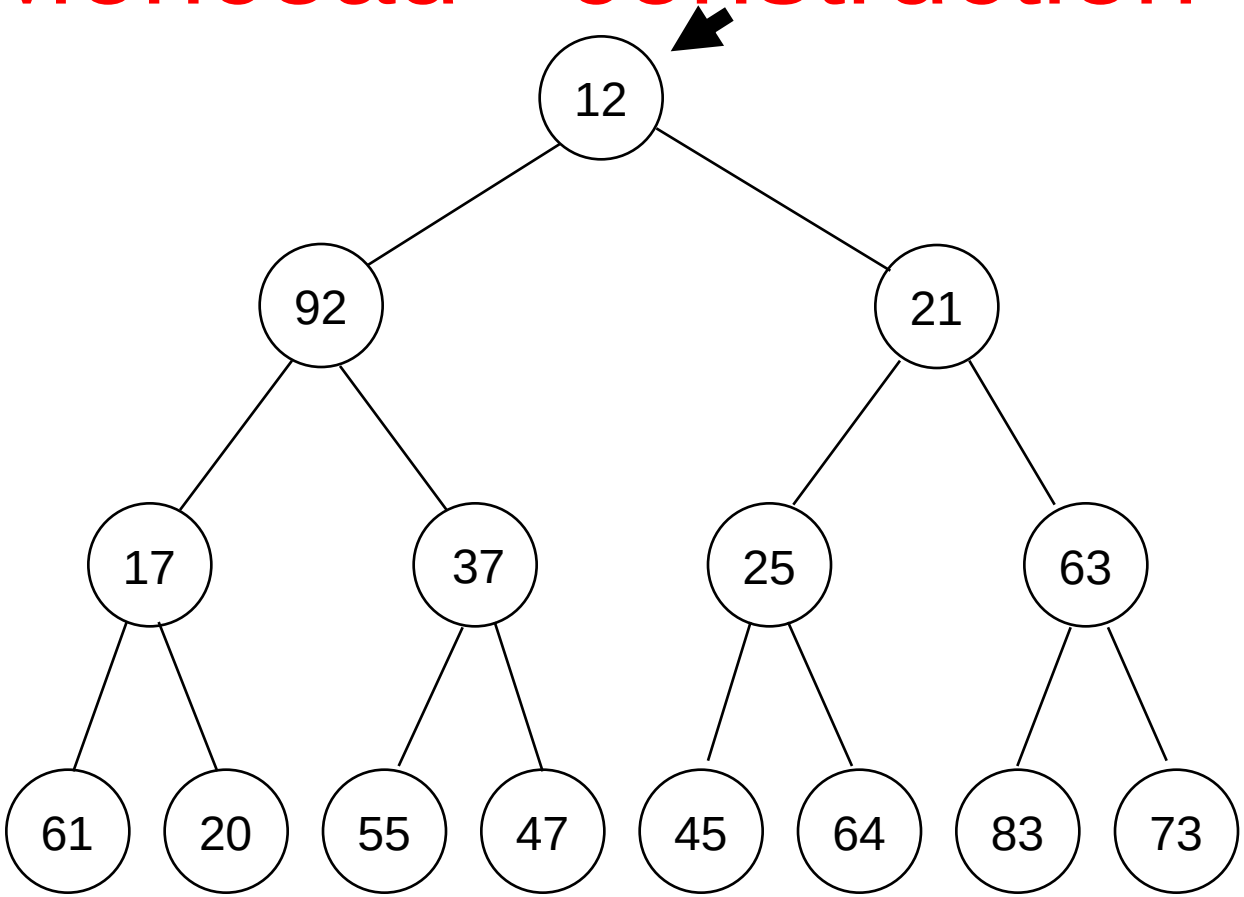
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	12	21	17	37	25	63	61	20	55	47	45	64	83	73

Monceau - construction



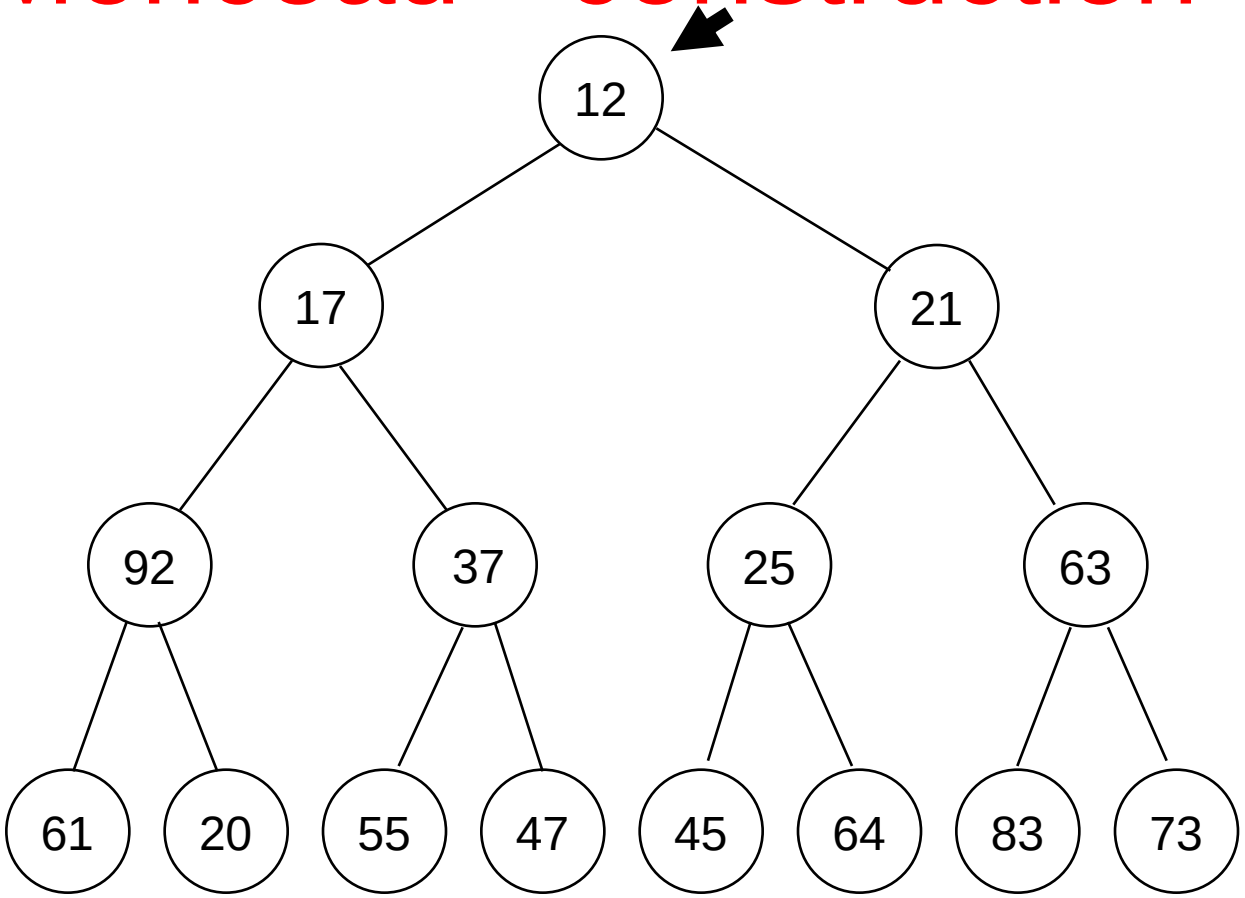
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	92	12	21	17	37	25	63	61	20	55	47	45	64	83	73

Monceau - construction



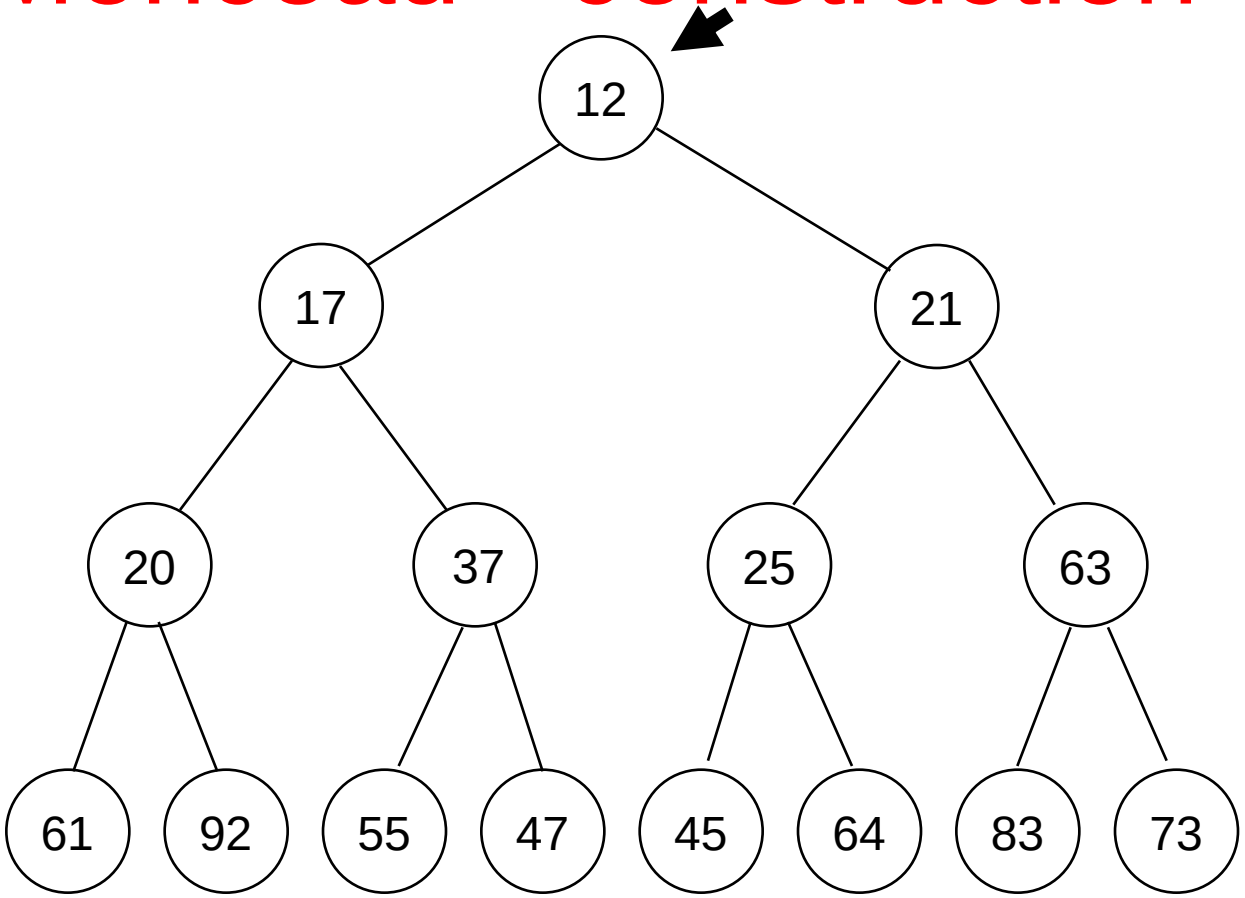
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	92	21	17	37	25	63	61	20	55	47	45	64	83	73

Monceau - construction



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	17	21	92	37	25	63	61	20	55	47	45	64	83	73

Monceau - construction



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	17	21	20	37	25	63	61	92	55	47	45	64	83	73

Plan

- Définition de monceau et implémentation
- Insertion et retrait d'éléments
- Construction d'un monceux
- Tri par monceau

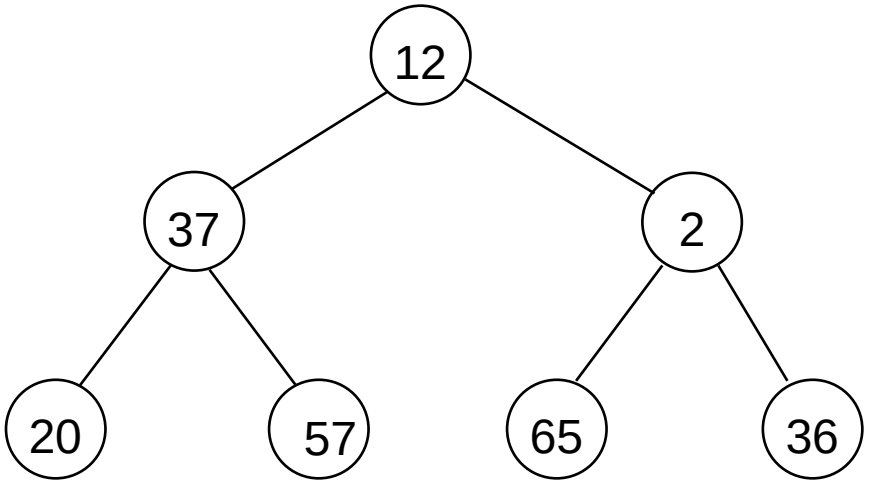
Tri par monceau - Principe

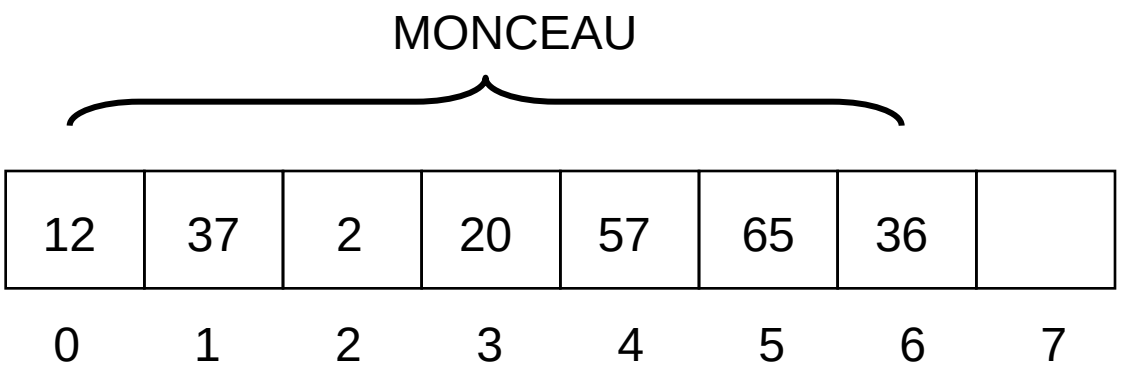
- On applique les modifications suivantes au monceau:
  - chaque nœud a une valeur plus grande ou égale à celle de tous ses descendants
  - dans le tableau qui implémente le monceau, la racine se trouve à l'index 0 (au lieu de 1)
  - pour chaque nœud i, les index sont donc 2*i +1 pour le fils gauche et 2*i+2 pour le fils droit

Tri par monceau - Principe

- La procédure est la suivante:
  1. On met dans le tableau les valeurs à trier
  2. On crée le monceau initial
  3. On retire la racine du monceau (qui est la plus grande valeur) et on la permute avec le dernier item du monceau
  4. On fait percoler la racine, s'il y a lieu
  5. On répète les étapes 3 et 4 avec le monceau obtenu qui contient maintenant un élément de moins

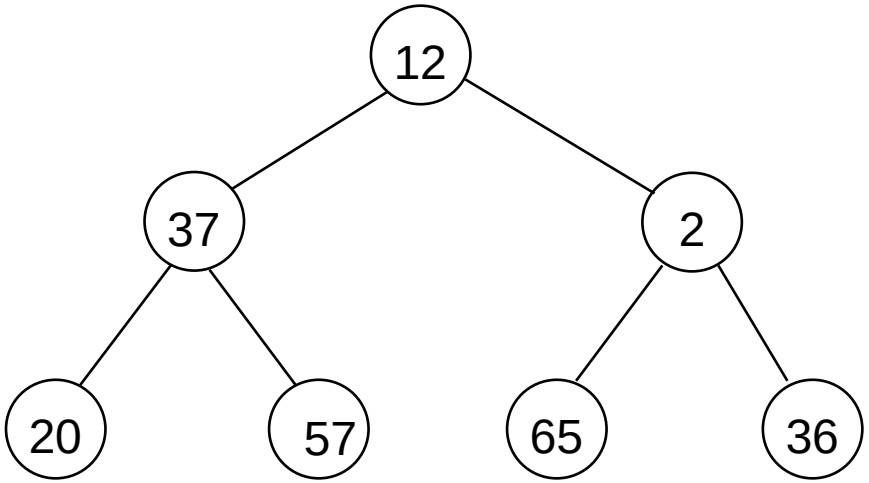
Tri par monceau - Exemple

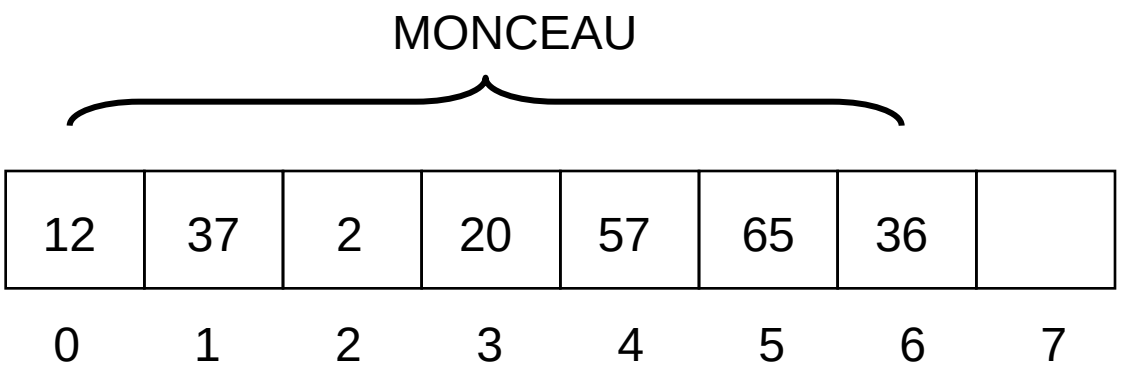




Tri par monceau - Exemple

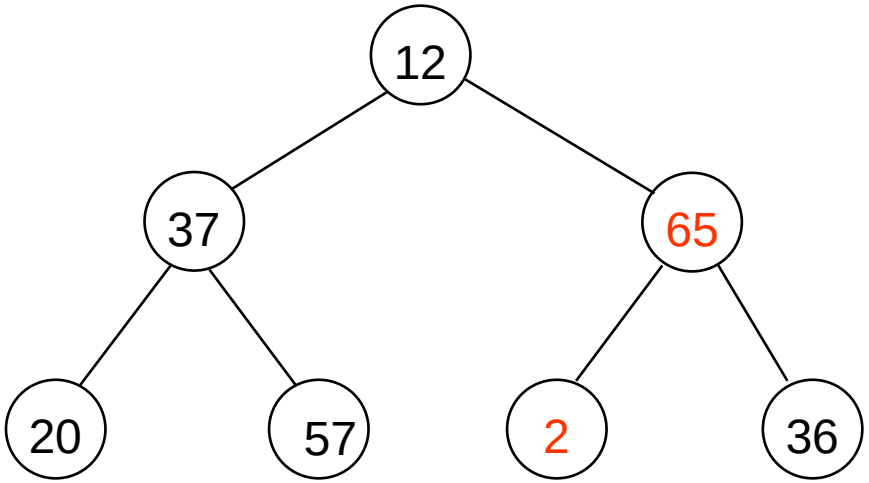
Création du monceau

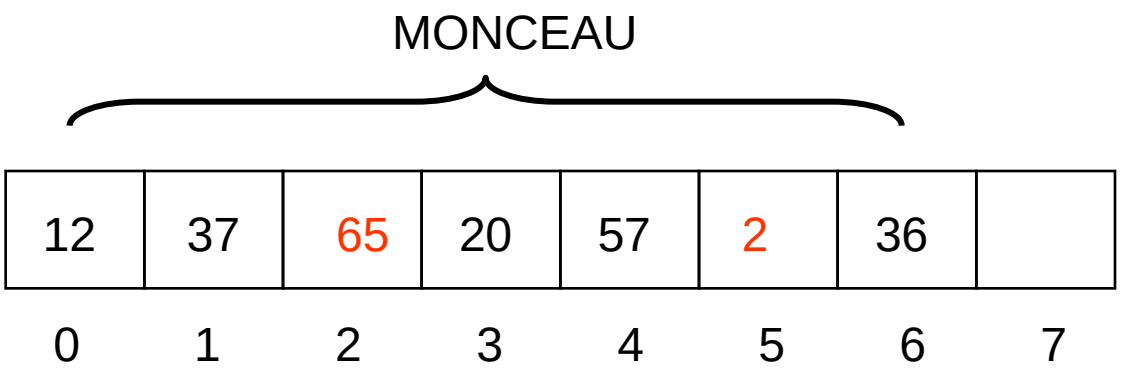




Tri par monceau - Exemple

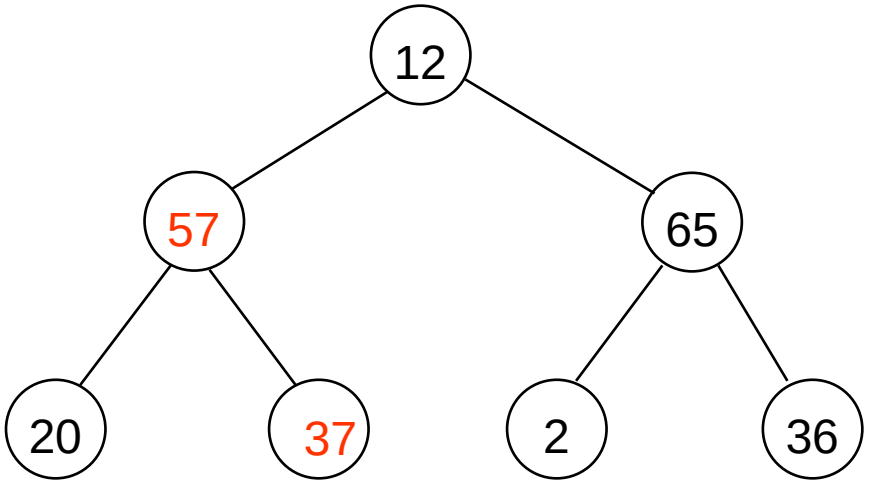
Création du monceau

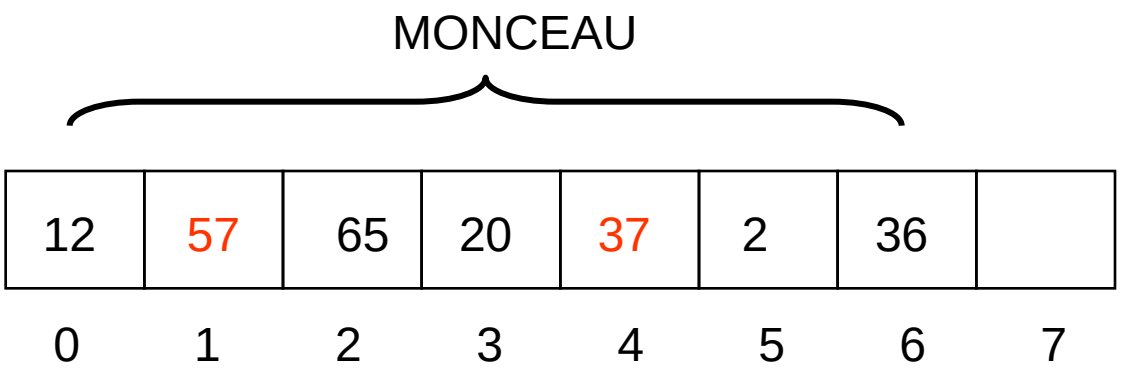




Tri par monceau - Exemple

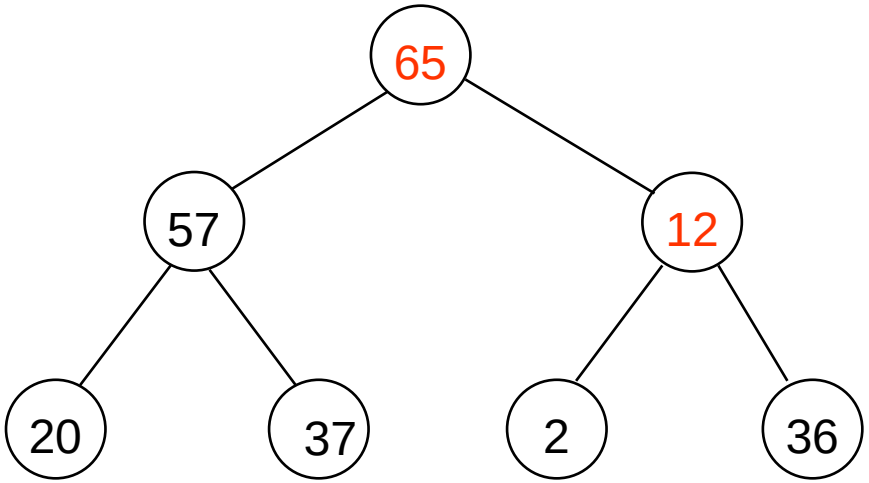
Création du monceau

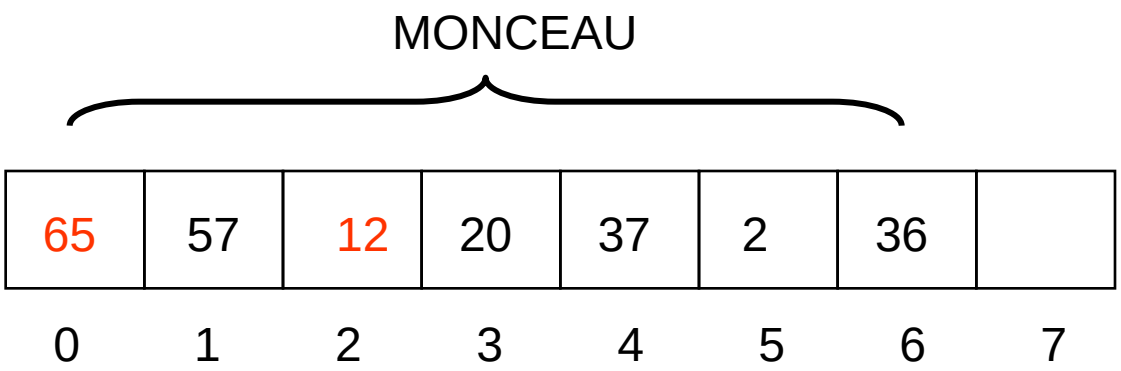




Tri par monceau - Exemple

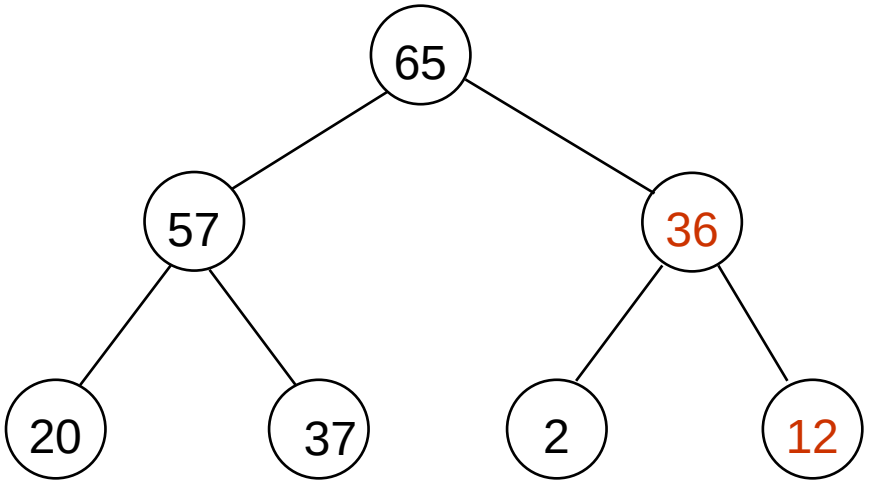
Création du monceau

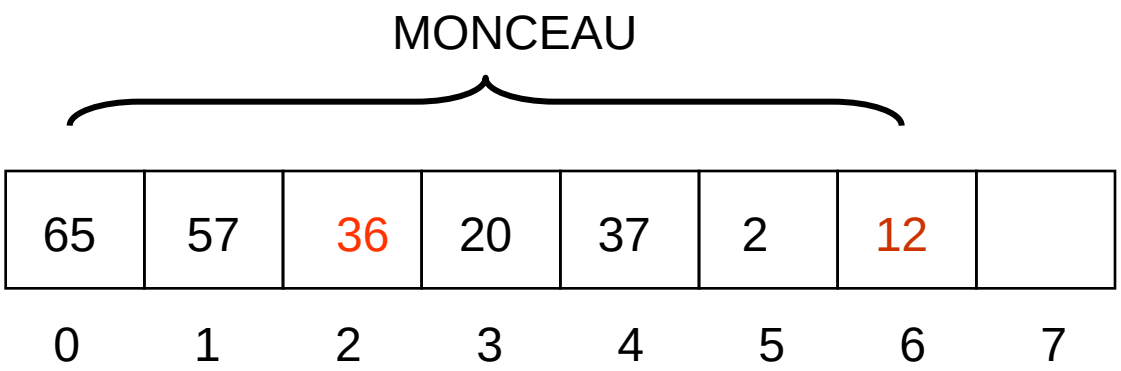




Tri par monceau - Exemple

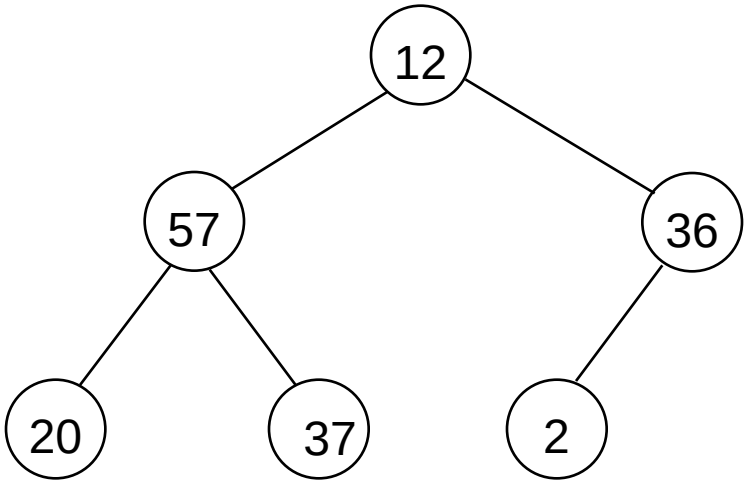
Création du monceau

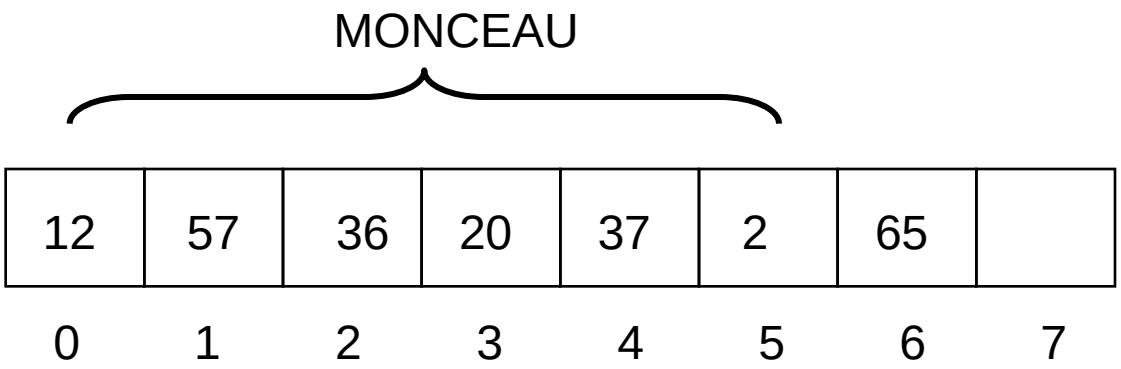




Tri par monceau - Exemple

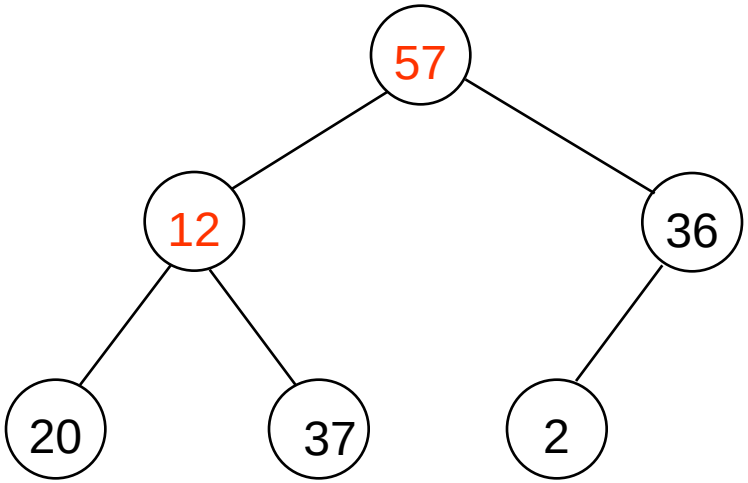
Retrait du 1er élément

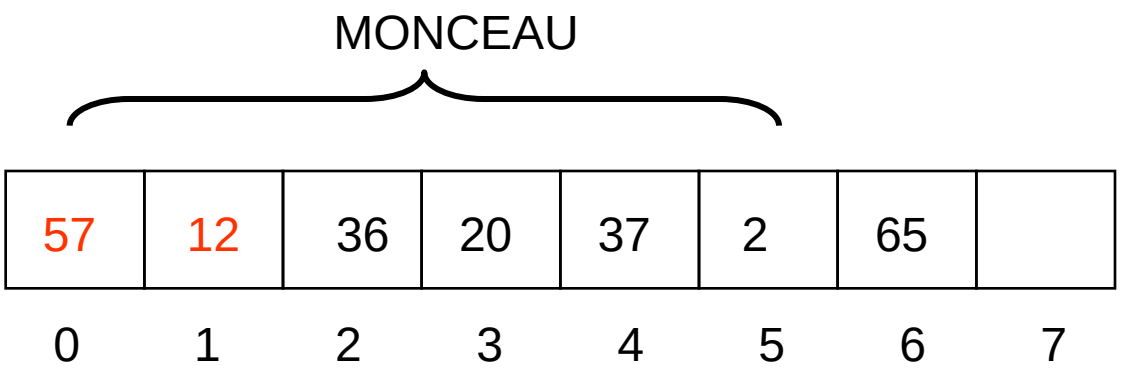




Tri par monceau - Exemple

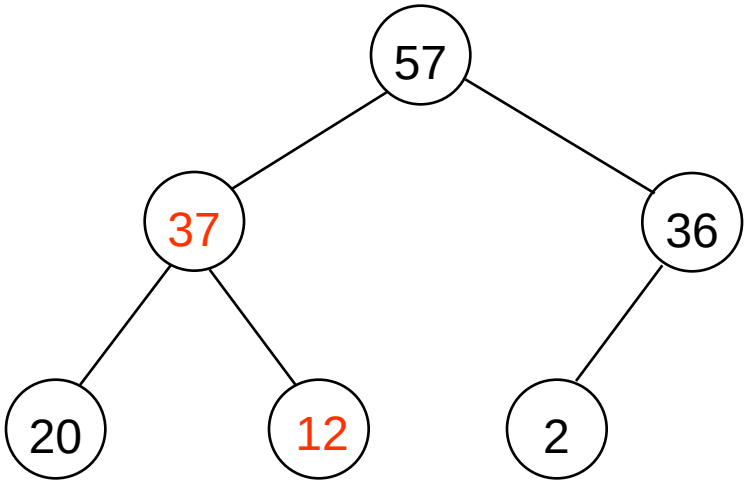
Percoler

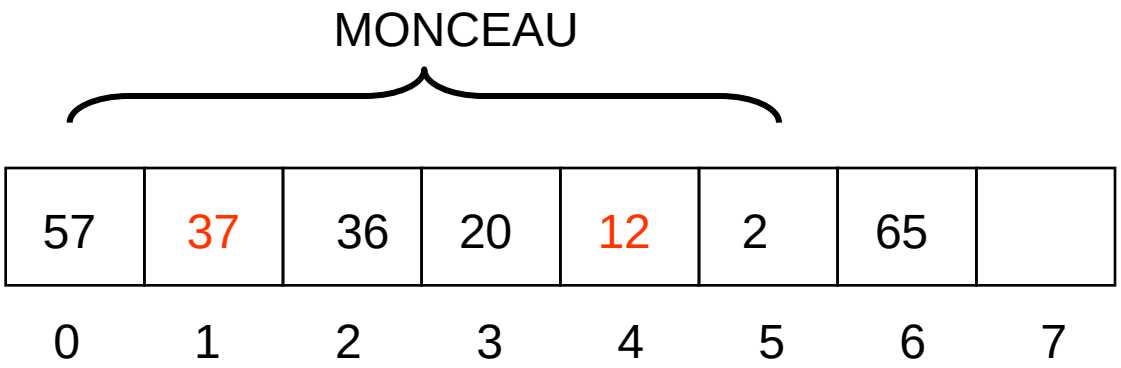




Tri par monceau - Exemple

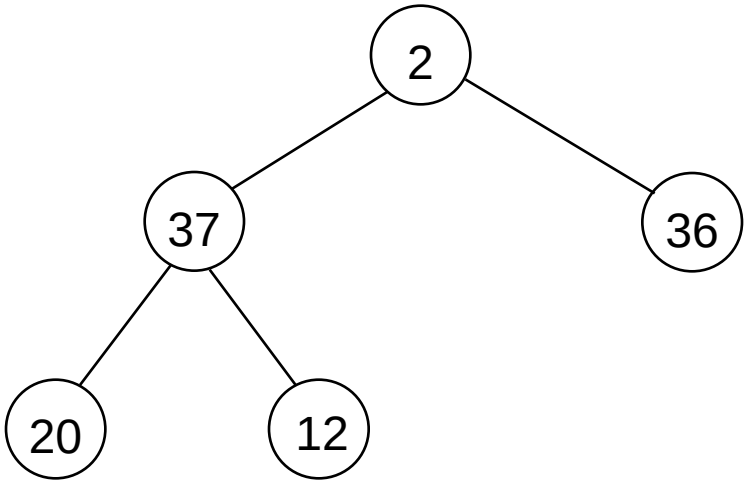
Percoler

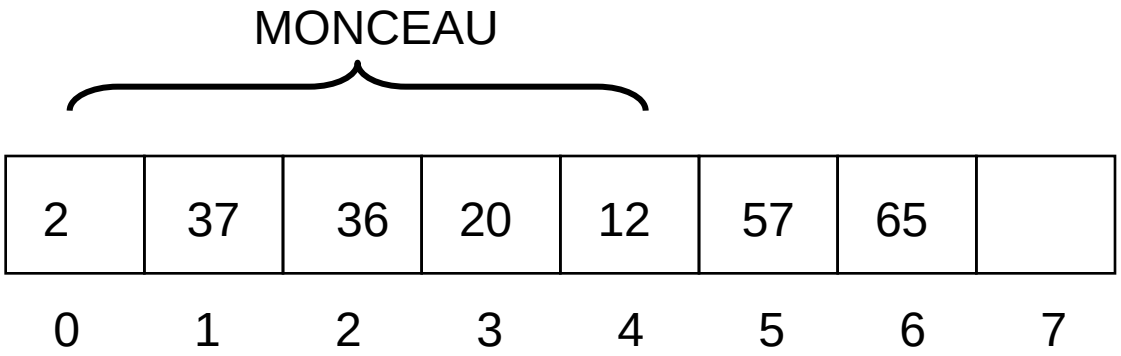




Tri par monceau - Exemple

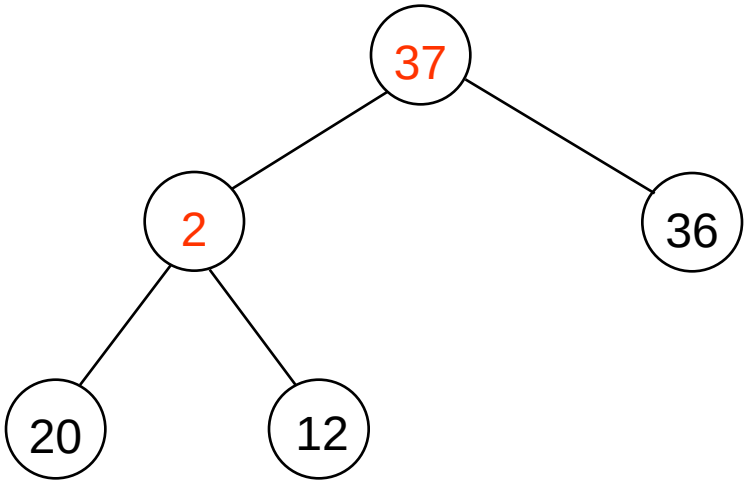
Retrait du 2ème élément

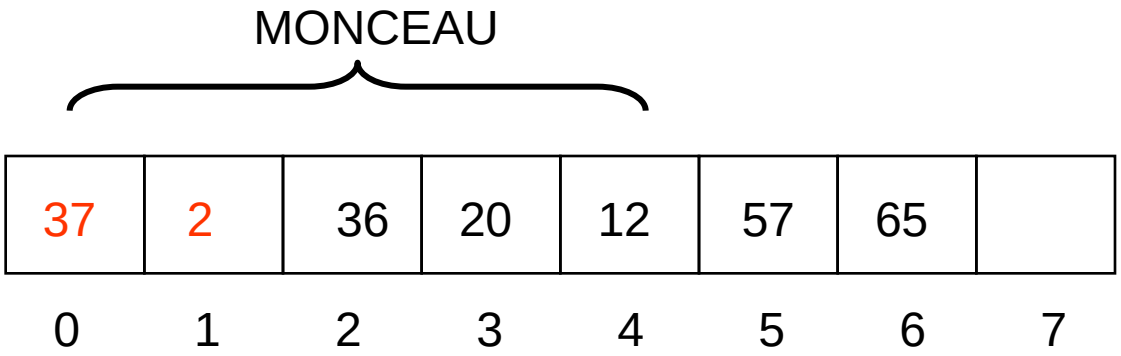




Tri par monceau - Exemple

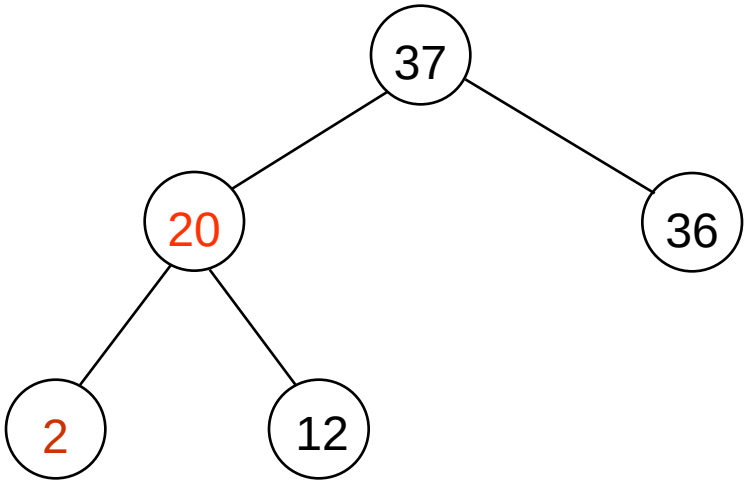
Percoler

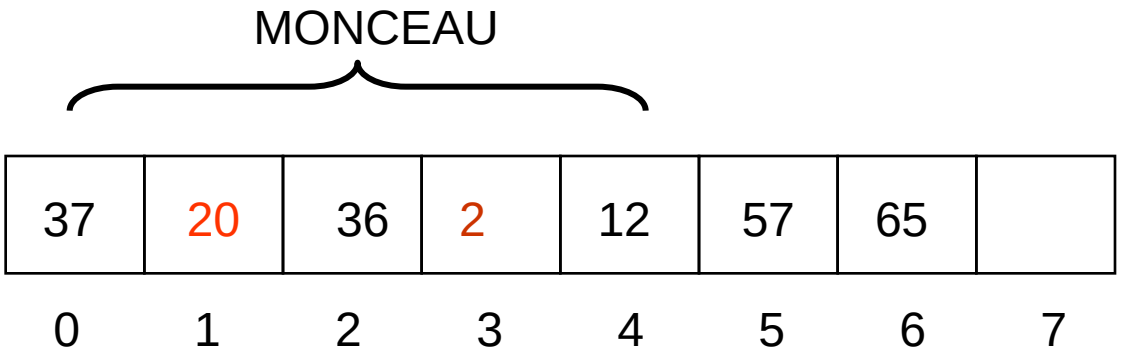




Tri par monceau - Exemple

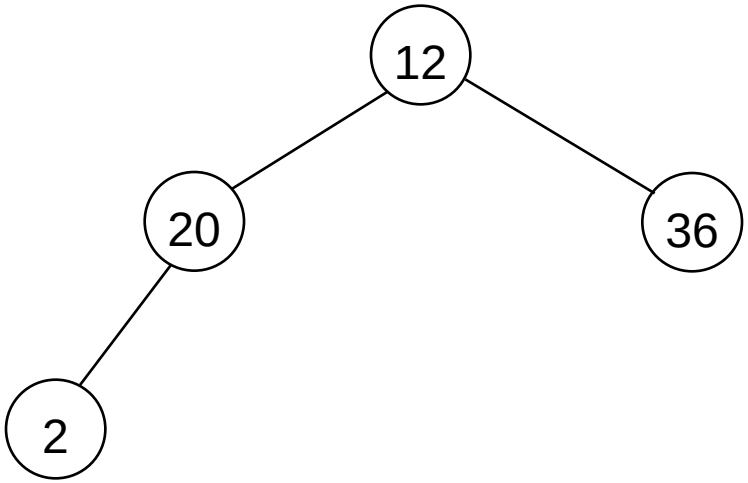
Percoler

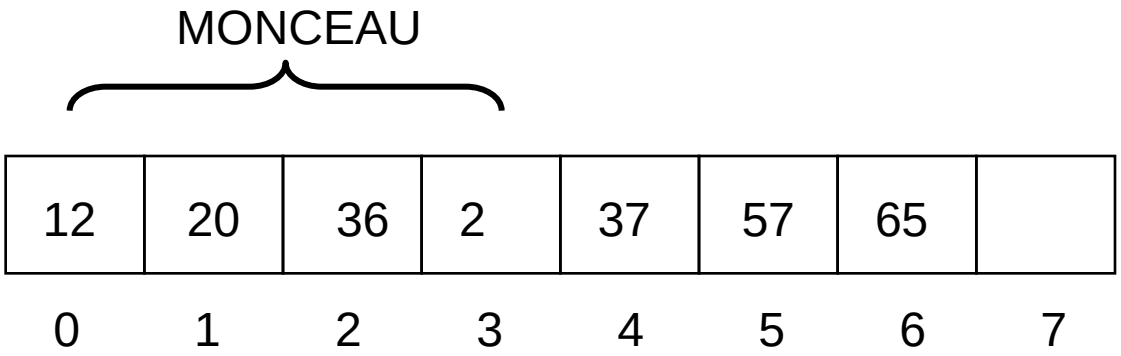




Tri par monceau - Exemple

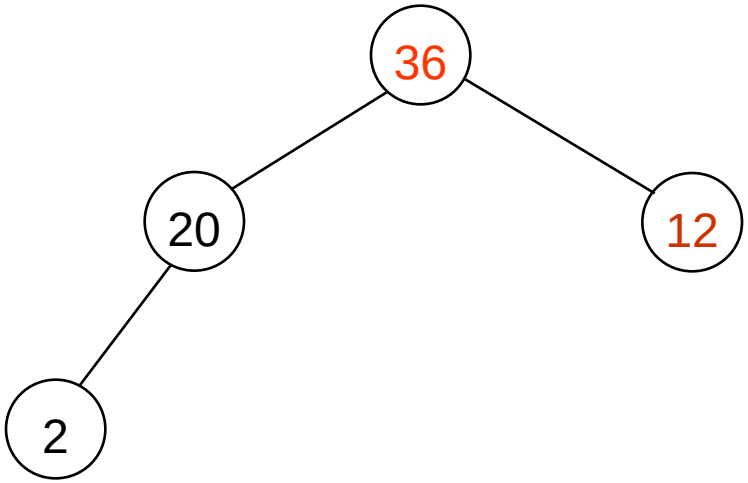
Retrait du 3ème élément

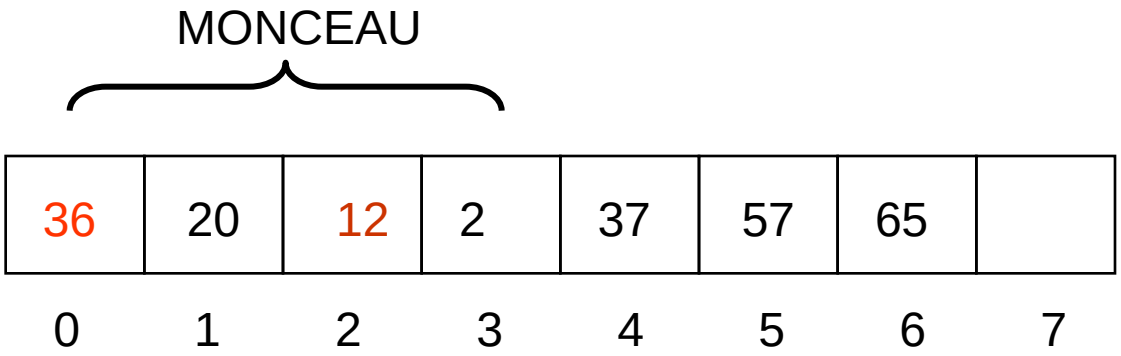




Tri par monceau - Exemple

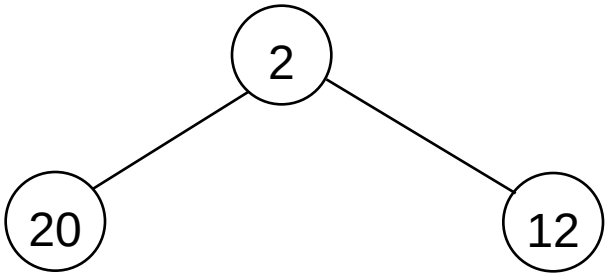
Percoler

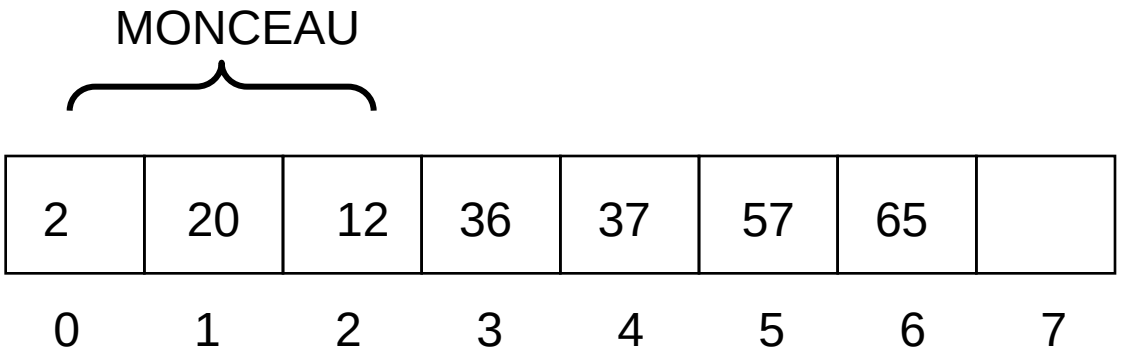




Tri par monceau - Exemple

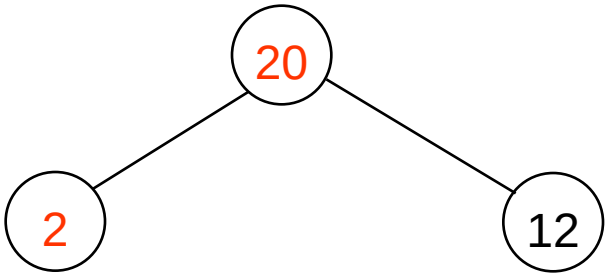
Retrait du 4ème élément

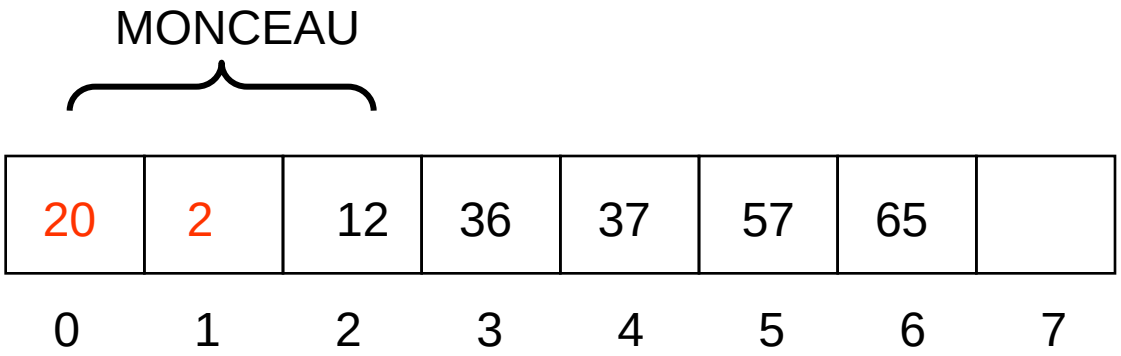




Tri par monceau - Exemple

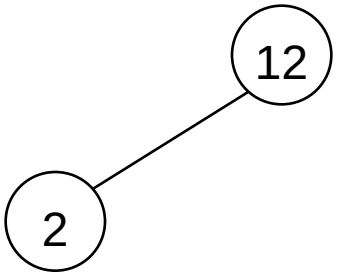
Percoler





Tri par monceau - Exemple

Retrait du 5ème élément

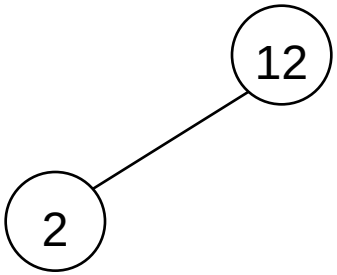


MONCEAU

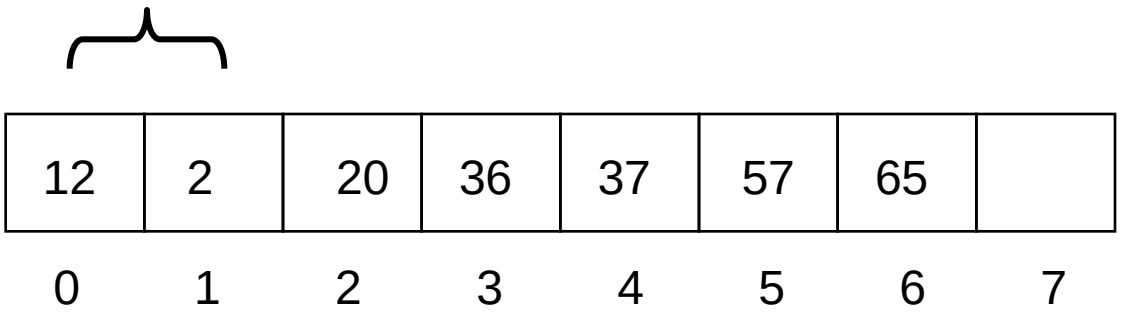
12	2	20	36	37	57	65	
0	1	2	3	4	5	6	7

Tri par monceau - Exemple

Percoler



MONCEAU



Tri par monceau - Exemple

Retrait du 6ème élément

Le tableau est maintenant trié

2	12	20	36	37	57	65	
0	1	2	3	4	5	6	7

Tri par monceau

```
/**
 * Standard heapsort.
 * @param a an array of Comparable items.
 */
public static <AnyType extends Comparable<? super AnyType>>
void heapsort( AnyType [ ] a )
{
    for( int i = a.length / 2; i >= 0; i-- )  /* construire le monceau */
        percDown( a, i, a.length );
    for( int i = a.length - 1; i > 0; i-- )
    {
        swapReferences( a, 0, i ); /* permuter le maximum (racine)
                    avec le dernière élément du monceau */
        percDown( a, 0, i );
    }
}
```

Tri par monceau (2)

```
/**
 * Internal method for heapsort that is used in deleteMax and buildHeap.
 * @param a:  un tableau dont les éléments sont de type Comparable.
 * @int i: la position de l'élément à percoler.
 * @int n: la position du dernière élément du monceau.
 */
private static <AnyType extends Comparable<? super AnyType>>
void percDown( AnyType [ ] a, int i, int n ) {
    int child;
    AnyType tmp;
    for( tmp = a[ i ]; leftChild( i ) < n; i = child ) {
        child = leftChild( i );
        if( child != n - 1 && a[ child ].compareTo( a[ child + 1 ] ) < 0 )
            child++;
        if( tmp.compareTo( a[ child ] ) < 0 )
            a[ i ] = a[ child ];
        else
            break;
    }
    a[ i ] = tmp;
}
```

Tri par monceau (3)

```
/**
  * Internal method for heapsort.
  * @param i the index of an item in the heap.
  * @return the index of the left child.
  */
 private static int leftChild( int i )
 {
     return 2 * i + 1;
 }
```

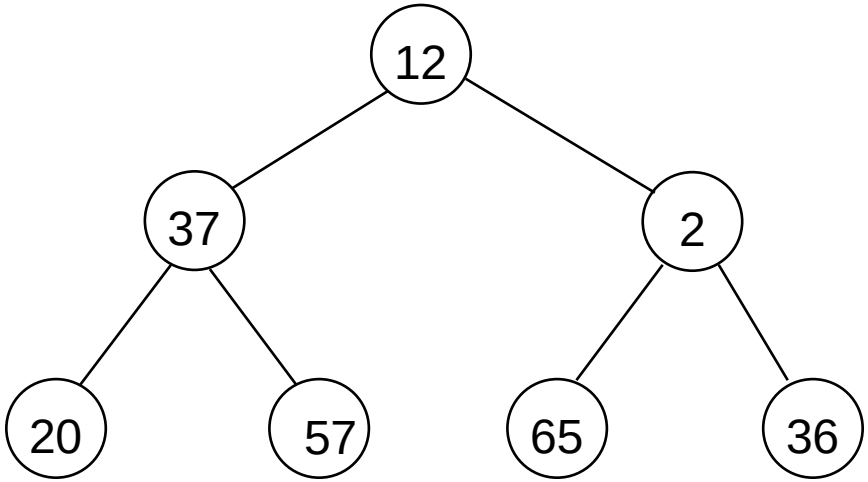
Tri partiel

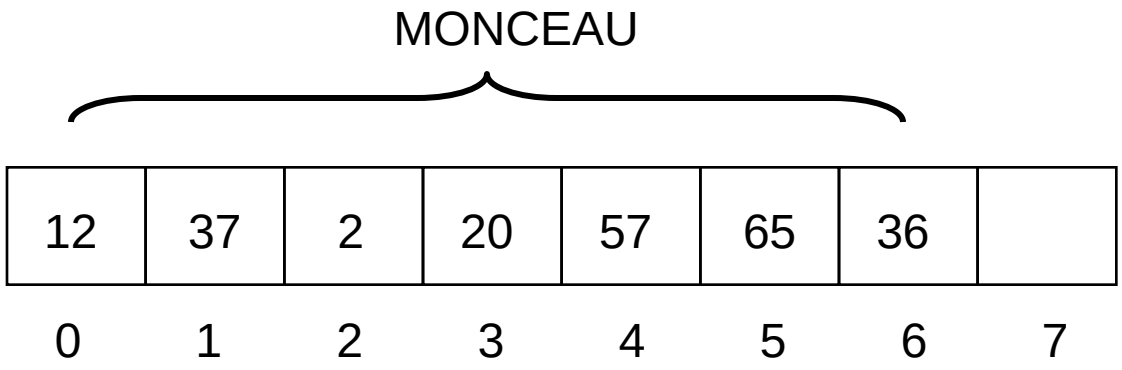
- Supposons maintenant un monceau dont la racine est le plus petit élément (contrairement à l'exemple précédent)
- Supposons que le tableau contient *n* éléments, et une valeur $m < n$
- On remarquera que après *m* itérations, on obtient, à la fin du tableau, et en ordre inverse, les *m* plus petits éléments du tableau

Tri partiel (2)

- On peut donc, de manière efficace, trier les *m* plus petits éléments du tableau: O*(m lg n)*
- C'est le même principe que celui du tri par sélection, mais le tri par sélection est moins efficace: *O(mn)*

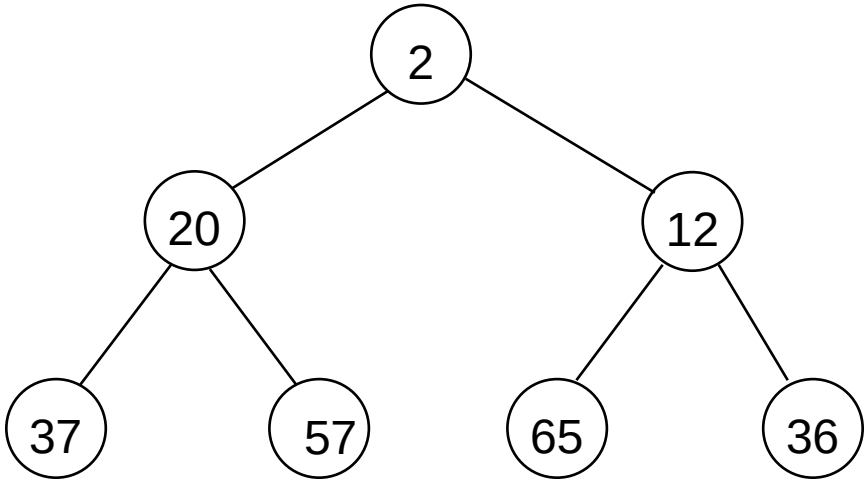
Tri partiel - exemple

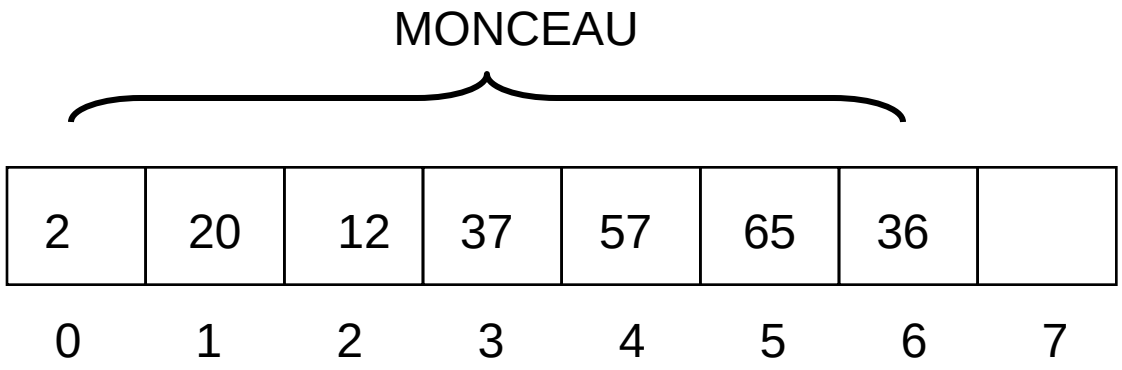




Tri partiel - exemple

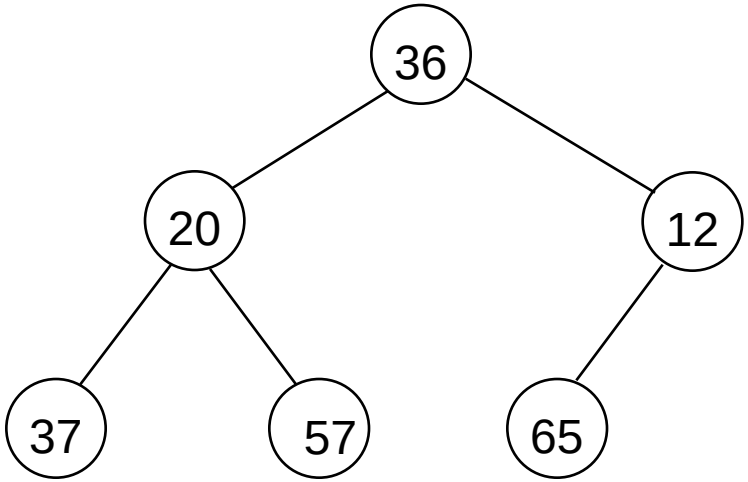
Création du monceau

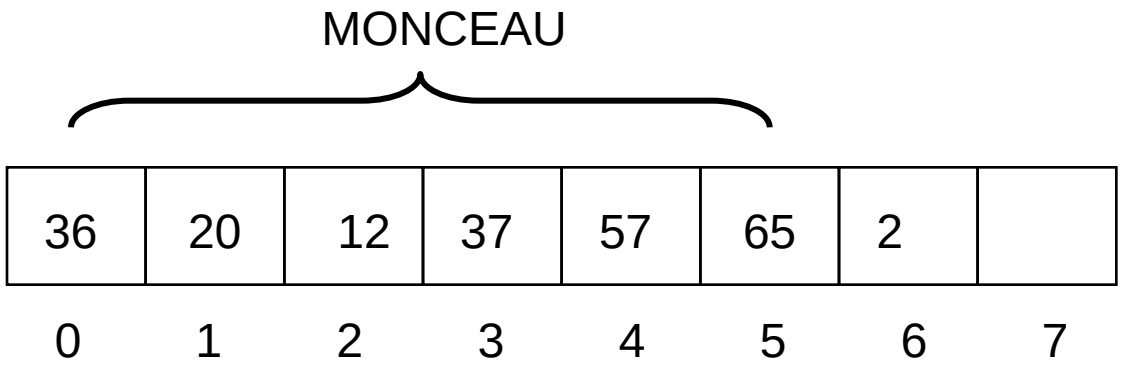




Tri partiel - exemple

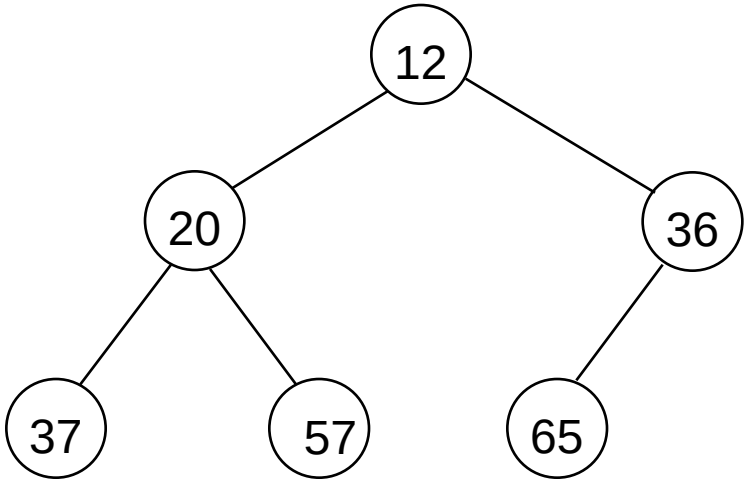
Retrait du 1er élément

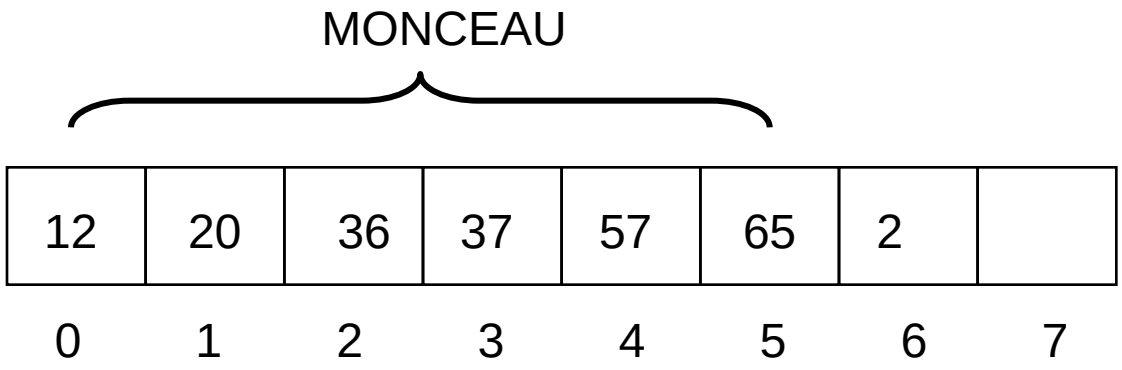




Tri partiel - exemple

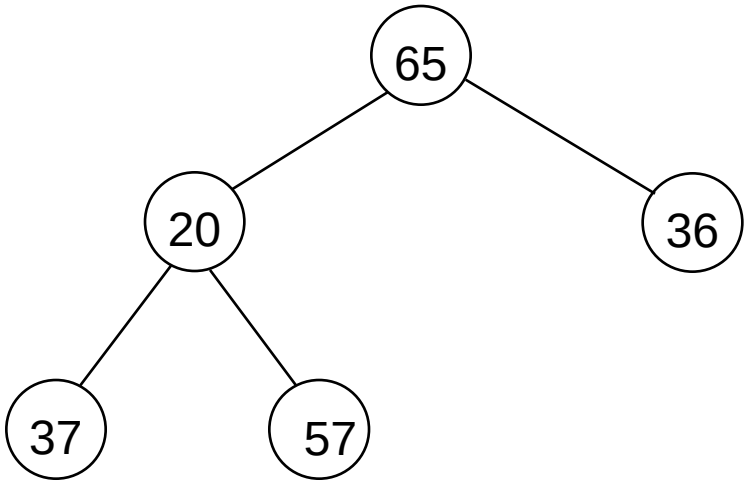
Percoler

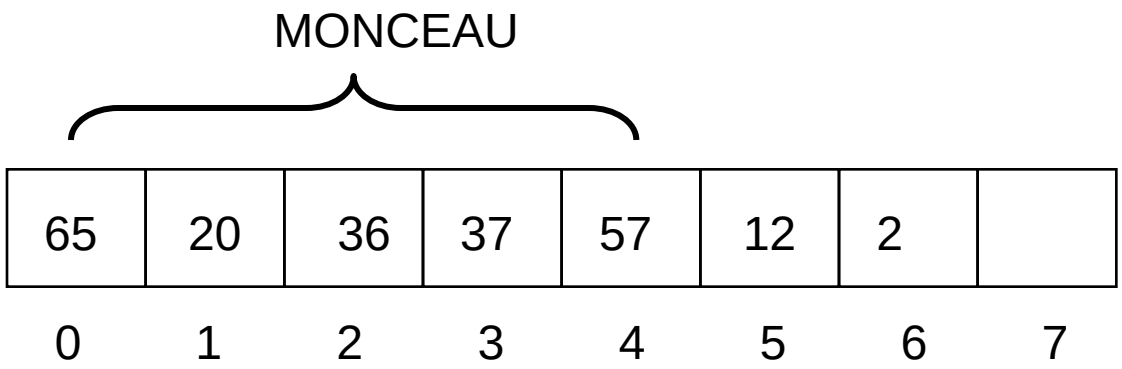




Tri partiel - exemple

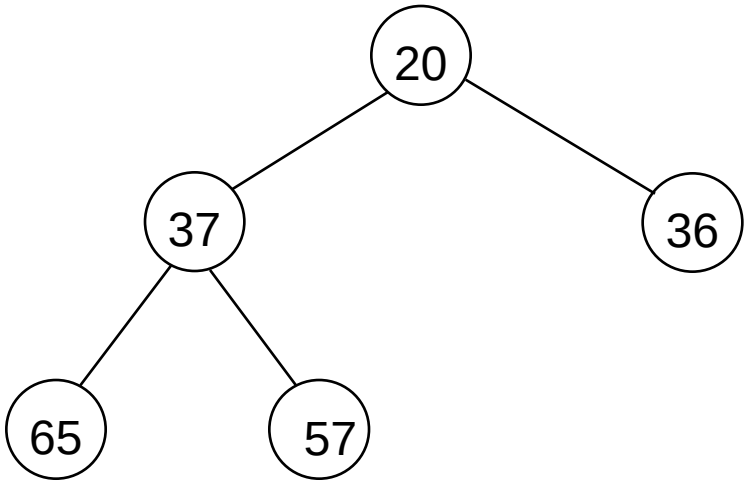
Retrait du 2ème élément

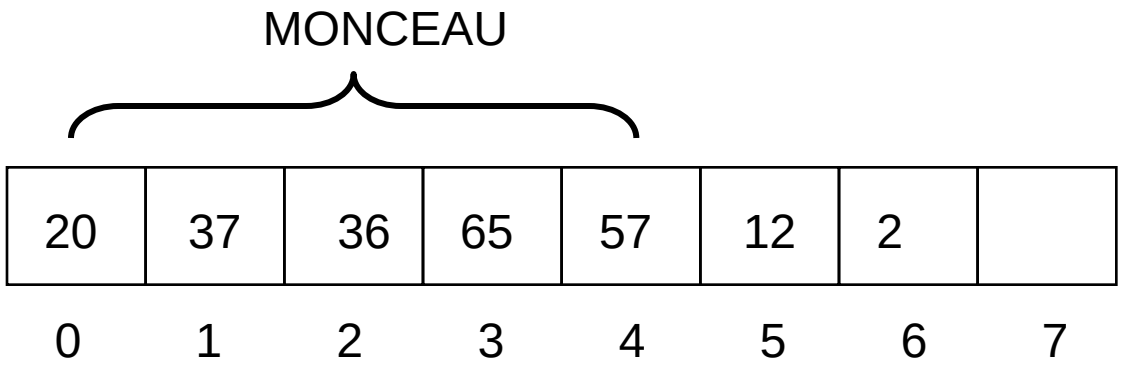




Tri partiel - exemple

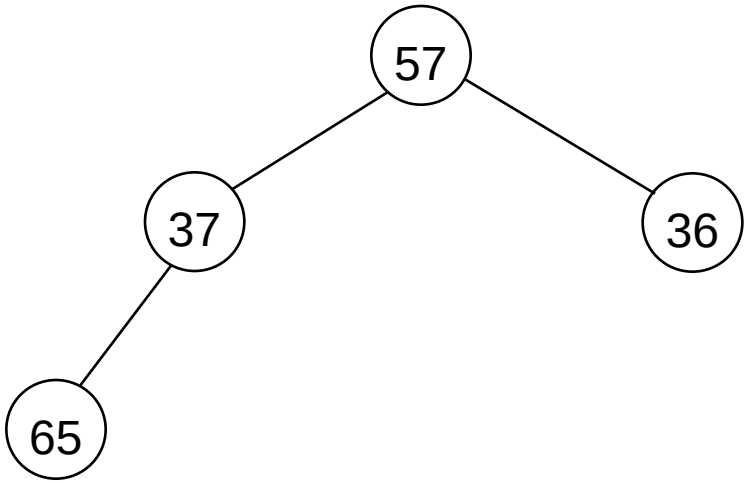
Percoler

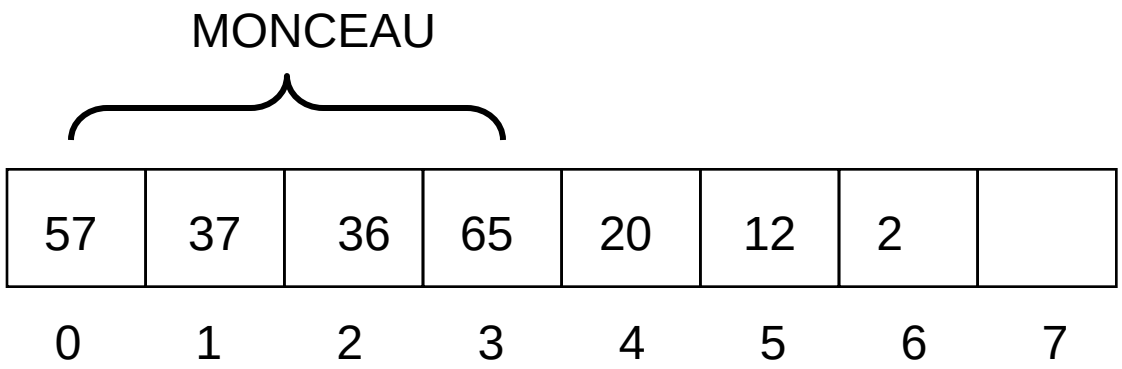




Tri partiel - exemple

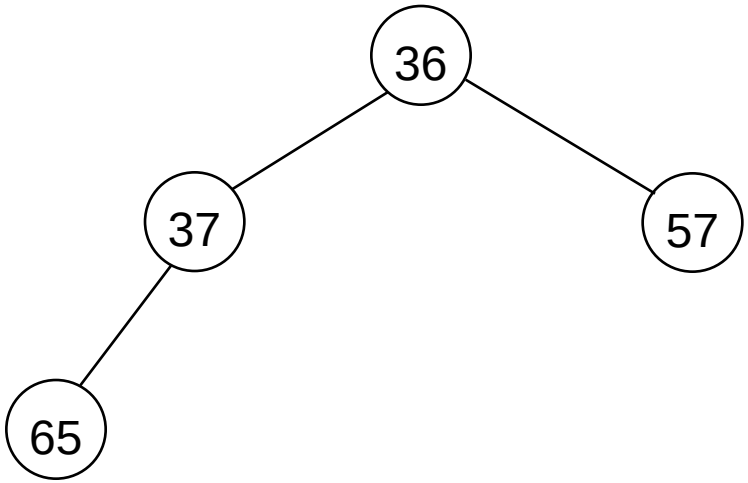
Retrait du 3ème élément

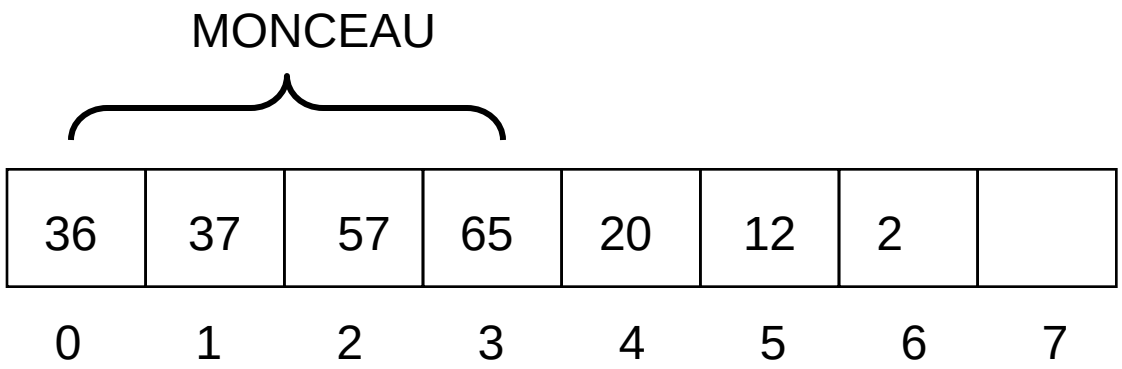




Tri partiel - exemple

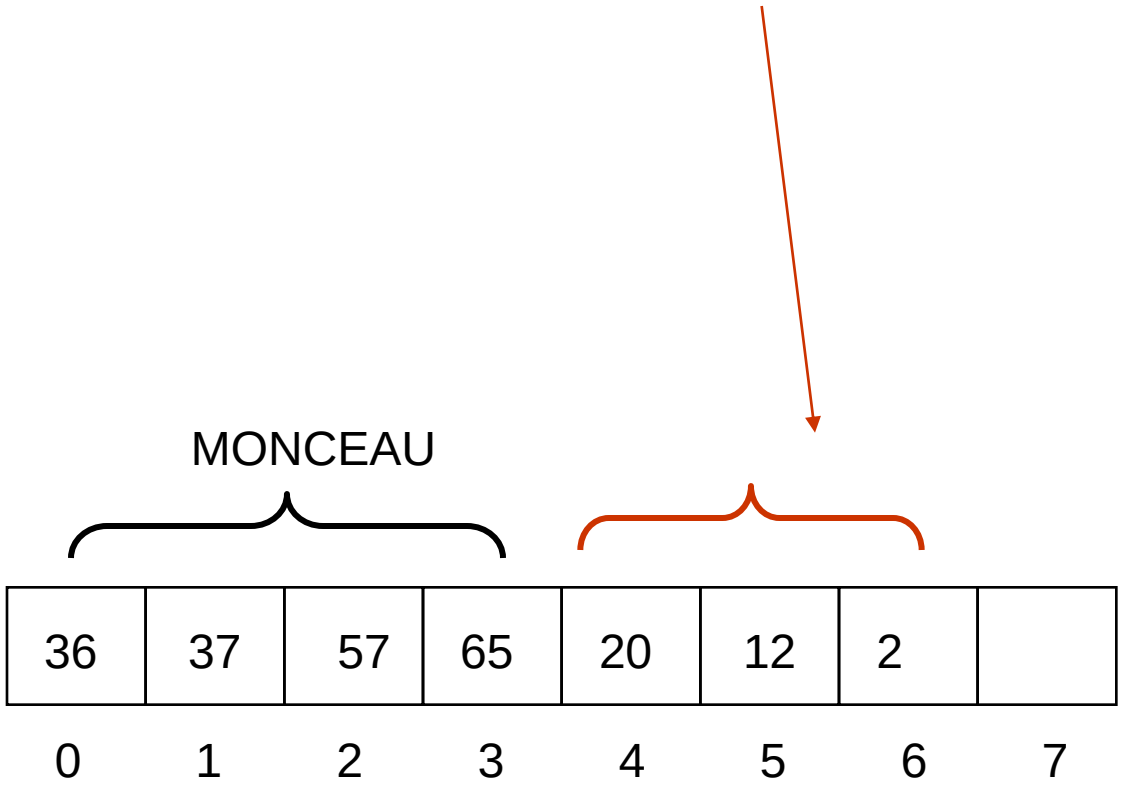
Percoler





Tri partiel - exemple

Si on arrête ici, on voit qu'on a bien obtenu les 3 plus petits éléments, qui sont en ordre croissant si on parcourt le tableau à partir de la fin



Exercice

On veut ajouter une fonction **int findMax( )** à la classe BinaryHeap. Donnez le code de la fonction **int findMax( )** en assurant un temps d'exécution dans le pire cas qui ne doit pas dépasser currentSize/2.

Solution

```
AnyType findMax( )
{
    AnyType max = array[1];

    for( int i = currentSize/2; i <= currentSize; i++ )
    {
        if (array[i].compareTo (max) >0)
            max = array[i];
    }

     return max ;
}
```