



# PROJET ALGAV

## Devoir de Programmation

Ariane ZHANG

Xue YANG

# PLAN

---

1

CLÉS 128 BITS

2

TAS PRIORITÉ MIN

3

FILES BINOMIALES

4

FONCTION DE HACHAGE

5

ARBRE DE RECHERCHE

# CLÉS 128 BITS

```
typedef struct
{
    // Tableau de 4 entiers de 32 bits
    uint32_t parts[4];
} uint128_t;
```

```
bool inf(const uint128_t cle1, const uint128_t cle2);
```

```
bool eg(const uint128_t cle1, const uint128_t cle2);
```



```
gcc cle.c testCle.c -o testCle
./testCle
Cle[0]: 3748217786 1834042959 1795298681 868080045
Cle[1]: 3516050676 3228559303 3741593351 3532701140
Cle[2]: 739618513 2850732339 953169039 317692475
Cle[3]: 1593850405 2268296789 2945062507 3896744521
Cle[4]: 359909585 1456488035 1008780143 28772450
Cle[5]: 3747446808 3501153562 2543666332 2435329020
```

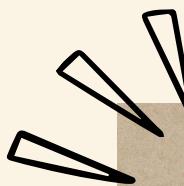
```
Cle[4] inf Cle[0]: 1
Cle[4] inf Cle[1]: 1
Cle[4] inf Cle[2]: 1
Cle[4] inf Cle[3]: 1
Cle[4] inf Cle[4]: 0
Cle[4] inf Cle[5]: 1
```

```
--tetster eg:
Cle[0] eg Cle[0]: 1
Cle[0] eg Cle[1]: 0
Cle[0] eg Cle[2]: 0
Cle[0] eg Cle[3]: 0
Cle[0] eg Cle[4]: 0
Cle[0] eg Cle[5]: 0
Cle[1] eg Cle[0]: 0
```

```
typedef struct Element Element;
struct Element
{
    uint128_t cle;
    Element *suivant;
};
```

```
typedef struct ListeCle ListeCle;
struct ListeCle
{
    Element *premier;
};
```

# TAS PRIORITÉ MIN



## TAS MIN TABLEAU

```
typedef struct Noeud
{
    uint128_t cle;
    struct Noeud *gauche, *droite;
} Noeud;
// Structure de tas minimum avec un tableau
typedef struct
{
    uint128_t *cles; // Tableau des clés
    int taille;      // Nombre d'éléments dans le tas
    int capacite;   // Capacité du tableau
} Tas;
```

```
Tas Construction(Tas *tas, uint128_t *cles, int nbCles);
```

```
void Ajout(Tas *tas, uint128_t cle);
```

```
void AjoutsIteratifs(Tas *tas, uint128_t *cles, int nbCles);
```

```
void SiftDown(Tas *tas, int index);
```

```
void SuprMin(Tas *tas);
```

```
Tas Union(Tas *tas1, Tas *tas2);
```

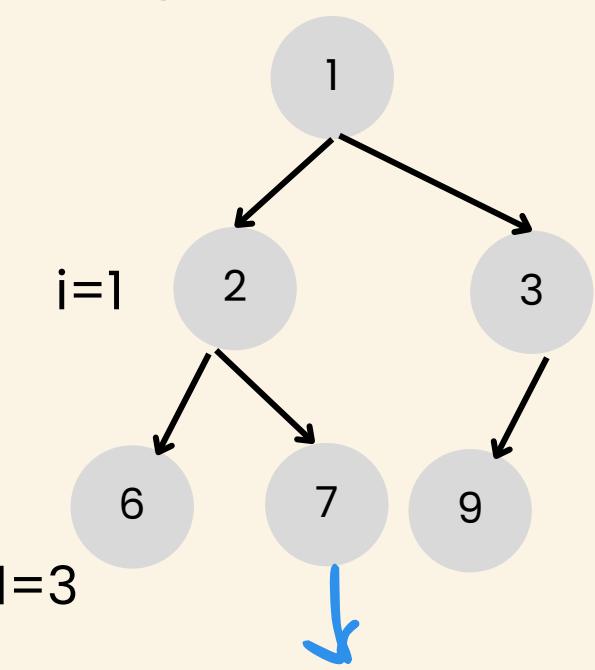
1	2	3	6	7	9
0	1	2	3	4	5

parent =  $(i-1)/2$

$i=1$

enfant gauche =  $2 * i + 1 = 3$

enfant droit =  $2 * i + 2 = 4$



# TAS PRIORITÉ MIN

## TAS MIN TABLEAU

```
gcc cle.c tas.c testTas.c -o testTas  
./testTas  
---Testing Construction...  
---Tas constructed with 6 Clestest_Cles.  
---Tas after Construction:  
----- clé_128 0 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 1 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 2 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 3 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 4 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 5 -----  
df5d8018 d0af5d1a 979d449c 91282bfc  
----- isMinHeap: 1
```

```
---testing Ajout...  
---Ajouter les Cles:  
---tas apres Ajout:  
----- clé_128 0 -----  
1233be51 326a1b2b 9cc45caa 867d053b  
----- clé_128 1 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 2 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 3 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 4 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 5 -----  
df5d8018 d0af5d1a 979d449c 91282bfc  
----- clé_128 6 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b
```

---Testing SupprMin...

---Tas after SupprMin:

```
----- clé_128 0 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 1 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 2 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 3 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 4 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 5 -----  
df5d8018 d0af5d1a 979d449c 91282bfc
```

---Testing AjoutsIteratifs...

---Added 2 Cles iteratively.

---Tas after AjoutsIteratifs:

```
----- clé_128 0 -----  
1233be51 326a1b2b 9cc45caa 867d053b  
----- clé_128 1 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 2 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 3 -----  
d0021c8d a304b898 c87101ae bbce50d1  
----- clé_128 4 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 5 -----  
df5d8018 d0af5d1a 979d449c 91282bfc  
----- clé_128 6 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 7 -----  
df6943ba 6d51464f 6b021579 33bdd9ad
```

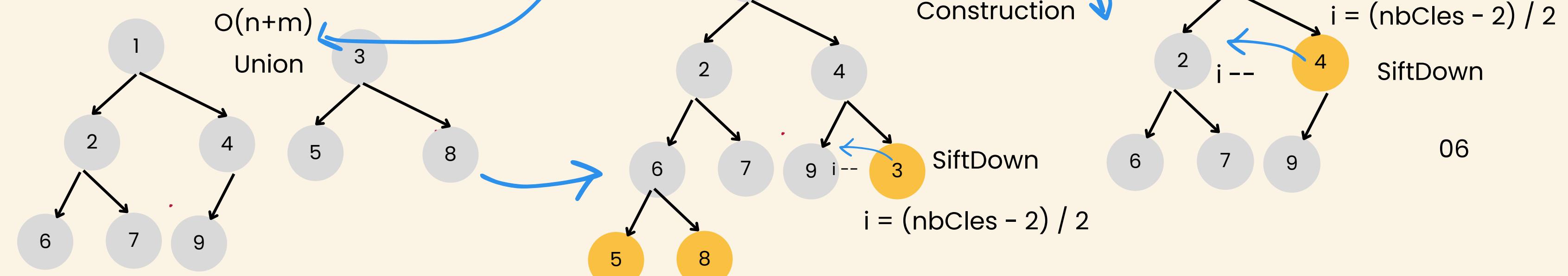
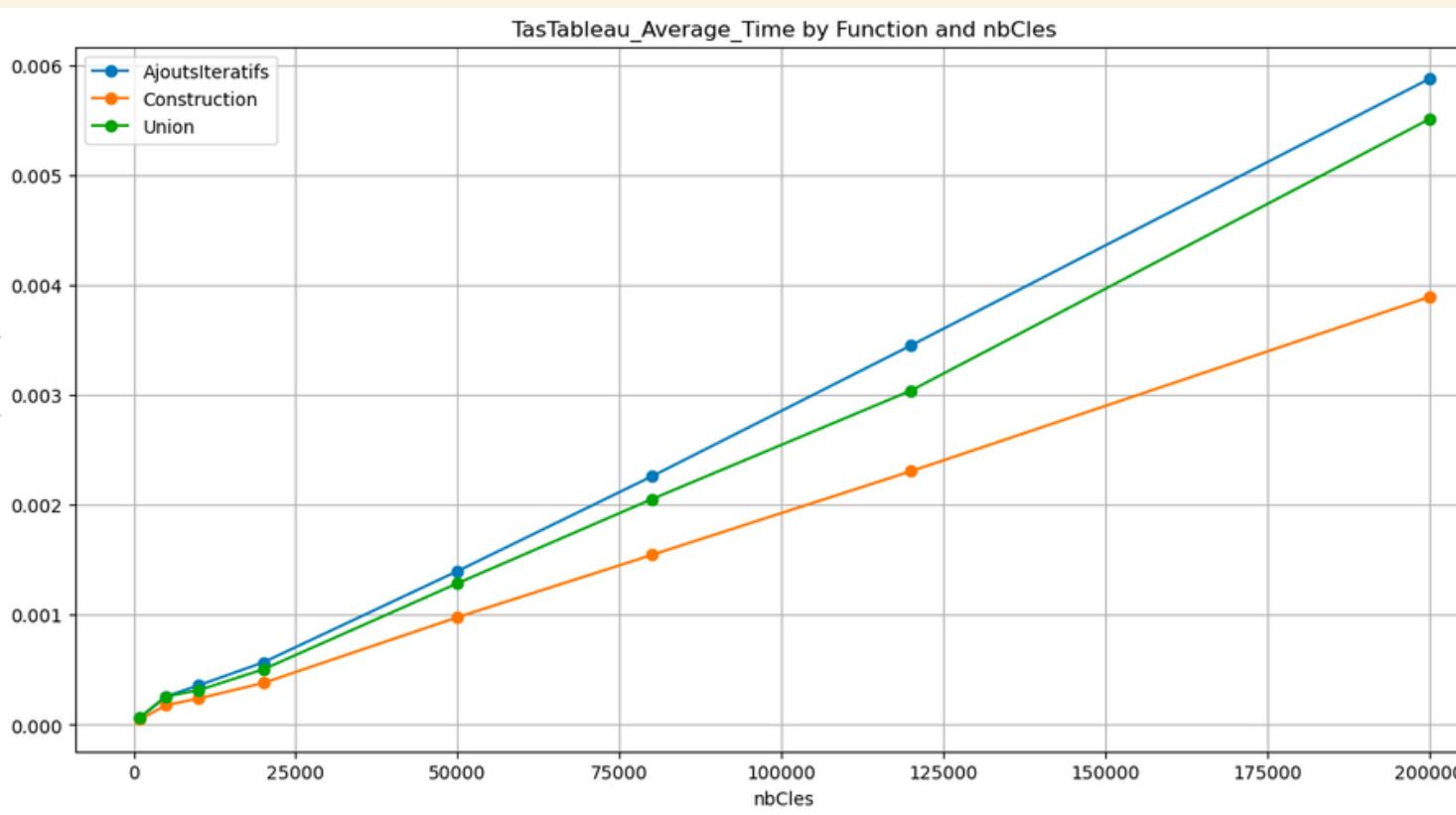
---Testing Union...

---Union tas1 et tas2

```
----- clé_128 0 -----  
1233be51 326a1b2b 9cc45caa 867d053b  
----- clé_128 1 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 2 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 3 -----  
76f3be51 426a9b2a 1cc45caa 967e153b  
----- clé_128 4 -----  
823fbe52 326a8b2b 2cc55cab 567d953c  
----- clé_128 5 -----  
df5d8018 d0af5d1a 979d449c 91282bfc  
----- clé_128 6 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 7 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 8 -----  
d0021c8d a304b898 c87101ae bbce50d1  
----- clé_128 9 -----  
e0029c8d b304b888 d87191ae abce70d2  
----- clé_128 10 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4
```

# TAS PRIORITÉ MIN

## TAS MIN TABLEAU



# TAS PRIORITÉ MIN

## TAS MIN ARBRE BINNAIRE

```
typedef struct TasArbre
{
    uint128_t cle; // Clé de l'élément
    struct TasArbre *gauche;
    struct TasArbre *droite;
} TasArbre;
```

## EXEMPLE AJOUT ITÉRATIF

Ajout itératif d'une liste de cle dans un tas vide tas1:

Liste de cle : 75786 68 25 351 -> 54 58 0 8 -> 35 585 575 34 -> 35 43 34 763 -> 7 68 25 351 -> 0 2 0 8 -> 0 0 0 1 -> NULL

Etas du tas apres chaque ajout (ajoutIteratifs) :

tas arbre apres ajout:

```
( 75786 68 25 351
  Gauche: vide
  Droite: vide )
```

tas arbre apres ajout:

```
( 54 58 0 8
  Gauche: ( 75786 68 25 351
            Gauche: vide
            Droite: vide )
  Droite: vide )
```

tas arbre apres ajout:

```
( 35 585 575 34
  Gauche: ( 75786 68 25 351
            Gauche: vide
            Droite: vide )
  Droite: ( 54 58 0 8
            Gauche: vide
            Droite: vide ) )
```

tas arbre apres ajout:

```
( 0 0 0 1
  Gauche: ( 35 43 34 763
            Gauche: ( 75786 68 25 351
                      Gauche: vide
                      Droite: vide )
            Droite: ( 35 585 575 34
                      Gauche: vide
                      Droite: vide ) )
  Droite: ( 0 2 0 8
            Gauche: ( 54 58 0 8
                      Gauche: vide
                      Droite: vide )
            Droite: ( 7 68 25 351
                      Gauche: vide
                      Droite: vide ) ) )
```

# TAS PRIORITÉ MIN

## TAS MIN ARBRE BINAIRE

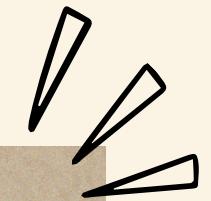
```
tas arbre apres ajout:  
( 0 0 0 1  
    Gauche: ( 35 43 34 763  
        Gauche: ( 75786 68 25 351  
            Gauche: vide  
            Droite: vide )  
        Droite: ( 35 585 575 34  
            Gauche: vide  
            Droite: vide ) )  
    Droite: ( 0 2 0 8  
        Gauche: ( 54 58 0 8  
            Gauche: vide  
            Droite: vide )  
        Droite: ( 7 68 25 351  
            Gauche: vide  
            Droite: vide ) ) )
```

## EXEMPLE SUPPRIMER

```
Supprimer le minimum de tas1 :  
( 0 2 0 8  
    Gauche: ( 35 43 34 763  
        Gauche: ( 75786 68 25 351  
            Gauche: vide  
            Droite: vide )  
        Droite: ( 35 585 575 34  
            Gauche: vide  
            Droite: vide ) )  
    Droite: ( 7 68 25 351  
        Gauche: ( 54 58 0 8  
            Gauche: vide  
            Droite: vide )  
        Droite: vide ) )
```

# TAS PRIORITÉ MIN

## TAS MIN ARBRE BINNAIRE



## EXEMPLE CONSTRUCTION

```
Construction du tas2 :  
Liste de cle : 76 534 345 265 -> 123 8345 3543 453 -> 38 245 34554 763 -> 212 68 25 351 -> 235 2 0 727 -> 77337 0 0 1 -> 35 43 0 763 -> NULL
```

pseudo code:  
ajout simple de la liste de tout  
les clé dans le tas  
applique la fonction descendre  
pour chaque niveau: chaque  
noeud de hauteur  $h-1$ , puis  $h-2$   
... jusqu'à la racine

```
tas arbre apres ajout simple:  
( 76 534 345 265  
  Gauche: ( 123 8345 3543 453  
    Gauche: ( 212 68 25 351  
      Gauche: vide  
      Droite: vide )  
    Droite: ( 235 2 0 727  
      Gauche: vide  
      Droite: vide ) )  
  Droite: ( 38 245 34554 763  
    Gauche: ( 77337 0 0 1  
      Gauche: vide  
      Droite: vide )  
    Droite: ( 35 43 0 763  
      Gauche: vide  
      Droite: vide ) ) )
```

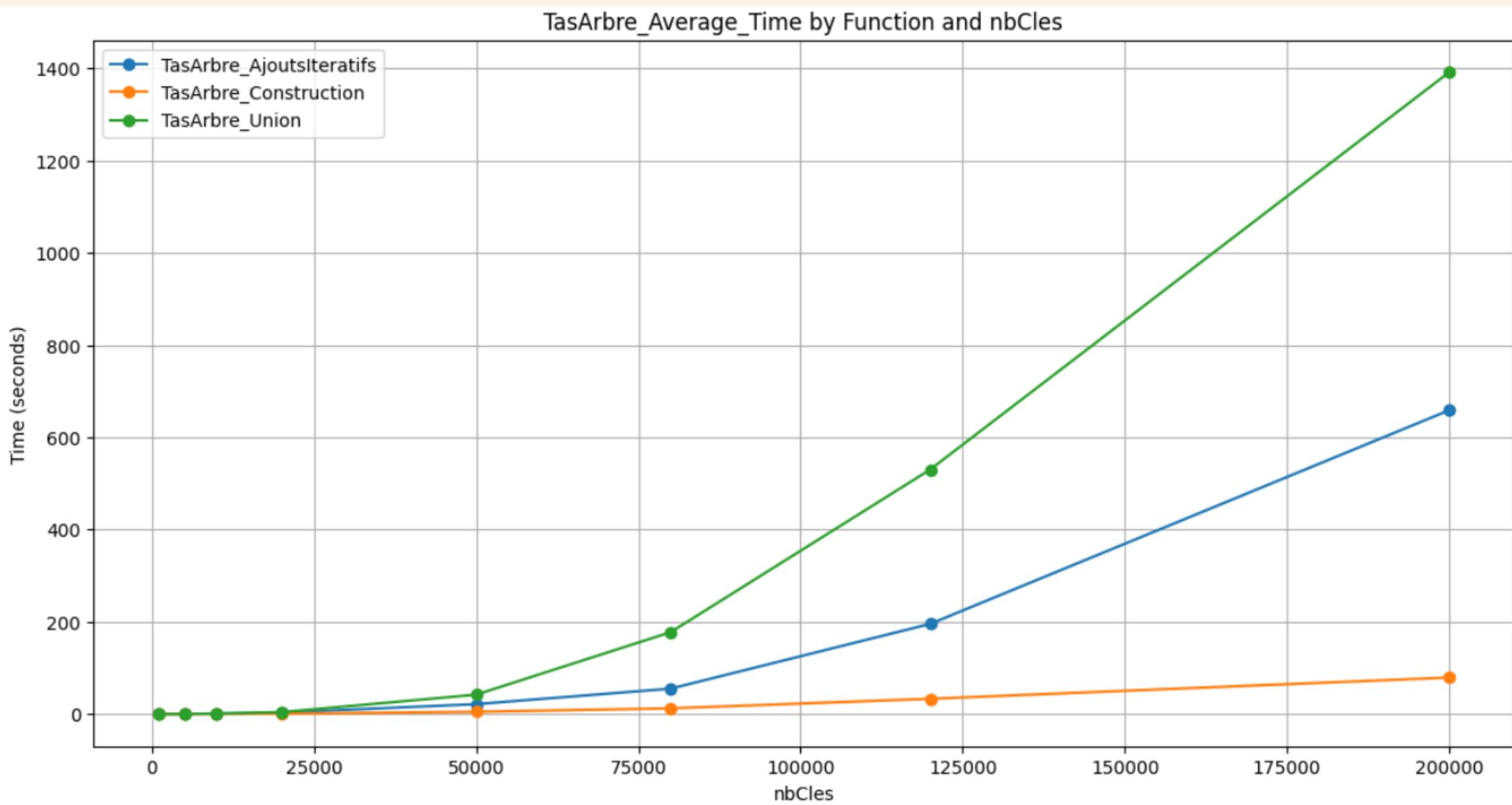
```
tas arbre apres application de descendre au profondeur 1 :  
( 76 534 345 265  
  Gauche: ( 123 8345 3543 453  
    Gauche: ( 212 68 25 351  
      Gauche: vide  
      Droite: vide )  
    Droite: ( 235 2 0 727  
      Gauche: vide  
      Droite: vide ) )  
  Droite: ( 35 43 0 763  
    Gauche: ( 77337 0 0 1  
      Gauche: vide  
      Droite: vide )  
    Droite: ( 38 245 34554 763  
      Gauche: vide  
      Droite: vide ) ) )
```

# TAS PRIORITÉ MIN

## TAS MIN ARBRE BINNAIRE

1. remonter cherche plusieurs fois  
dans l'arbre pour trouver le parent d'un noeud

2. chercher dernier noeud  
(noeud plus à bas et plus à gauche)  
-> mauvaise complexité



# FILES BINOMIALES

```

struct Node {
    uint128_t cle;
    int degree;
    struct Node *parent;
    struct Node *enfant;
    struct Node *frere;
};

typedef struct Node * BinNode;
// Binomial Heap
struct Heap {
    int size;
    int capacity;
    BinNode *list;
};

typedef struct Heap * BinHeap;
typedef struct Node * BinTree;

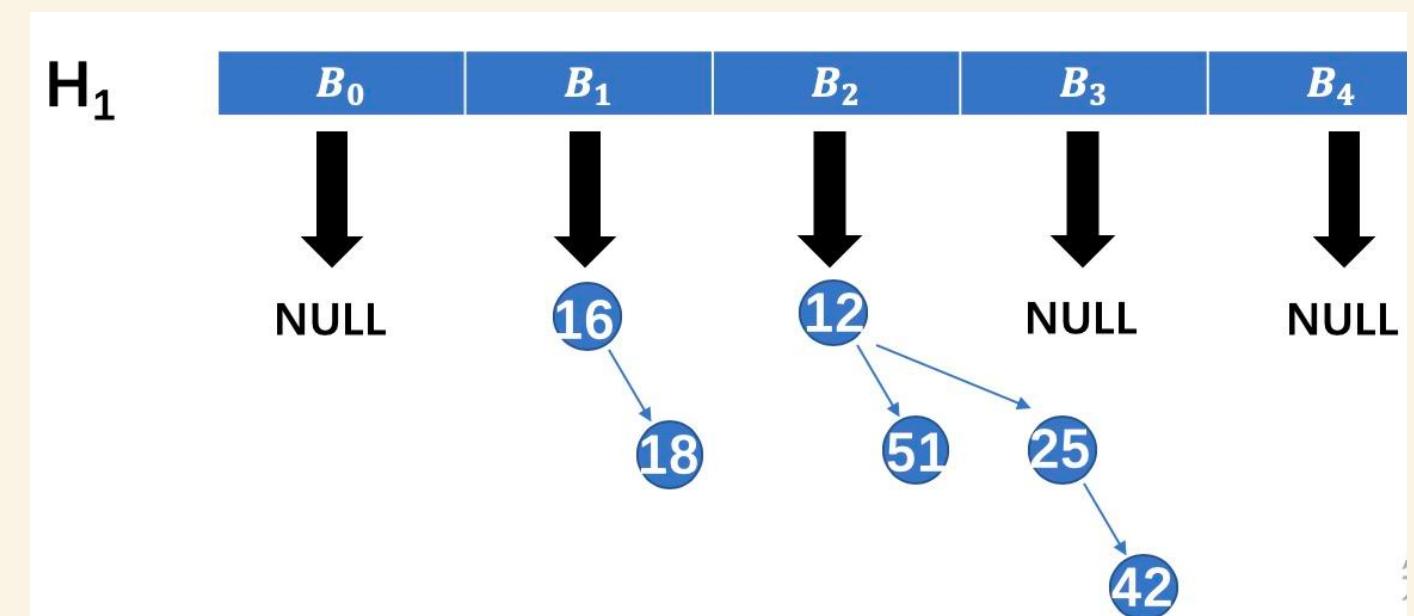
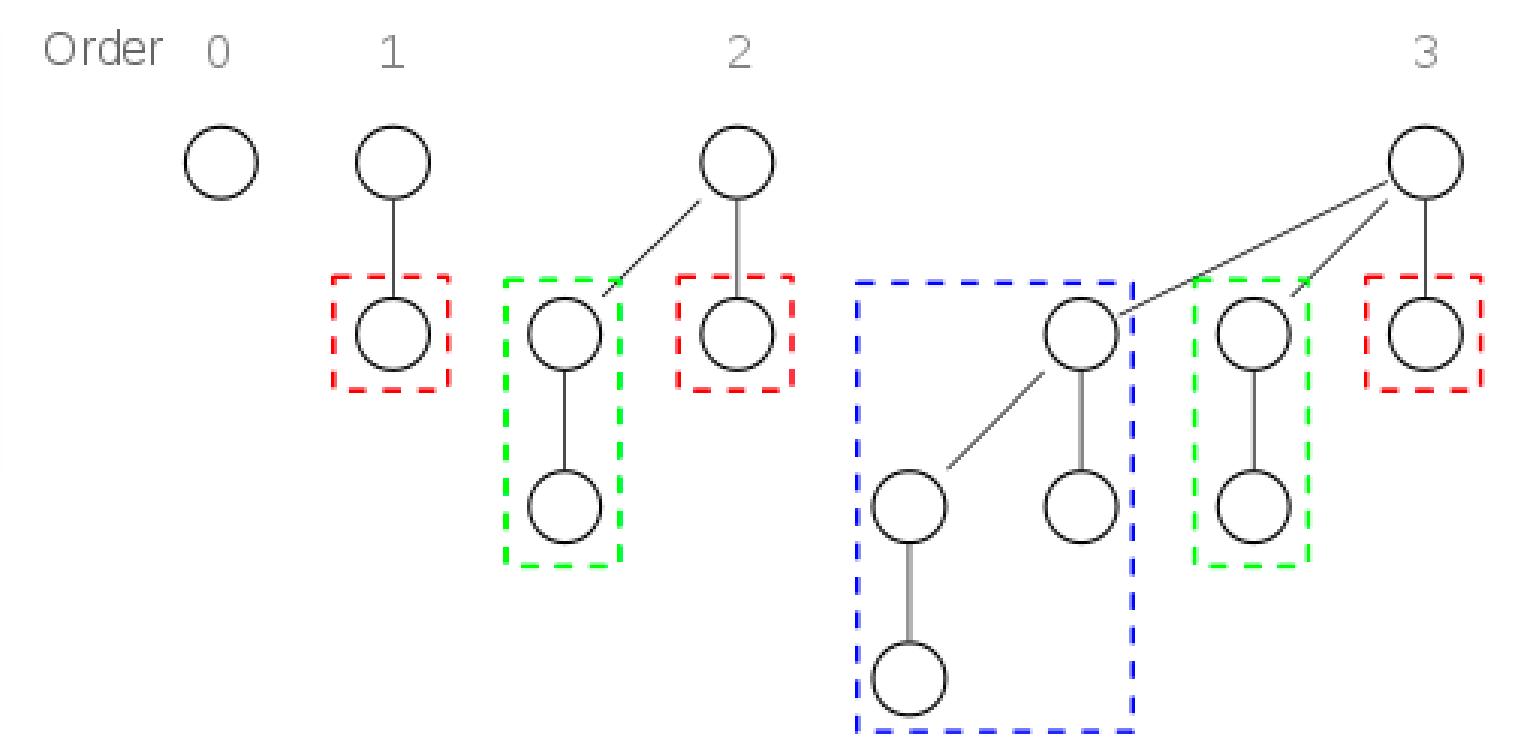
```

```
void fb_union(BinHeap H1, BinHeap H2) ;
```

```
void ajout(BinHeap H, uint128_t Cle);
```

```
uint128_t fb_SupprMin(BinHeap H);
```

```
BinHeap fb_Construction(uint128_t *elements, int n);
```



# FILES BINOMIALES

\*H1\*

Binomial Heap:

Tree of degree 1:

```
----- clé_128 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 -----  
df5d8018 d0af5d1a 979d449c 91282bfc
```

SupprMin\_\_\*H1\*:  
1573c8d1 56d03e63 3c20c36f 01b70862

Tree of degree 2:

```
----- clé_128 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 -----  
5f003a25 87337655 af8a166b e8439a49
```

Tree of degree 0:

```
----- clé_128 -----  
df5d8018 d0af5d1a 979d449c 91282bfc
```

Tree of degree 2:

```
----- clé_128 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 -----  
5f003a25 87337655 af8a166b e8439a49
```

ajout\_\_\*H1\*

\*H2\*

Binomial Heap:

Tree of degree 1:

```
----- clé_128 -----  
1233be51 326a1b2b 9cc45caa 867d053b  
----- clé_128 -----  
d0021c8d a304b898 c87101ae bbce50d1
```

fb\_union\_\_(\*H1\* \*H2\*):

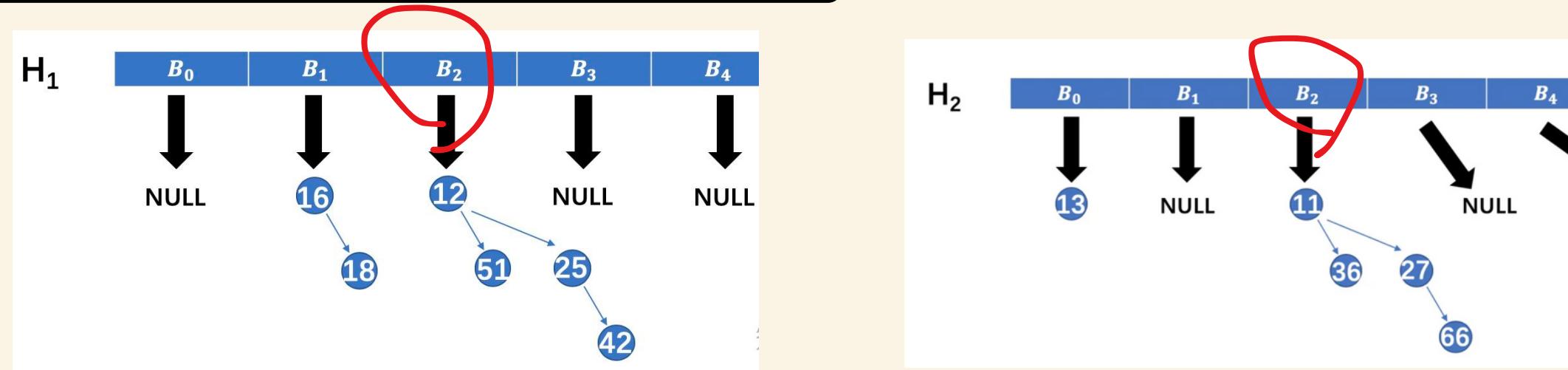
Binomial Heap:

Tree of degree 3:

```
----- clé_128 -----  
1233be51 326a1b2b 9cc45caa 867d053b  
----- clé_128 -----  
2c15aed1 a9eab933 38d0348f 12ef9a3b  
----- clé_128 -----  
d192acf4 c06fe7c7 df042f07 d290bdd4  
----- clé_128 -----  
df6943ba 6d51464f 6b021579 33bdd9ad  
----- clé_128 -----  
5f003a25 87337655 af8a166b e8439a49  
----- clé_128 -----  
1573c8d1 56d03e63 3c20c36f 01b70862  
----- clé_128 -----  
df5d8018 d0af5d1a 979d449c 91282bfc
```

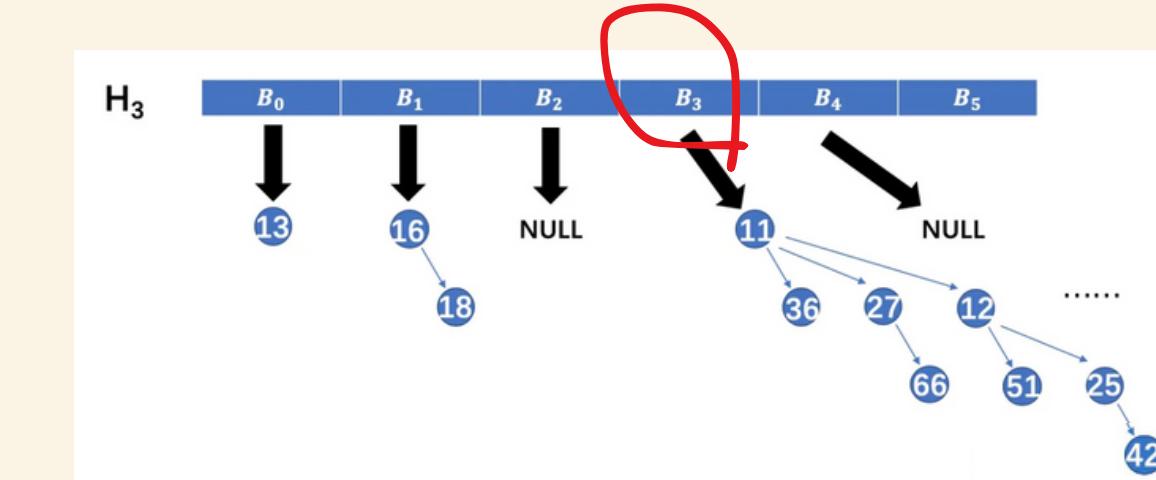
# FILES BINOMIALES

FB\_UNION



$O(\log n)$

**nb de arbre  $\leq \log n$**



**1. Étendre la capacité de  $H1$  (si nécessaire)**

**2. Carry : pour stocker temporairement les résultats de la fusion**

**3. Parcourir et fusionner : chaque arbre dans  $H1$  et  $H2$  à chaque indice**

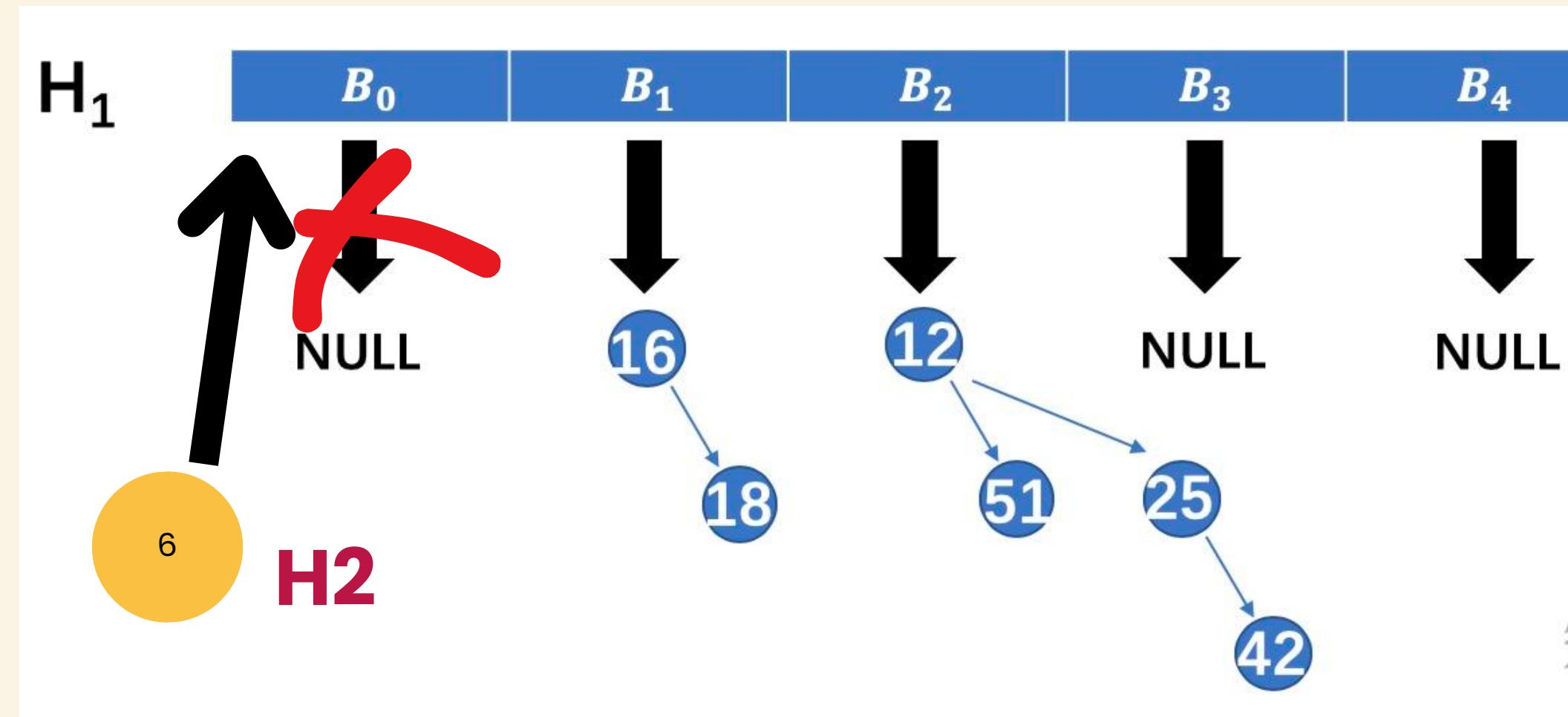
**4. Mise à jour et Libération**

# FILES BINOMIALES

AJOUT

$O(\log n)$

nb de arbre  $\leq \log n$



1. Vérifier si étendre la capacité

2. Créer un arbre binomial avec un seul nœud

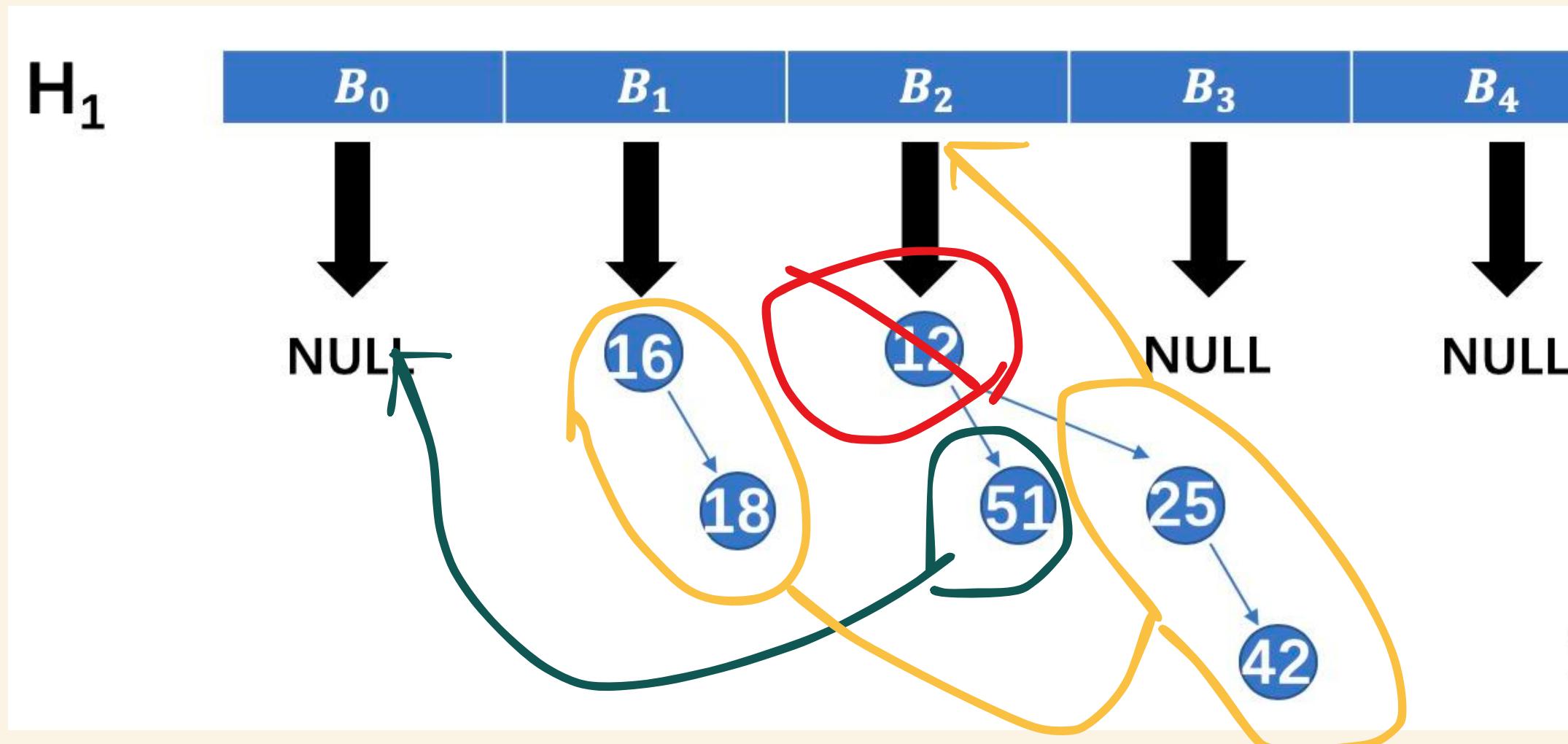
3. Initialiser le `tempHeap` et y insérer le nouveau nœud

4. Utiliser la `fb_union`

# FILES BINOMIALES

FB\_SUPPRMIN

$O(\log n)$



1. Parcourir les racines des arbres du tas binomial

2. Suppression du Minimum

3. Fusion

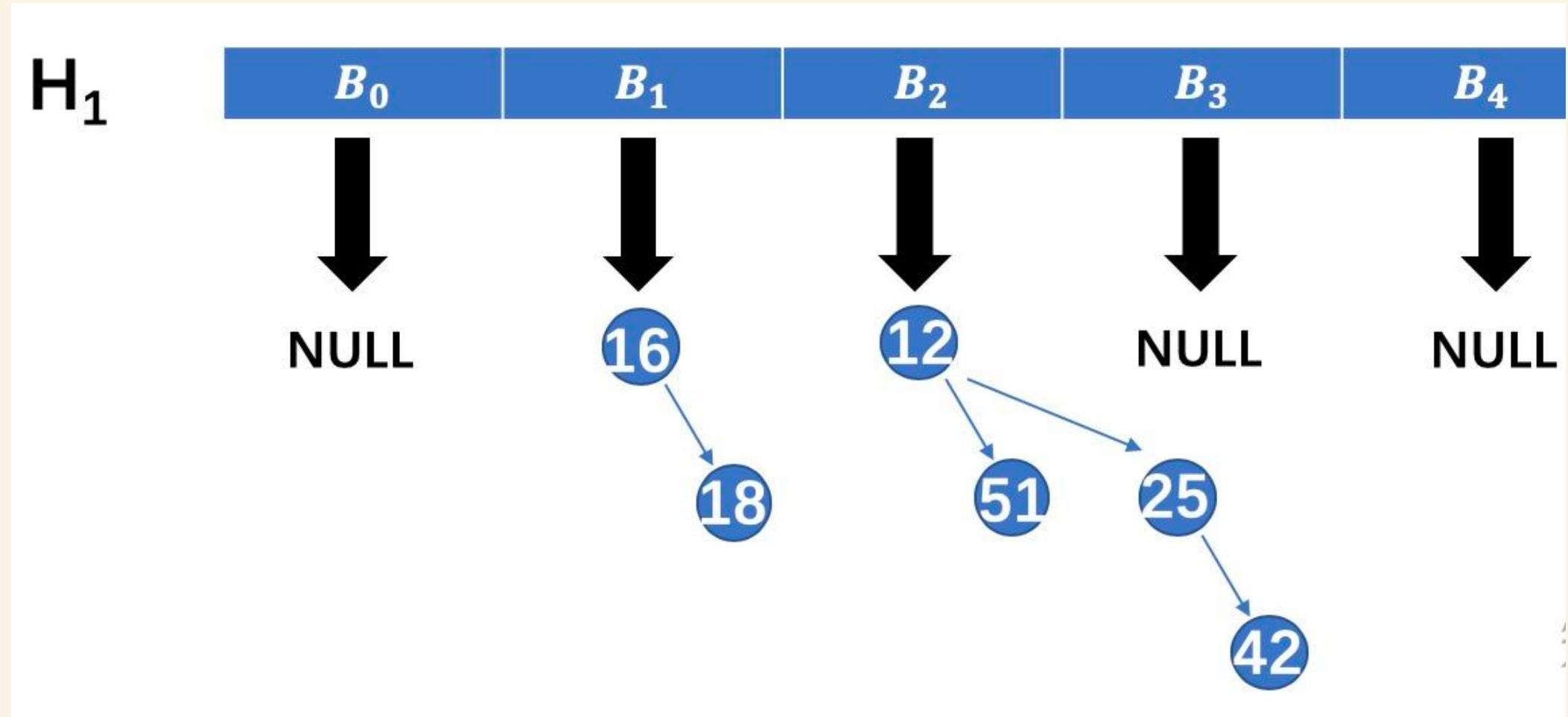
4. Mise à jour et Libération

# FILES BINOMIALES

## CONSTRUCTION

$O(n \log n)$

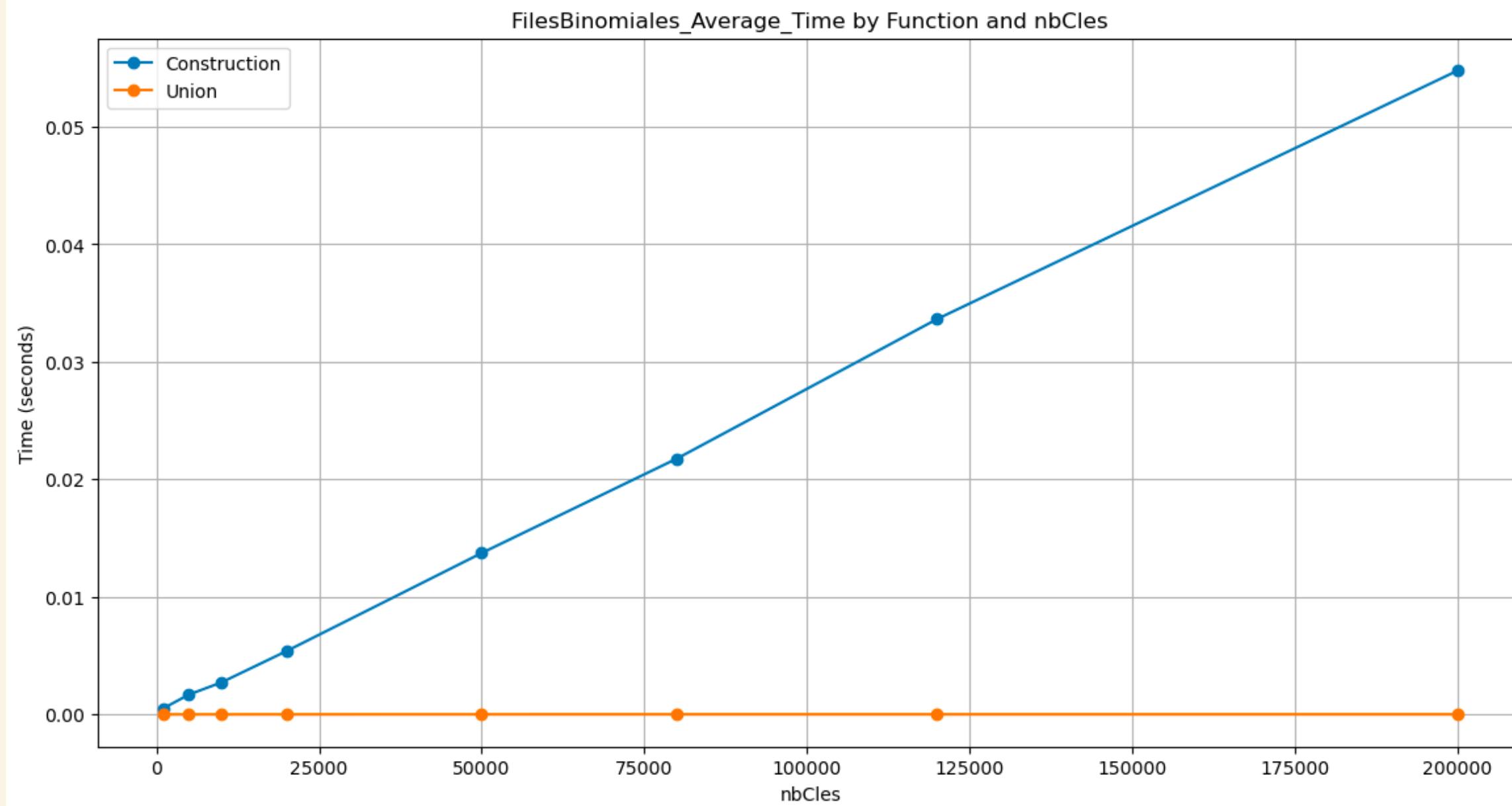
nb de arbre  $\leq \log n$



1. Initialisation du file binomial

2. Ajouter des éléments

# FILES BINOMIALES



performance\_files\_binomiales.csv

Function	nbCles	Time
Construction	1000	0.000258
Union	1000	0.000001
Construction	1000	0.000265
Union	1000	0.000001
Construction	1000	0.000238
Union	1000	0.000001
Construction	1000	0.000248
Union	1000	0.000001
Construction	1000	0.000277
Union	1000	0.000000
Construction	5000	0.001285
Union	5000	0.000001
Construction	5000	0.001296
Union	5000	0.000002
Construction	5000	0.001302
Union	5000	0.000001

# FONCTION DE HACHAGE

# **Etape :**

# Conversion du message en tableau de octect

# Padding (remplissage de bit)

# Découpage en blocs de 512 bits

Subdivision de chaque bloc en 16 mots de 32 bits

# Boucle principal qui fait les opérations

# Bonne réponse :

ed076287532e86365e841e92bfc50d8c

`md5("Hello World!") =`

876207ed36862e53921e845e8c0dc5bf

## FONCTION DE HACHAGE

Idée pour trouver la liste des différent mot du livre :  
AbrExistant contenant tout les mots différent du livre Shakespear  
Liste contenant tout les mots du livre Shakespear

# ARBRE DE RECHERCHE

```
typedef struct ABR{
    uint128_t cle;
    struct ABR *gauche;
    struct ABR *droite;
}ABR;
```

Liste de cle a ajouter dans l'ABR :

Liste de cle : 46 68 25 351 -> 5 58 0 8 -> 91 585 575 34 -> 35 43 34 763 -> 842 68 25 351 -> 76 2 0 8 -> 6115 0 0 1 -> NULL

Arbre :

```
Noeud( 46 68 25 351
      Gauche: Noeud( 5 58 0 8
                  Gauche: vide
                  Droite: Noeud( 35 43 34 763
                                  Gauche: vide
                                  Droite: vide ) )
      Droite: Noeud( 91 585 575 34
                      Gauche: Noeud( 76 2 0 8
                                      Gauche: vide
                                      Droite: vide )
                      Droite: Noeud( 842 68 25 351
                                      Gauche: vide
                                      Droite: Noeud( 6115 0 0 1
                                          Gauche: vide
                                          Droite: vide ) ) ) )
```

08

Cheche une cle existant 46 68 25 351 : true

Cheche une cle inexistant 35 43 0 763 : false

20

**MERCI DE VOTRE ATTENTION**

---

Ariane ZHANG

Xue YANG

---