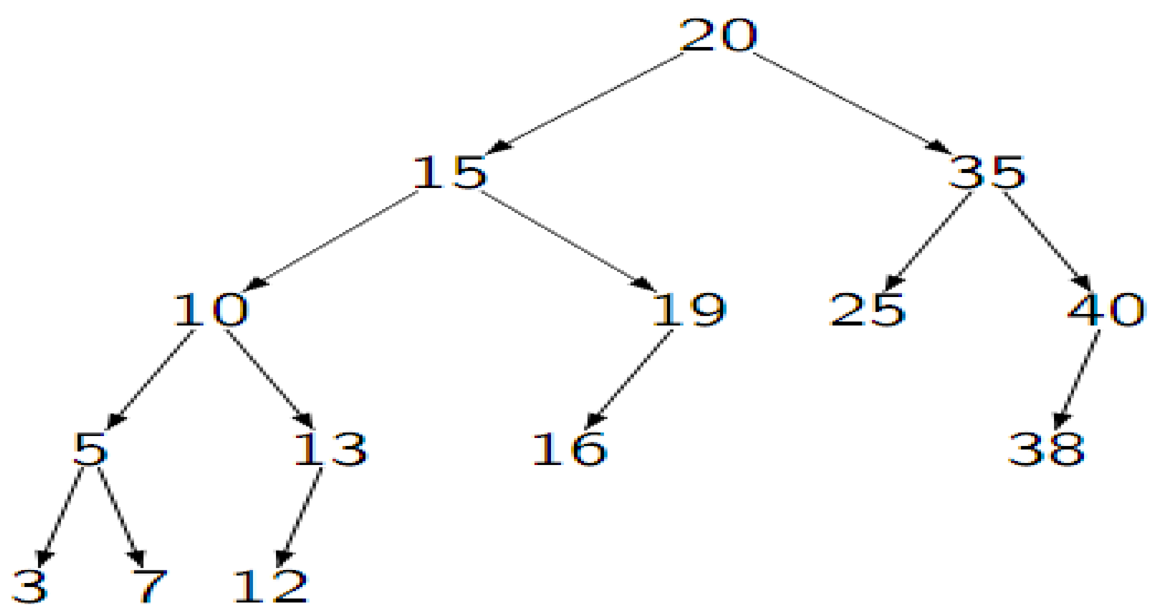
ALGORITHMES DE TRI

Master : Cyber Sécurité et Intelligence Artificielle
CSIA

Année universitaire : 2025 / 2026

Tri par ABR

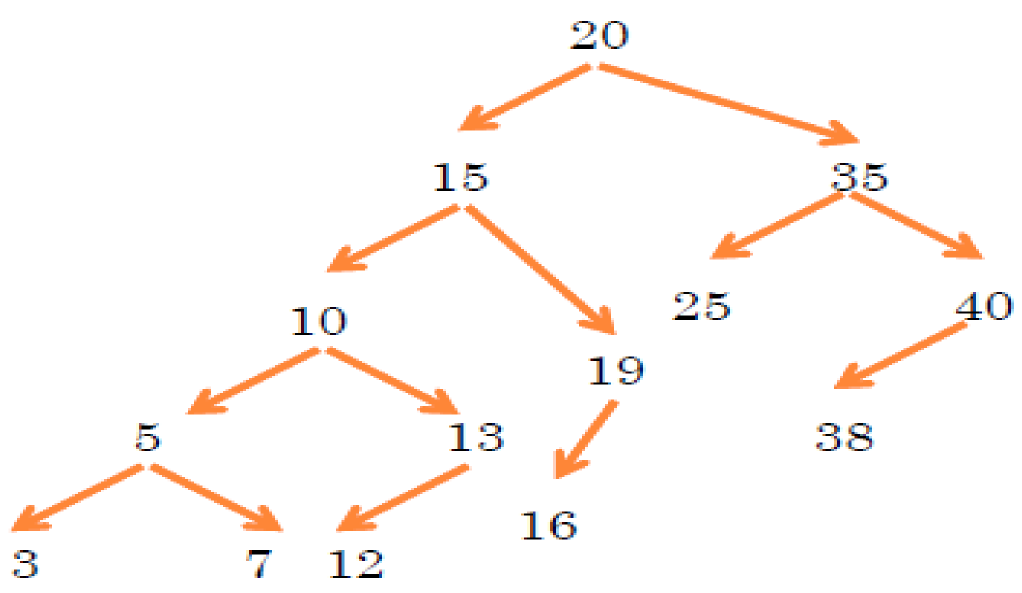
- Un Arbre Binaire de Recherche (ABR) est un arbre binaire, dans lequel chaque nœud contient un entier en respectant la propriété suivante :
  - Inférieur (ou égal) aux entiers de son sous-arbre gauche
  - Supérieur strictement aux entiers de son sous-arbre droit



Tri par ABR

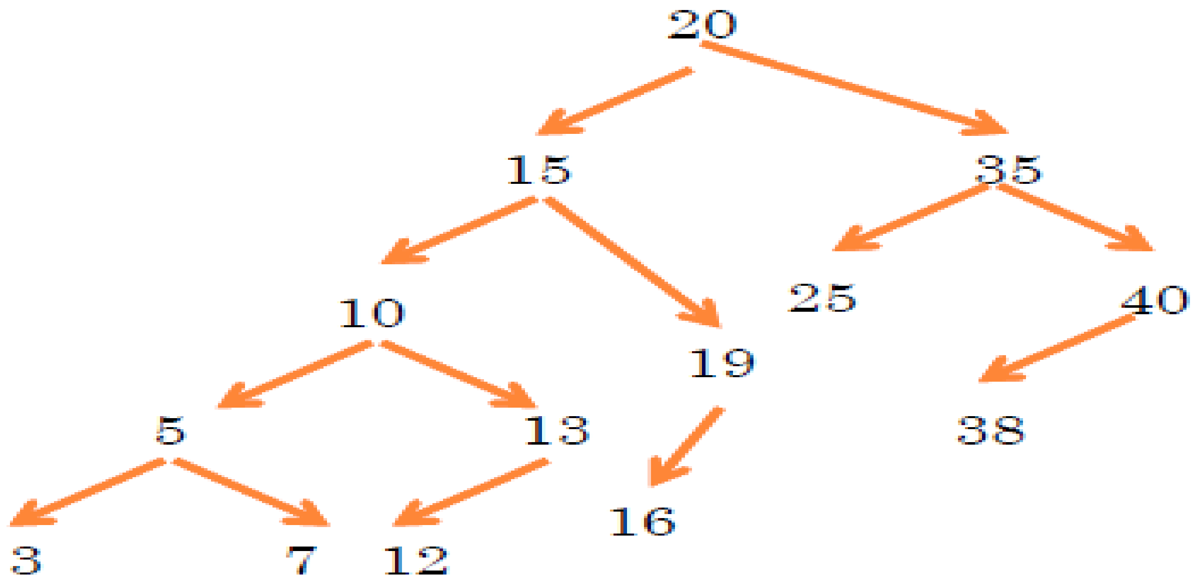
- **Insérer toutes les éléments du tableau dans un ABR**

20	15	10	35	19	13	5	3	12	7	16	40	25	38



Tri par ABR

- **Parcourir l'ABR en ordre infixé: *fils gauche, noeud, fils droit***



3	5	7	10	12	13	15	16	19	20	25	35	38	40

Tri par ABR

Soit AR un arbre de recherche binaire

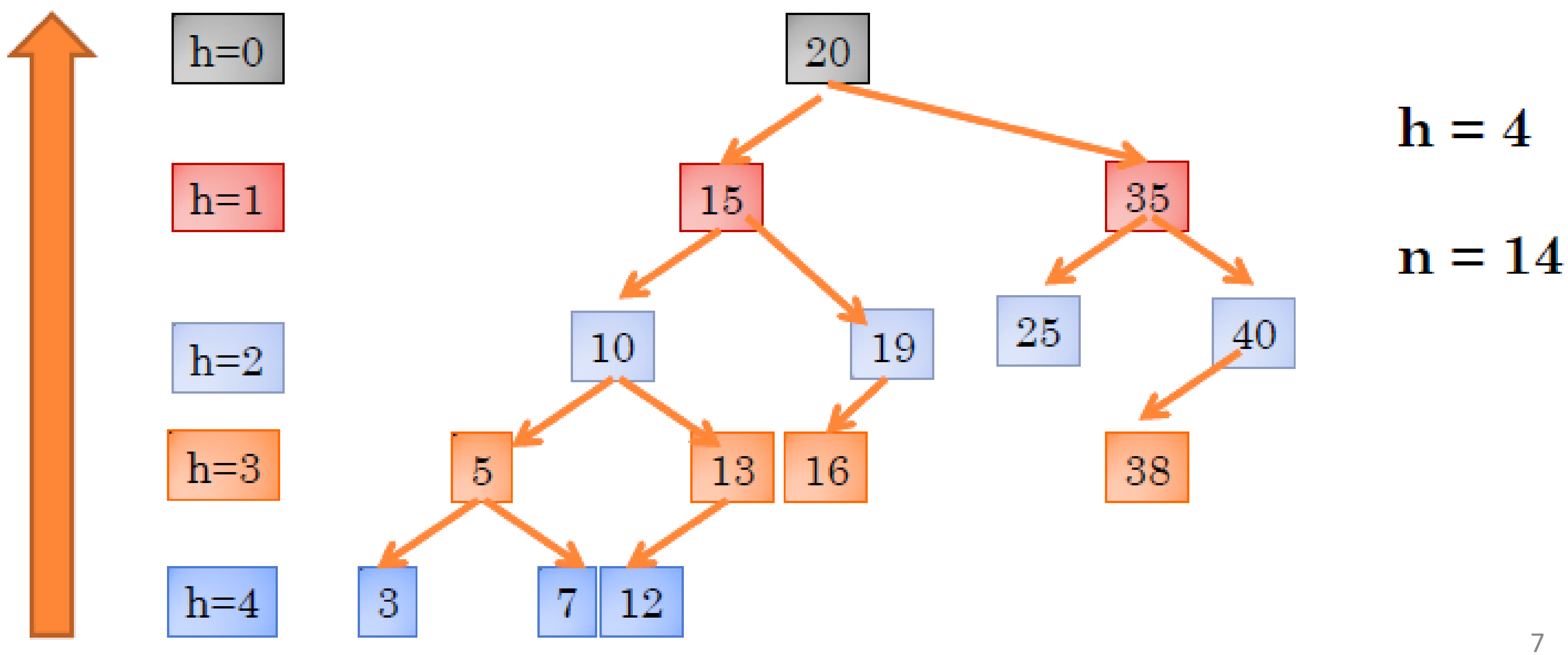
```
Tri_ARB_IT (T: Tableau, n: entier)
Debut
// Construire ARB
AR←Nil
Pour i←1 à n faire
    AR←Insérer (AR, T[i]).
FinPour
Parcours_Infixe (AR, T); //Parcours Infixe
Fin
```

Tri par ABR
Fonction insérer

```
Insérer (AR: noeud , x: entier): noeud
Debut
SI (AR=Nil) alors // L'arbre est vide
    AR← Noeud(x,Nil,Nil) // Créer la racine (le premier noeud)
SINON
SI x ≤ valeur(AR) alors
    AR.FG←Insérer(FG(AR), x) // Insérer à gauche
SINON
    AR.FD←Insérer(FD(AR), x) // Insérer à droite
FSI
Insérer ← AR
Fin
```

Tri par ABR

- Si **h** est la hauteur d'un arbre binaire de recherche, et **n** le nombre des noeuds (ou la taille du tableau)



Tri par ABR
Complexité de la fonction insérer

Insérer (AR: noeud , x: entier): noeud

Debut — Tinsérer(h) tq h est la hauteur de l'arbre

SI (AR=Nil) alors *// L'arbre est vide*

    AR ← Noeud(x,Nil,Nil) *// Créer la racine (le premier noeud)*

SINON

SI x ≤ valeur(AR) alors

    AR.FG←Insérer(FG(AR), x) *// Insérer à gauche* Tinsérer(h+1)

SINON

    AR.FD←Insérer(FD(AR), x) *// Insérer à droite* Tinsérer(h+1)

FSI

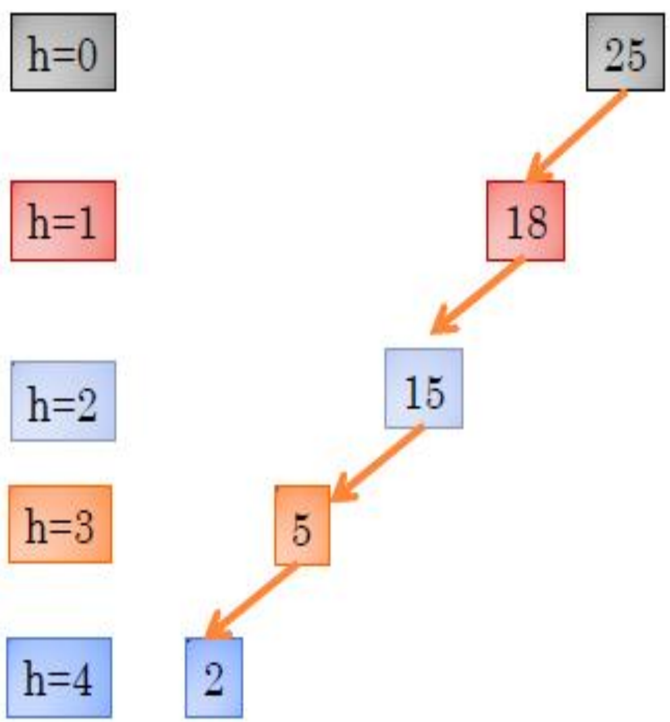
Insérer ← AR

Fin

Tinsérer(h) = Tinsérer(h+1) + c → O(Tinsérer) = O(h)

Tri par ABR
Complexité de la fonction insérer

- **Pire de cas:** un tableau déjà trié → Arbre mal équilibré → h = n-1 → O(Tinsérer) = O(n-1) = O(n)



25	18	15	5	2

Tri par ABR
fonction Parcours_Infixe

Indice : une variable initilisé à 1

```
Parcours_Infixe (AR: noeud, T: Tableau, indice)
Debut
Si ( AR != Nil) alors   //Arbre n'est pas vide
   Parcours_Infixe(FG(AR), T, indice)
   T[indice]←valeur (AR)  //Écrire la valeur dans le tableau
    indice ← indice + 1;
    Parcours_Infixe(FD(AR), T, indice)
FSI
Fin
```

Tri par ABR
complexité de la fonction Parcours_Infixe

Indice : une variable initilisé à 1

Parcours_Infixe (AR: noeud, T: Tableau, indice) Tinfixe(n)

Debut

Si (AR != Nil) alors *//Arbre n'est pas vide*

  Parcours_Infixe(FG(AR), T, indice) Tinfixe(k)

  T[indice]←valeur (AR) *//Écrire la valeur dans le tableau*

  indice ← indice + 1;

  Parcours_Infixe(FD(AR), T, indice) Tinfixe(n-k)

FSI

Fin

Tinfixe(n) = Tinfixe(k) + Tinfixe(n-k) + c → O(Tinfixe) = O(n)

le parcours infixé passe par tous les noeuds de l'arbre

Tri par ABR
Complexité

Tri_ARB_IT (T: Tableau, n: entier) Tarb(n)

Debut

// Construire ARB

AR←Nil

Pour i←1 à n faire n fois

    AR←Insérer (AR, T[i]). Tinsérer(n)

FinPour

Parcours_Infixe (AR, T); *//Parcours Infixe* Tinfixe(n)

Fin

Tarb(n) = n * Tinsérer(n) + Tinfixe(n) →

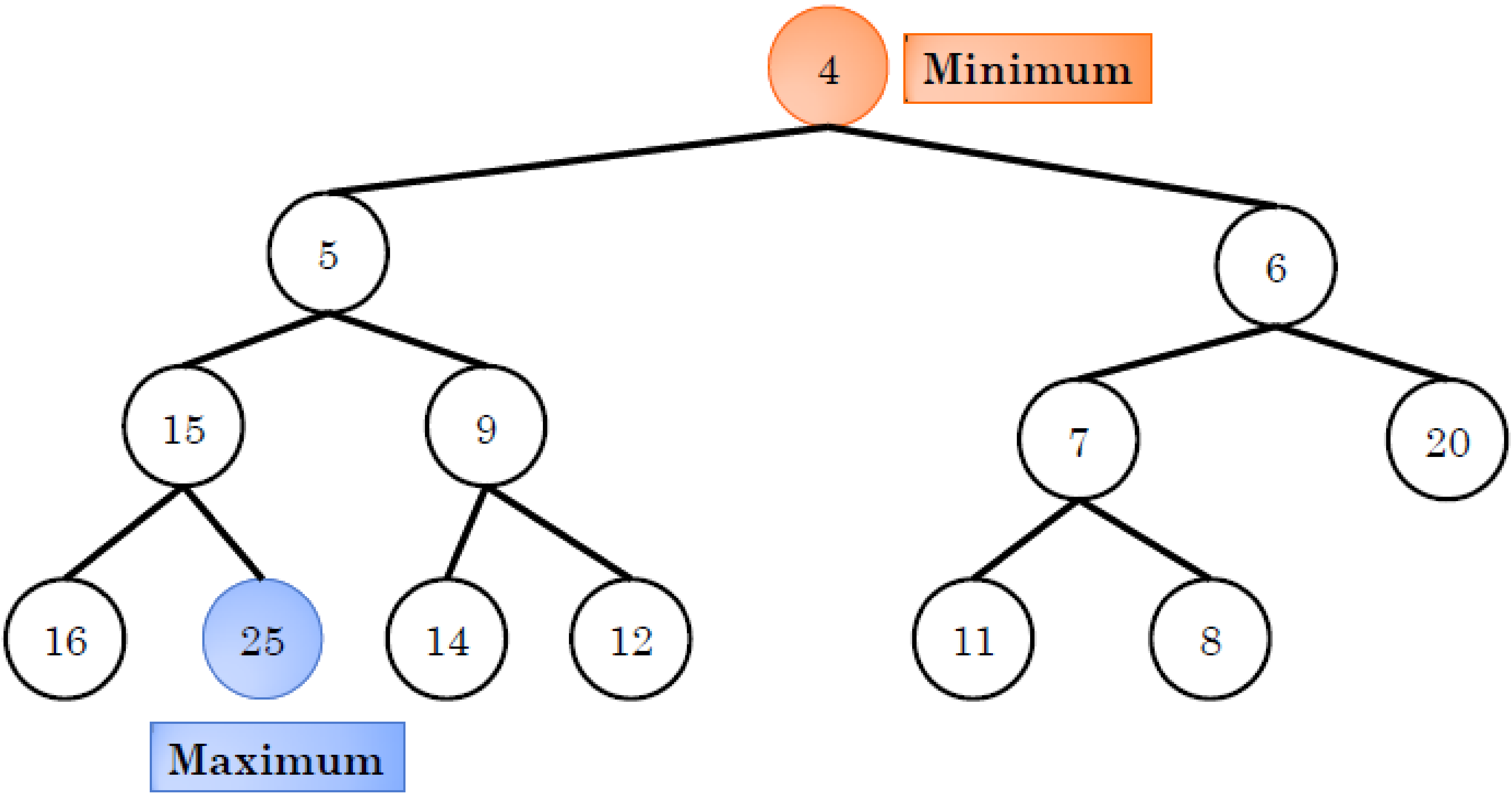
$O(Tarb) = O(n^2) + O(n) = O(n^2)$

Tri par TAS

- Un TAS un arbre binaire qui vérifie les deux propriétés suivantes :
- ***Propriété structurelle:*** arbre binaire complet (ou parfait), i.e. Tous les niveaux sont totalement remplis sauf le dernier qui est rempli de la gauche vers la droite.
- ***Propriété d'ordre :***
- TAS_{min}: valeur (père) <= valeur (fils)
- TAS_{max}: valeur (père) >= valeur (fils)

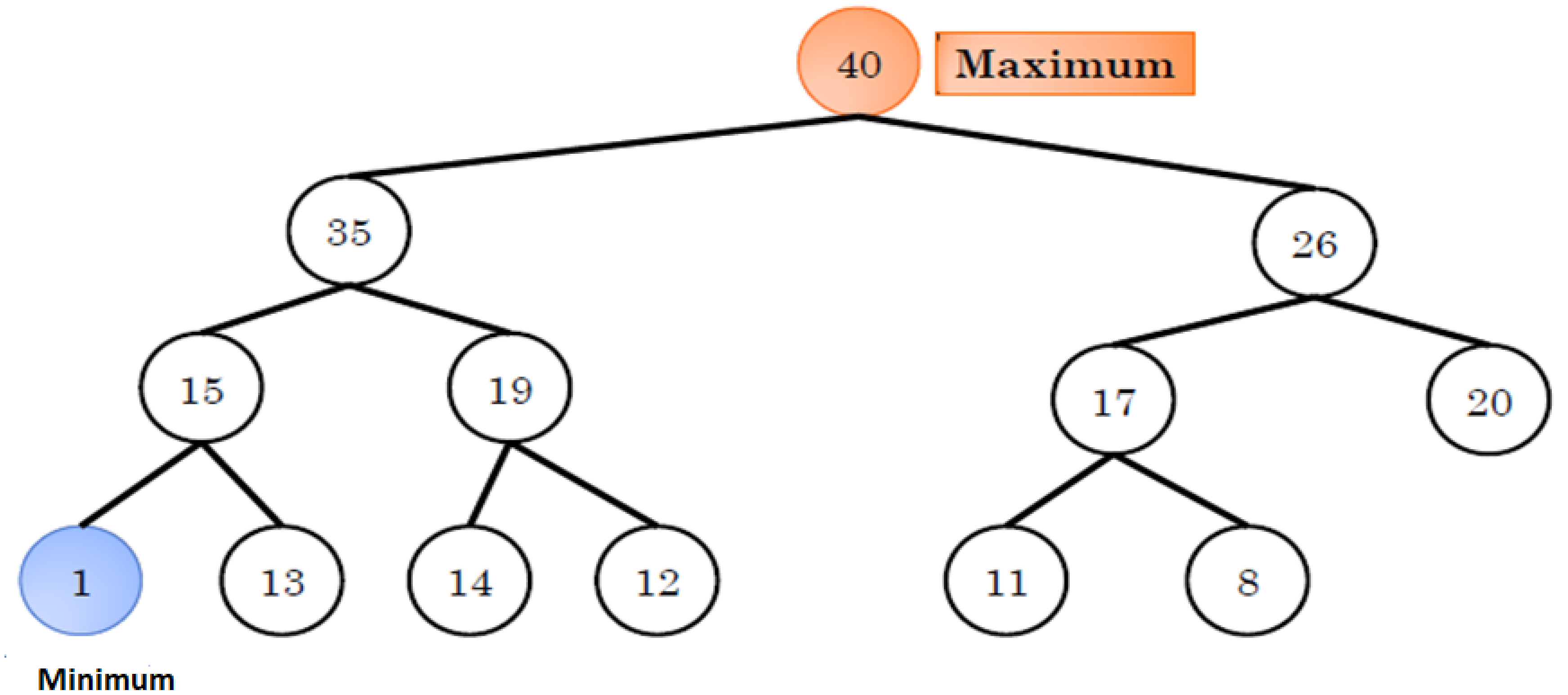
Tri par TAS

TAS_{min}



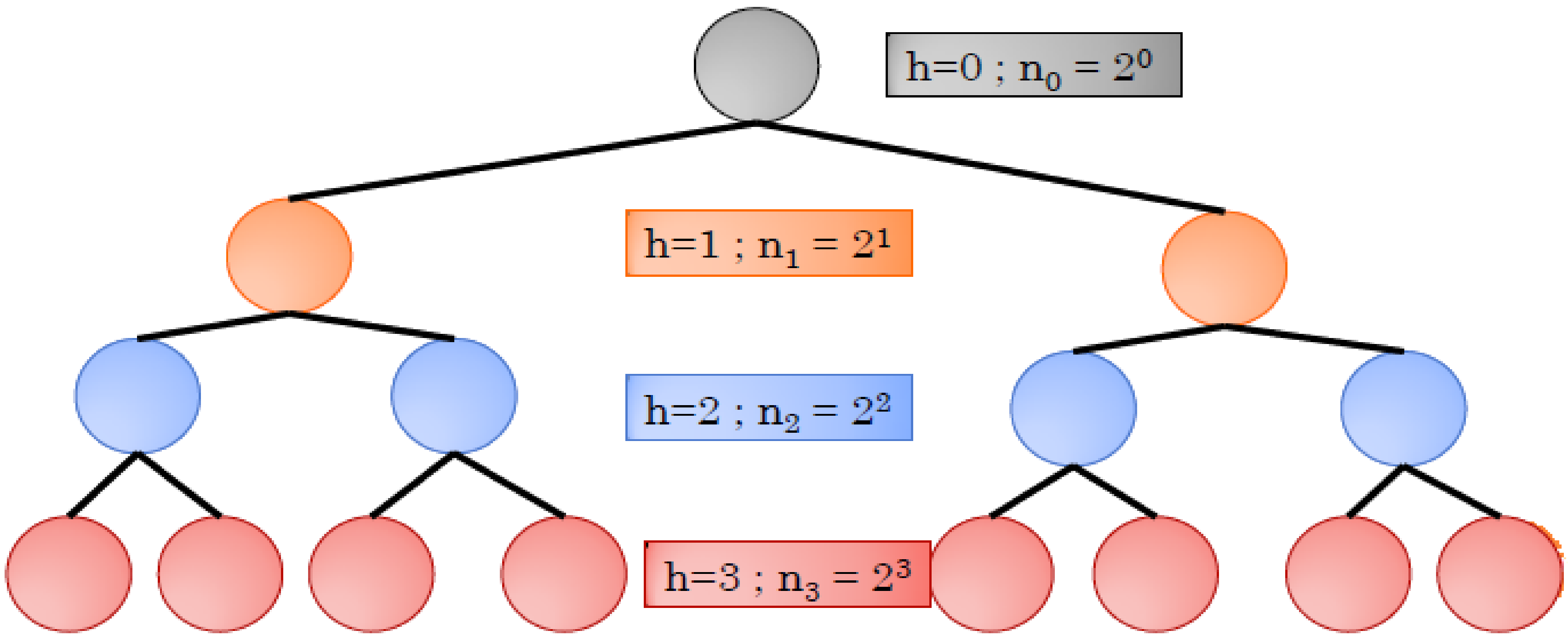
Tri par TAS

TAS_{max}



Tri par TAS

- **Théorème:** Un tas de n nœud a une hauteur O(log2 n)
- **Démonstration**
  - Soit h, la hauteur d'un tas de n noeud



Tri par TAS

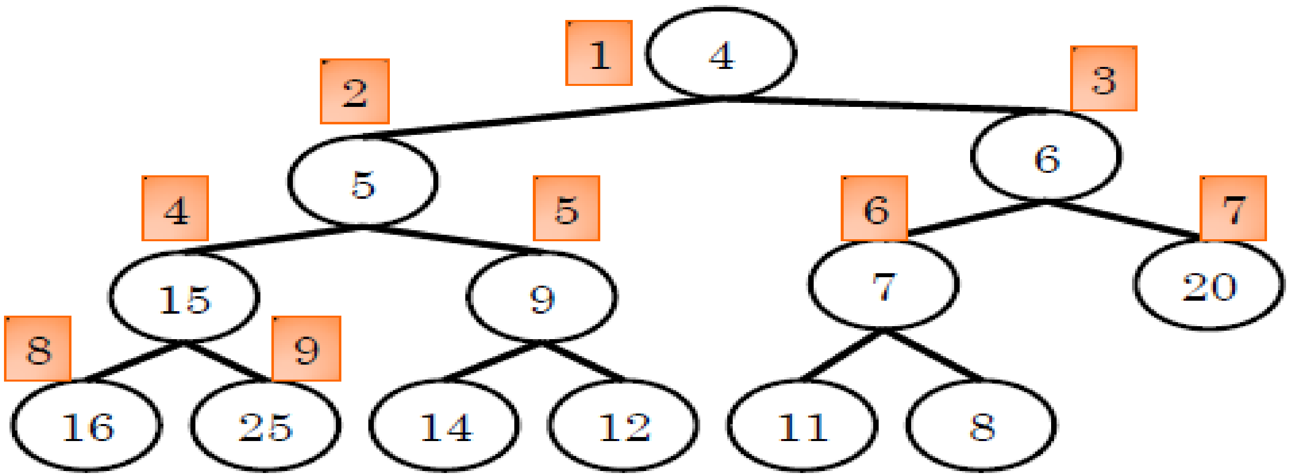
- Soit h, la hauteur d'un tas de n noeud
- Au niveau $i<>h$, $n_i=2^i$
- Donc $n = 2^0 + 2^1 + 2^2 + ...+ 2^{h-1} + c$ tel que , $0<=c<2^h$

$n = 2^h + c >= 2^h \rightarrow h <= \log_2 n \rightarrow O(h)=O(\log_2 n)$

- **Conséquence: Les opérations proportionnelle à h sont O ($\log_2$ n)**

Tri par TAS
REPRÉSENTATION PAR TABLEAU

- Un tas se représente naturellement par un tableau:
  - Les sommets sont numérotés par un parcours en largeur, de gauche à droite.
  - Le sommet « i » est rangé dans la case d'indice i du tableau.



1	2	3	4	5	6	7	8	9	10	11	12	13
4	5	6	15	9	7	20	16	25	14	12	11	8

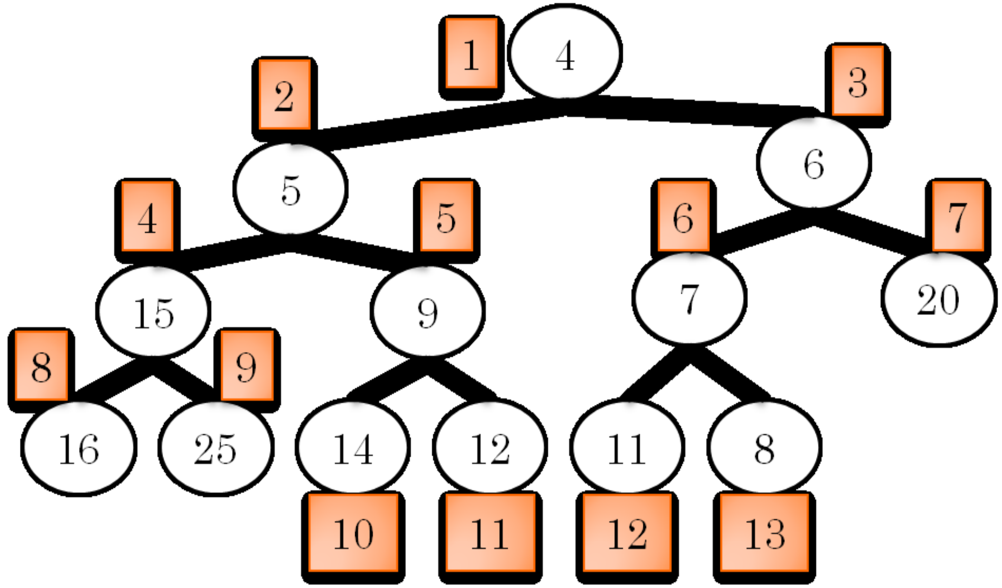
Tri par TAS
REPRÉSENTATION PAR TABLEAU

Indice(racine)=1

Indice(FG)=2*Indice(Père)

Indice(FD)=2*Indice(Père)+1

Indice(Père)= [Indice (Fils)/2]



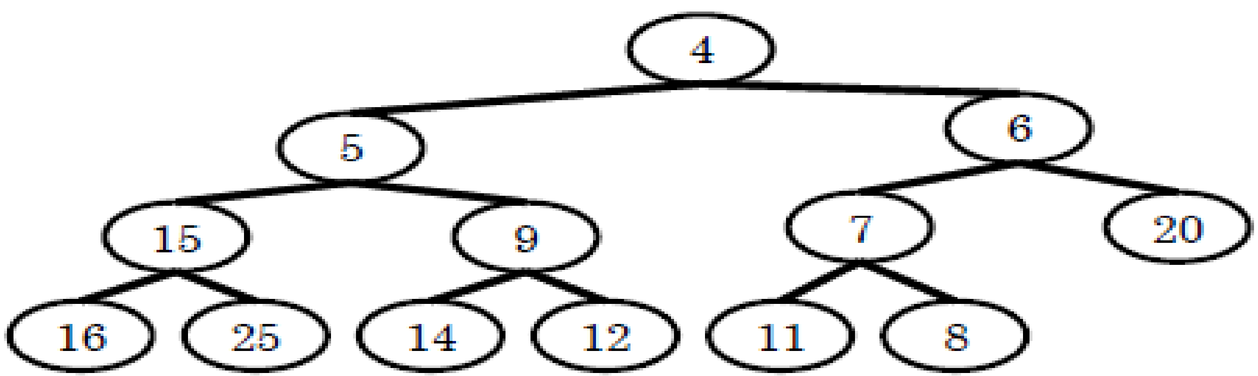
1	2	3	4	5	6	7	8	9	10	11	12	13
4	5	6	15	9	7	20	16	25	14	12	11	8

Tri par TAS_{Min}
INSERTION

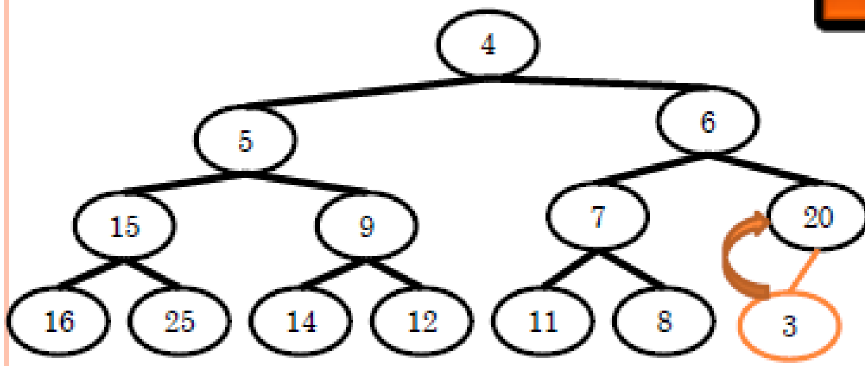
- Pour insérer une valeur « v » dans un TAS_{min}

  1.Insérer « v » à la fin du dernier niveau de l'arbre (à la fin du tableau).

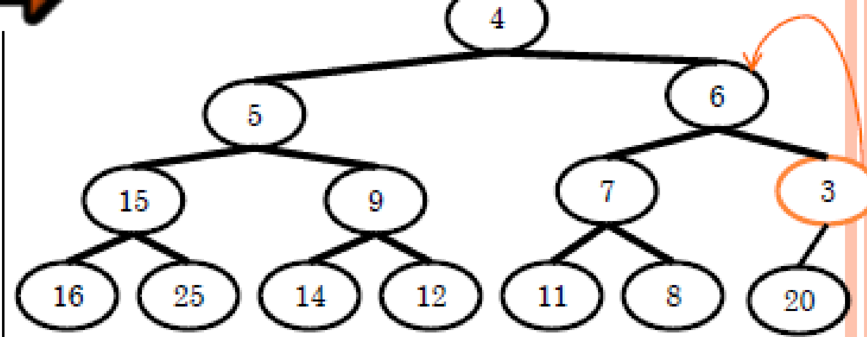
  2.Tant que la valeur du père de « v » est plus grande que « v », échanger la valeur du père de v avec « v ».
- Exemple: insertion de 3



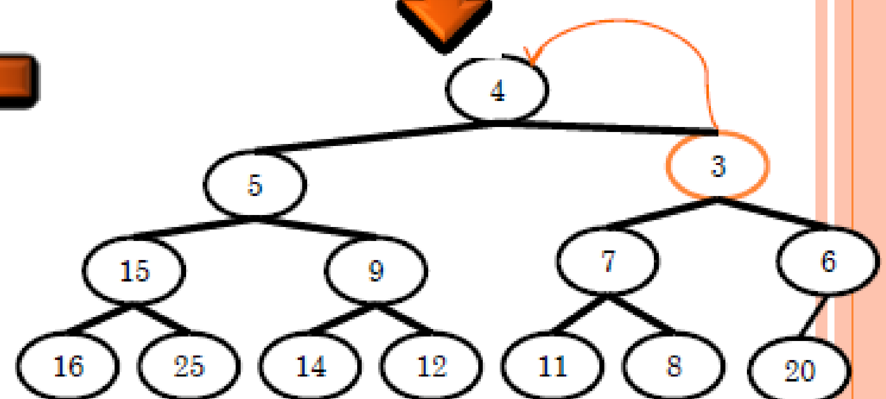
4	5	6	15	9	7	20	16	25	14	12	11	8



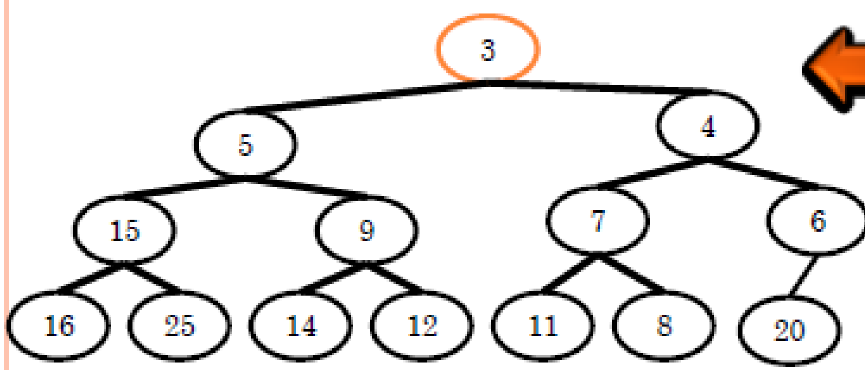
4	5	6	15	9	7	20	16	25	14	12	11	8	3



4	5	6	15	9	7	3	16	25	14	12	11	8	20



4	5	3	15	9	7	6	16	25	14	12	11	8	20



3	5	4	15	9	7	6	16	25	14	12	11	8	20

Tri par TAS_{Min}

INSERTION

Procédure Insérer_TAS (Tas: tableau, n, x: entier)

Début

n $\leftarrow$ n + 1

i $\leftarrow$ n

Tas[i] $\leftarrow$ x

Tant que (i/2 > 0 et Tas[i/2] > x) faire

    Permuter (Tas, i, i/2)

    i $\leftarrow$ i/2

Fin

$O(Tinserer\_TAS(n)) = O(\log_2 n)$

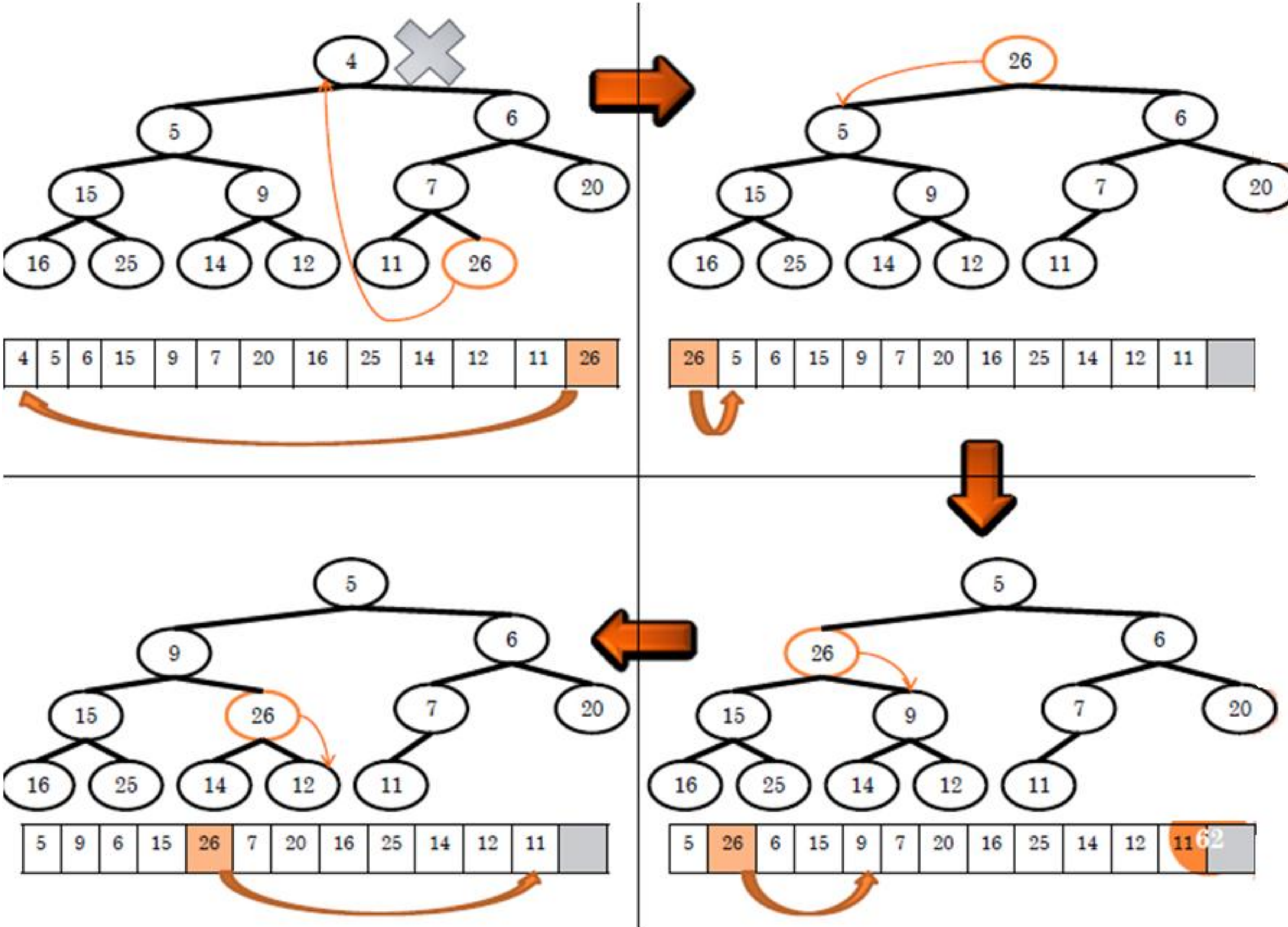
Tri par TAS_{Min}

EXTRAIRE MINIMUM

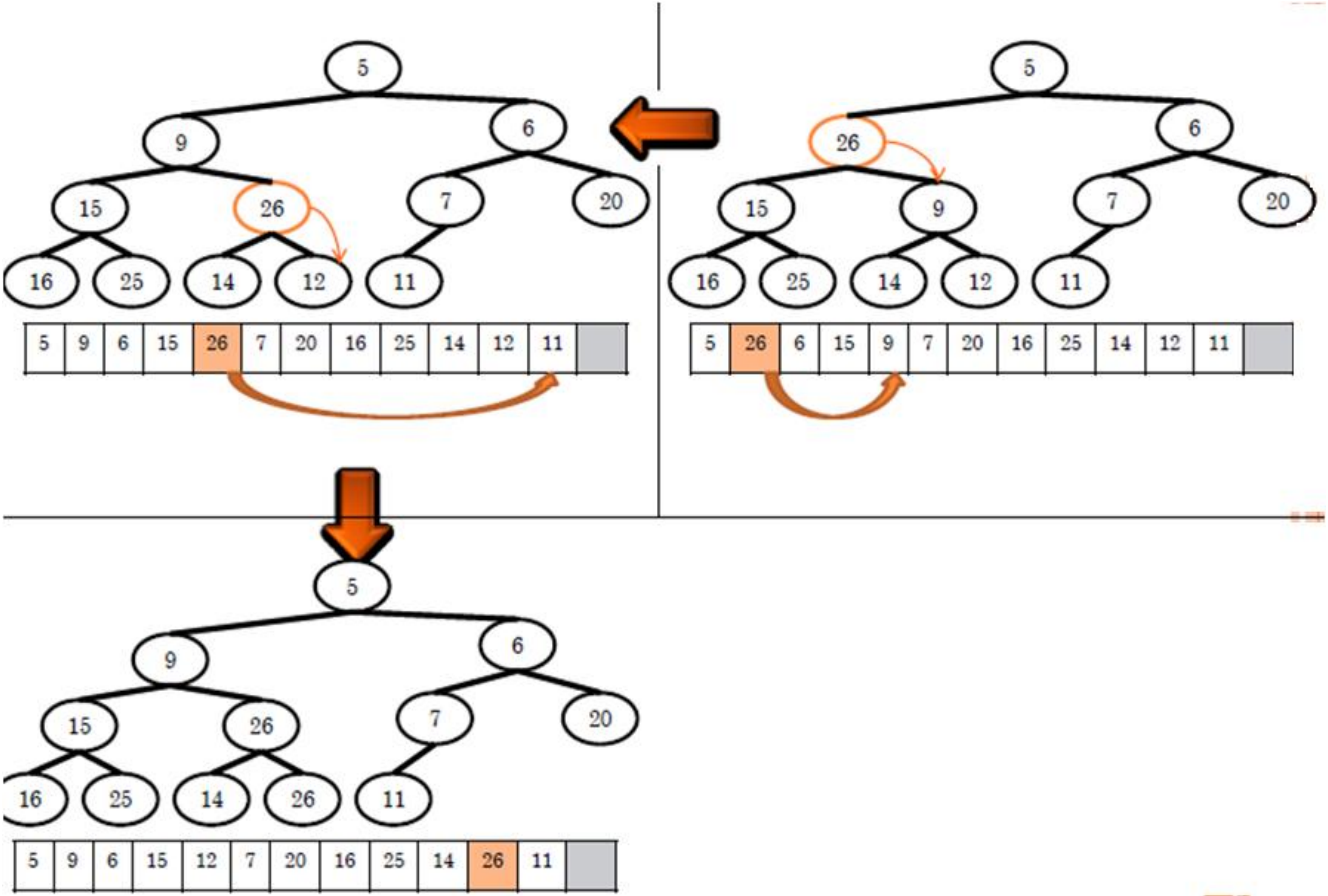
- Le minimum se trouve à la racine.
- Pour supprimer la racine:

1.Remplacer la racine par le dernier élément «v» (à la fin du tableau).

2.Tant que la valeur « v » est supérieure à celle de l'un de ses fils, échanger la valeur «v» avec celle du plus petit de ses fils.



4 5 6 15 9 7 20 16 25 14 12 11 26
26 5 6 15 9 7 20 16 25 14 12 11
5 9 6 15 26 7 20 16 25 14 12 11
5 26 6 15 9 7 20 16 25 14 12 11


5 9 6 15 26 7 20 16 25 14 12 11
5 26 6 15 9 7 20 16 25 14 12 11
5 9 6 15 12 7 20 16 25 14 26 11

Tri par TAS$_{Min}$
EXTRAIRE MINIMUM

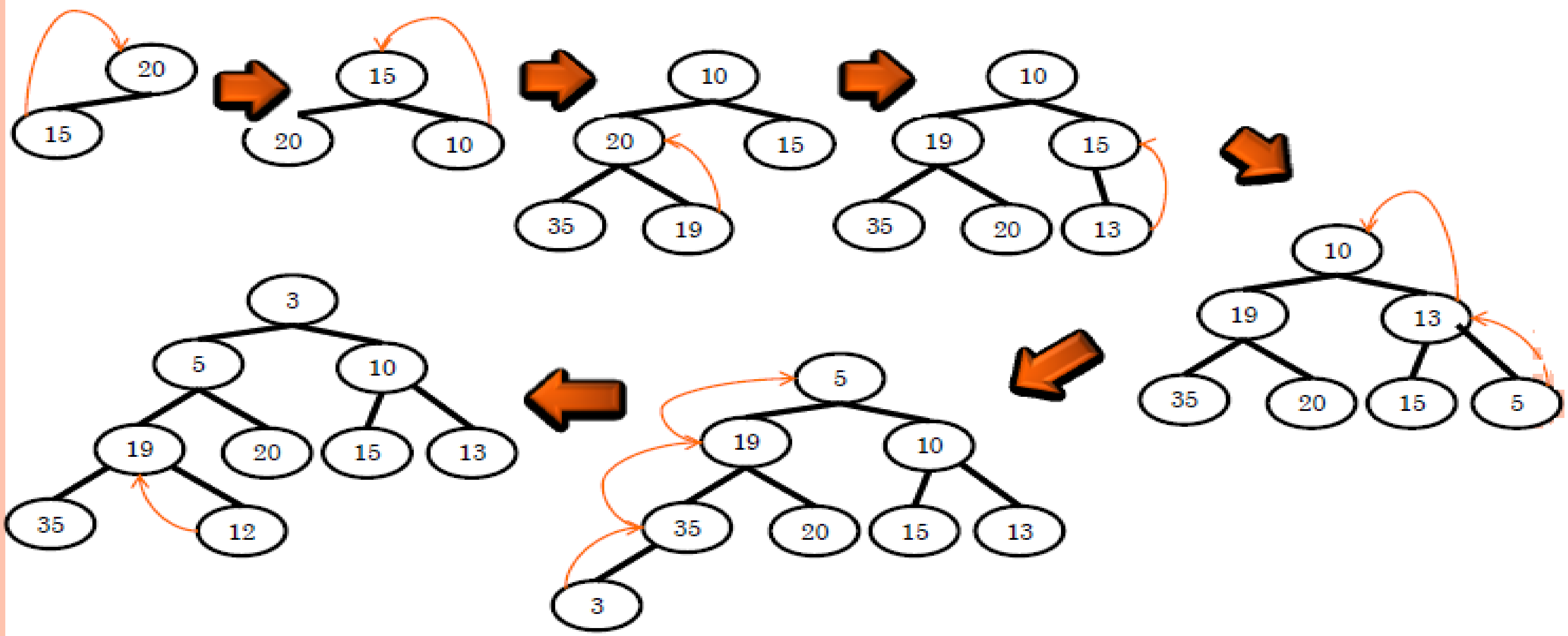
```
ExtraireMin (Tas: tableau, n : entier )
Début
Tas [1] ← T[n];
min ← 1; Sortie ←faux
TQ (non sortie) faire
   i←min; g ← 2* i ; d ← 2 *i + 1
   Si g < n et Tas[g] < Tas[min] alors min ← g
   Si d < n et Tas[d] < Tas[min] alors min ← d
   Si min <> i alors Permuter (Tas, i, min)
   Sinon Sortie ←vrai
Fin
```

O(TextraireMin(n)) = O(log2 n)

Tri par TAS_{Min}

1. Transformer le tableau en un TAS_{Min}

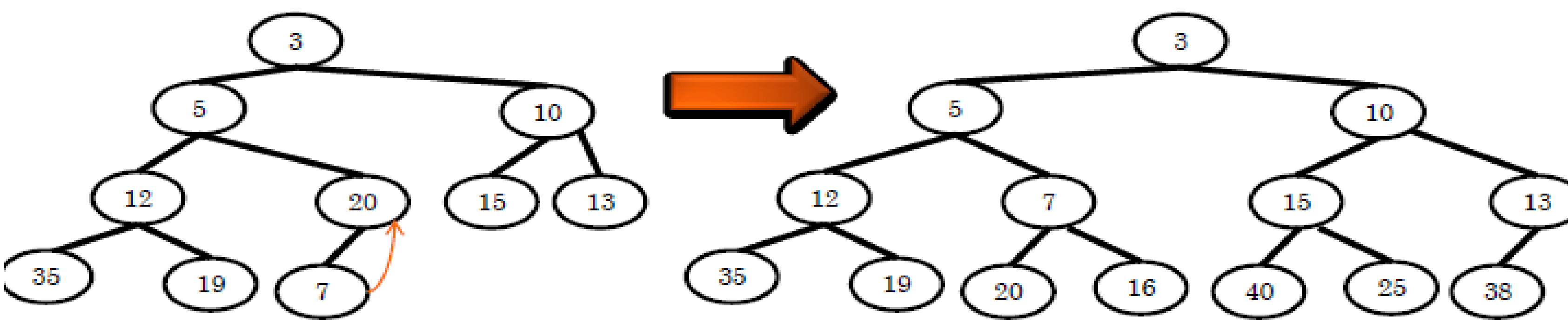
20	15	10	35	19	13	5	3	12	7	16	40	25	38



Tri par TAS_{Min}

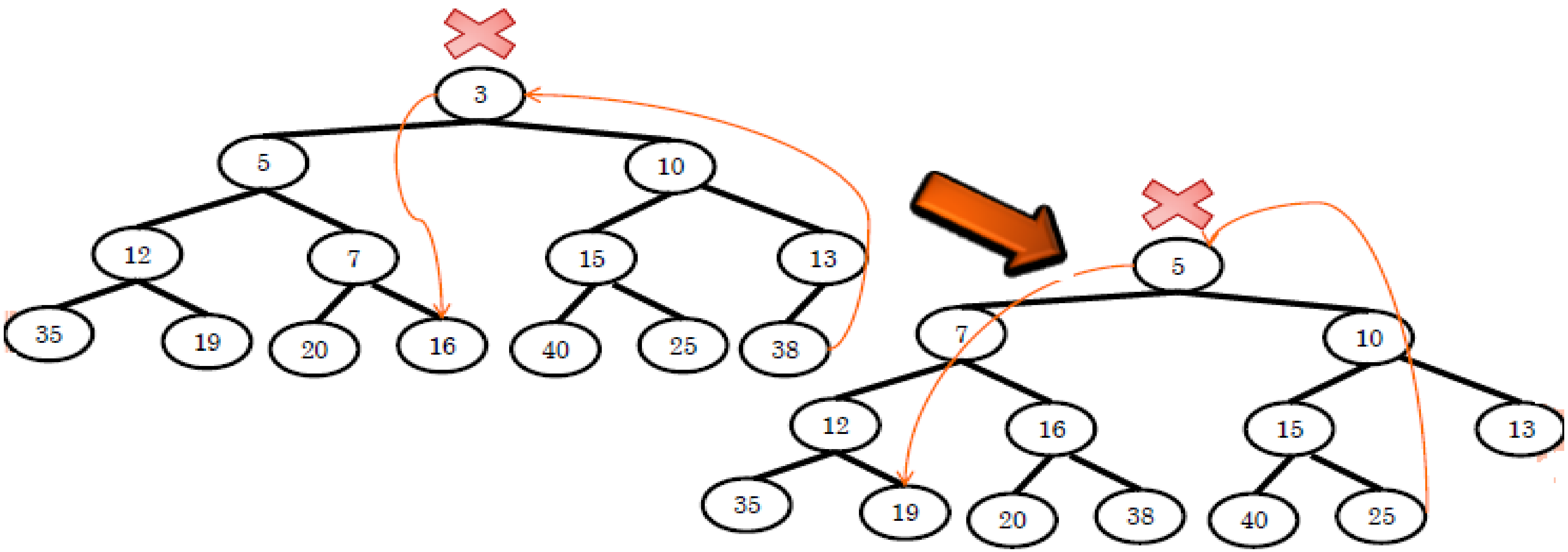
1. Transformer le tableau en un TAS_{Min}

20	15	10	35	19	13	5	3	12	7	16	40	25	38



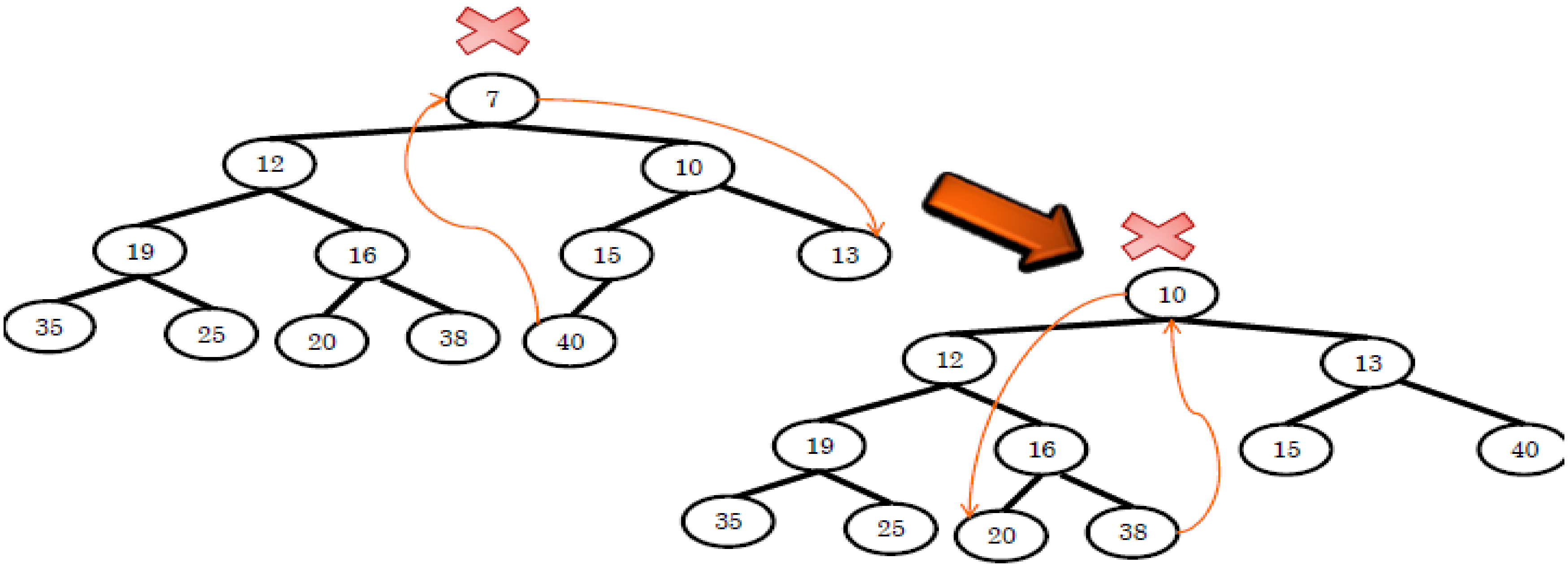
Tri par TAS_{Min}

2. Extraire n fois le minimum du tas:



3	5

Tri par TAS$_{Min}$



3	5	7	10

Tri par TAS_{Min}

```
Tri_TASmin (T: Tableau, n: entier)
Début
//Construire le TAS
Pour i← 1 à n faire
    Insérer_TAS (Tas, i-1, T[i])
    // Extraire les minimums
Pour i←1 à n faire
     T[i] ← TAS[1]
     Extraire_Minimum (TAS, n)
Finpour
Fin
```

$O(Ttri\_TAS(n)) = O(n \log_2 n)$

Conclusion

Algorithme de Tri	Complexité au Pire
Tri par Sélection	$O(n^2)$
Tri par Insertion	$O(n^2)$
Tri par Propagation	$O(n^2)$
Tri par ABR	$O(n^2)$
Tri Rapide	$O(n^2)$ $O(n\log_2(n))$
Tri par Fusion	$O(n\log_2(n))$
Tri par TAS	$O(n\log_2(n))$