Arbres et tas

Algorithmique et structures de données, 2023-2024

P. Kunzli (Cloud) et O. Malaspinas (A401), ISC, HEPIA

2024-03-19

En partie inspirés des supports de cours de P. Albuquerque

Rappel: L'insertion

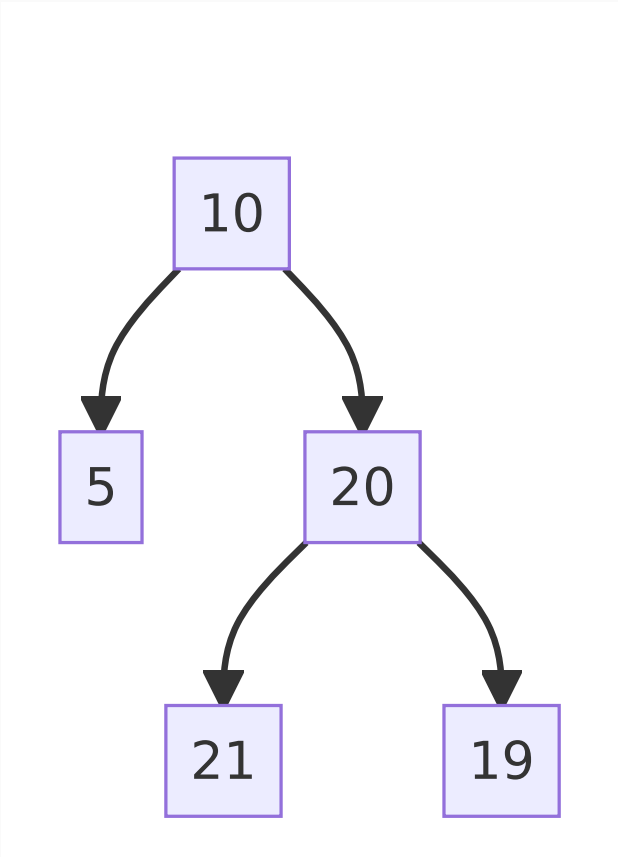
```
rien ajout(arbre, clé)
    si est_vide(arbre)
        arbre = nœud(clé)
    sinon
        arbre = position(arbre, clé)
        si clé < clé(arbre)
            gauche(arbre) = nœud(clé)
        sinon si clé > clé(arbre)
            droite(arbre) = nœud(clé)
        sinon
            retourne
```

La suppression de clé

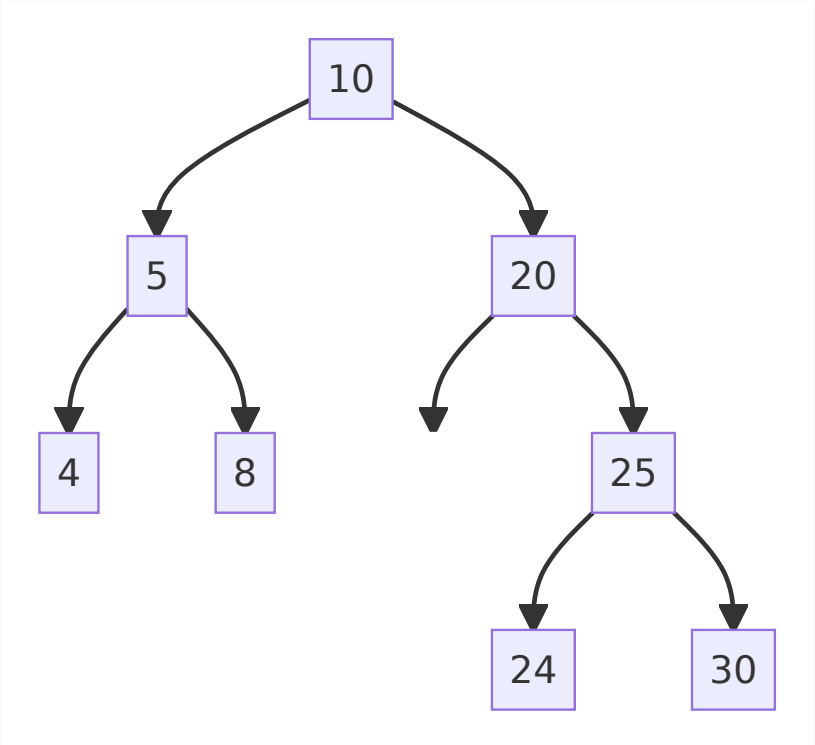
Cas simples:

- le nœud est absent,
- le nœud est une feuille
- le nœuds a un seul fils.

Une feuille (le 19 p.ex.).



Un seul fils (le 20 p.ex.).



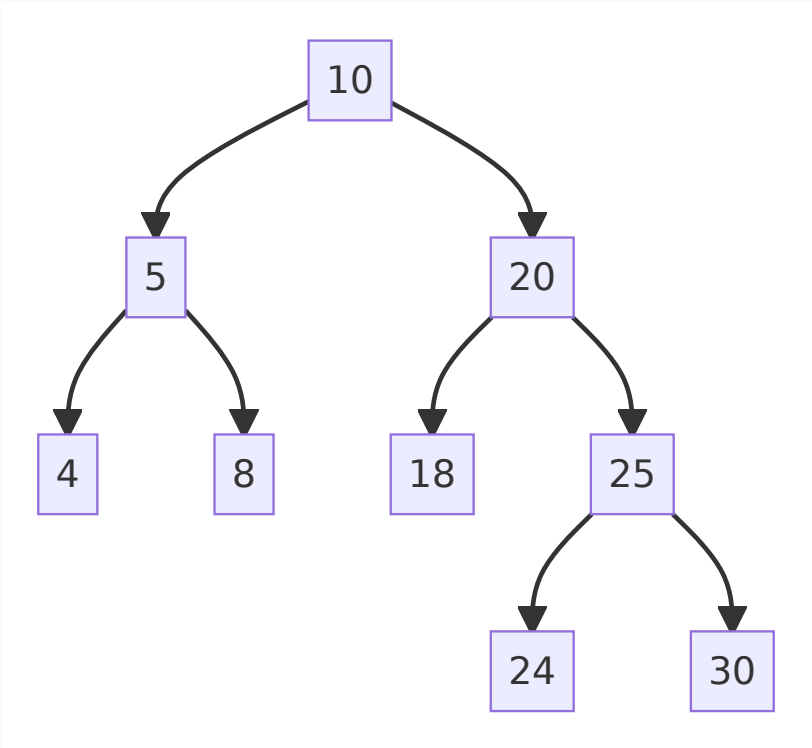
Dans tous les cas

- Chercher le nœud à supprimer: utiliser `position()`.

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).

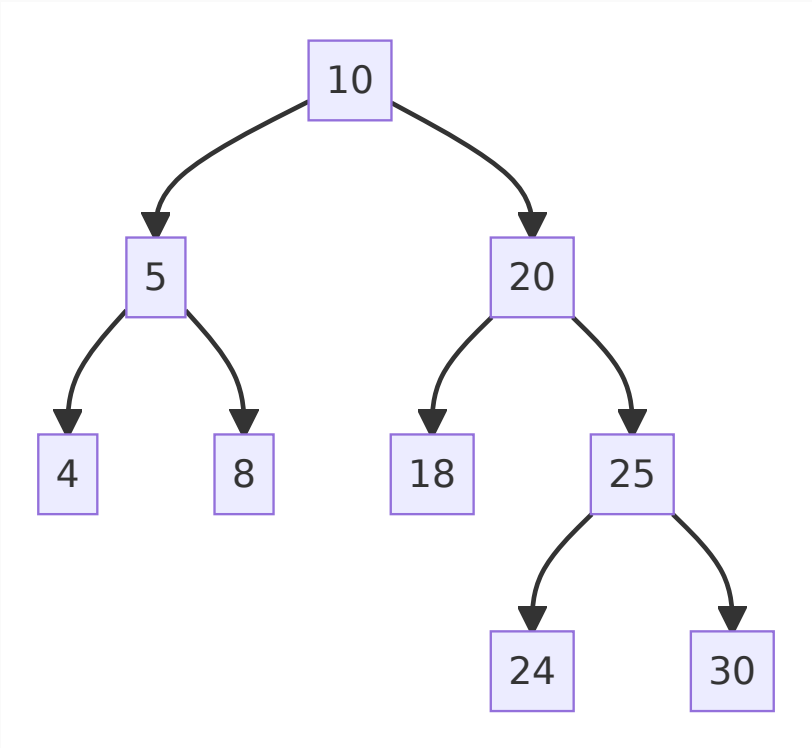


- Si on enlève 10 il se passe quoi?

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).

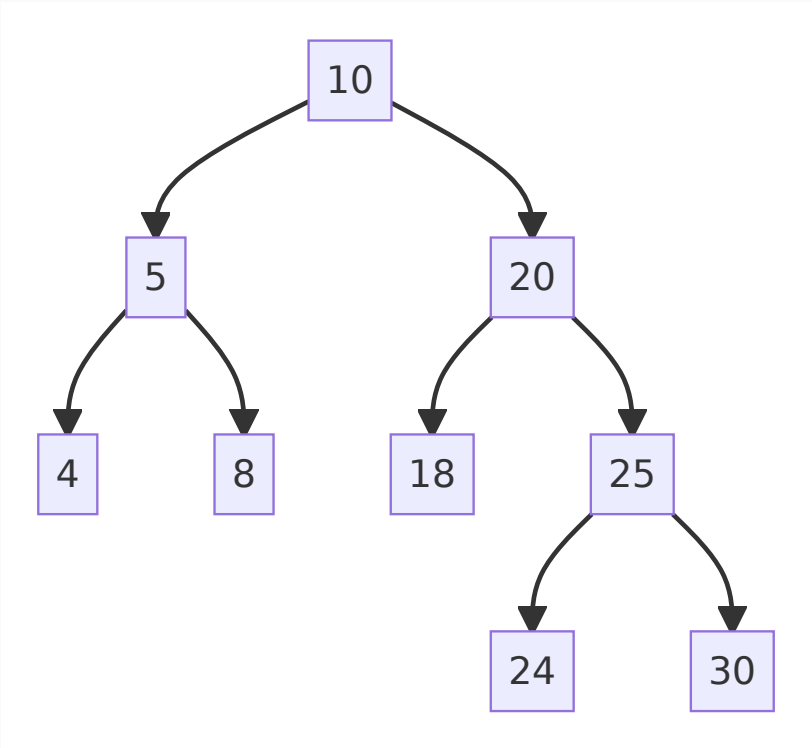


- Si on enlève 10 il se passe quoi?
- On peut pas juste enlever 10 et recoller...
- Proposez une solution bon sang!

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).



- Si on enlève 10 il se passe quoi?
- On peut pas juste enlever 10 et recoller...
- Proposez une solution bon sang!

Solution

- Échange de la valeur à droite dans le sous-arbre de gauche ou ...
- de la valeur de gauche dans le sous-arbre de droite!
- Puis, on retire le nœud.

Le pseudo-code de la suppression

Pour une feuille ou absent (ensemble)

```
arbre suppression(arbre, clé)
    sous_arbre = position(arbre, clé)
    si est_vide(sous_arbre) ou clé(sous_arbre) != clé
        retourne vide
    sinon
        si est_feuille(sous_arbre) et clé(sous_arbre) == clé
            nouvelle_feuille = parent(arbre, sous_arbre)
            si est_vide(nouvelle_feuille)
                arbre = vide
            sinon
                si gauche(nouvelle_feuille) == sous_arbre
                    gauche(nouvelle_feuille) = vide
                sinon
                    droite(nouvelle_feuille) = vide
        retourne sous_arbre
```

Il nous manque le code pour le `parent`

Pseudo-code pour trouver le parent (5min -> matrix)

Il nous manque le code pour le `parent`

Pseudo-code pour trouver le parent (5min -> matrix)

```
arbre parent(arbre, sous_arbre)
    si est_non_vide(arbre)
        actuel = arbre
        parent = actuel
        clé = clé(sous_arbre)
        faire
            si (clé != clé(actuel))
                parent = actuel
                si clé < clé(actuel)
                    actuel = gauche(actuel)
                sinon
                    actuel = droite(actuel)
            sinon
                retour parent
        tant_que (actuel != sous_arbre)
    retourne vide
```

Le pseudo-code de la suppression

Pour un seul enfant (5min -> matrix)

Le pseudo-code de la suppression

Pour un seul enfant (5min -> matrix)

```
arbre suppression(arbre, clé)
    sous_arbre = position(arbre, clé)
    si est_vide(gauche(sous_arbre)) ou est_vide(droite(sous_arbre))
        parent = parent(arbre, sous_arbre)
        si est_vide(gauche(sous_arbre))
            si droite(parent) == sous_arbre
                droite(parent) = droite(sous_arbre)
            sinon
                gauche(parent) = droite(sous_arbre)
        sinon
            si droite(parent) == sous_arbreou est_
                droite(parent) = gauche(sous_arbre)
            sinon
                gauche(parent) = gauche(sous_arbre)
        retourne sous_arbre
```

Le pseudo-code de la suppression

Pour au moins deux enfants (ensemble)

```
arbre suppression(arbre, clé)
    sous_arbre = position(arbre, clé) # on revérifie pas que c'est bien la clé
    si est_non_vide(gauche(sous_arbre)) et est_non_vide(droite(sous_arbre))
        max_gauche = position(gauche(sous_arbre), clé)
        échange(clé(max_gauche), clé(sous_arbre))
        suppression(gauche(sous_arbre), clé)
```

Exercices (poster sur matrix)

1. Écrire le pseudo-code de l'insertion purement en récursif.

Exercices (poster sur matrix)

1. Écrire le pseudo-code de l'insertion purement en récursif.

```
arbre insertion(arbre, clé)
    si est_vide(arbre)
        retourne nœud(clé)

    si (clé < arbre->clé)
        gauche(arbre) = insert(gauche(arbre), clé)
    sinon
        droite(arbre) = insert(droite(arbre), clé)
    retourne arbre
```

Exercices (poster sur matrix)

2. Écrire le pseudo-code de la recherche purement en récursif.

Exercices (poster sur matrix)

2. Écrire le pseudo-code de la recherche purement en récursif.

```
bool recherche(arbre, clé)
    si est_vide(arbre)
        retourne faux // pas trouvée
    si clé(arbre) == clé
        retourne vrai // trouvée
    si clé < clé(arbre)
        retourne recherche(gauche(arbre), clé)
    sinon
        retourne recherche(droite(arbre), clé)
```

Exercices (à la maison)

3. Écrire une fonction qui insère des mots dans un arbre et ensuite affiche l'arbre.

Trier un tableau à l'aide d'un arbre binaire

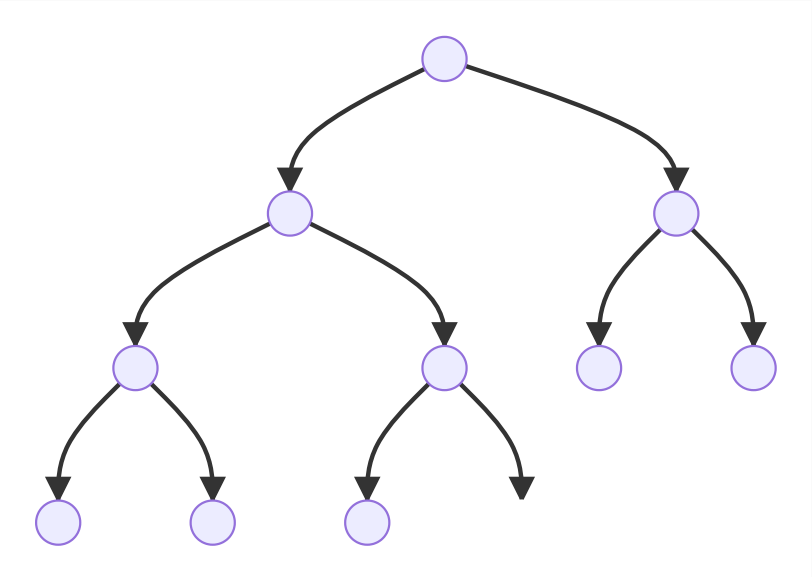
- Tableau représenté comme un arbre binaire.
- Aide à comprendre "comment" trier, mais on ne construit jamais l'arbre.
- Complexité $O(N \log_2 N)$ en moyenne et grande stabilité (pas de cas dégénérés).

Lien entre arbre et tableau

- La racine de l'arbre set le premier élément du tableau.
- Les deux fils d'un nœud d'indice i, ont pour indices $2i + 1$ et $2i + 2$:
  - Les fils du nœud $i = 0$, sont à $2 \cdot 0 + 1 = 1$ et $2 \cdot 0 + 2 = 2$.
  - Les fils du nœud $i = 1$, sont à $2 \cdot 1 + 1 = 3$ et $2 \cdot 1 + 2 = 4$.
  - Les fils du nœud $i = 2$, sont à $2 \cdot 2 + 2 = 5$ et $2 \cdot 1 + 2 = 6$.
  - Les fils du nœud $i = 3$, sont à $2 \cdot 3 + 1 = 7$ et $2 \cdot 3 + 2 = 8$.
- Un élément d'indice i a pour parent l'élément $(i - 1)/2$ (division entière):
  - Le parent du nœud $i = 8$ est $(8 - 1)/2 = 3$.
  - Le parent du nœud $i = 7$ est $(7 - 1)/2 = 3$.

Visuellement

- Où vont les indices correspondant du tableau?



- Les flèche de gauche à droite, parent -> enfants.
- Les flèche de droite à gauche, enfants -> parent.

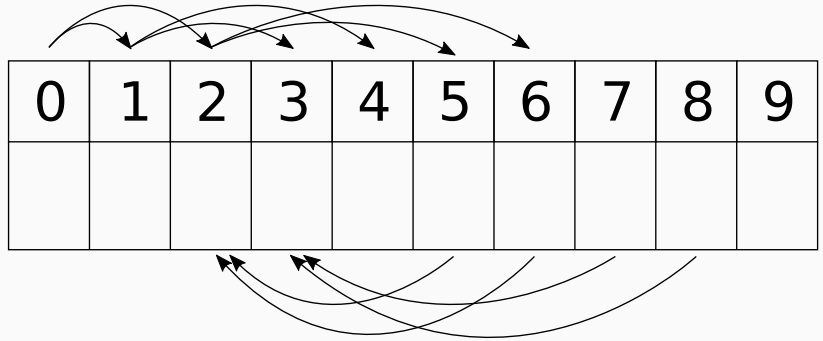


Figure 1: Dualité tableau arbre binaire.

Propriétés:

1. les feuilles sont toutes sur l'avant dernier ou dernier niveau.
2. les feuilles de profondeur maximale sont "tassée" à gauche.

Le tas (ou heap)

Définition

- Un arbre est un tas, si la valeur de chacun de ses descendants est inférieure ou égale à sa propre valeur.

Exemples (ou pas)

```
16  8 14 6 2 10 12 4 5 # Tas
16 14  8 6 2 10 12 4 5 # Non-tas, 10 > 8 et 12 > 8
```

Exercices (ou pas)

```
19 18 12 12 17 1 13 4 5 # Tas ou pas tas?
19 18 16 12 17 1 12 4 5 # Tas ou pas tas?
```

Le tas (ou heap)

Définition

- Un arbre est un tas, si la valeur de chacun de ses descendants est inférieure ou égale à sa propre valeur.

Exemples (ou pas)

```
16  8 14 6 2 10 12 4 5 # Tas
16 14  8 6 2 10 12 4 5 # Non-tas, 10 > 8 et 12 > 8
```

Exercices (ou pas)

```
19 18 12 12 17 1 13 4 5 # Tas ou pas tas?
19 18 16 12 17 1 12 4 5 # Tas ou pas tas?

19 18 12 12 17 1 13 4 5 # Pas tas! 13 > 12
19 18 16 12 17 1 12 4 5 # Tas!
```

Exemple de tri par tas (1/13)

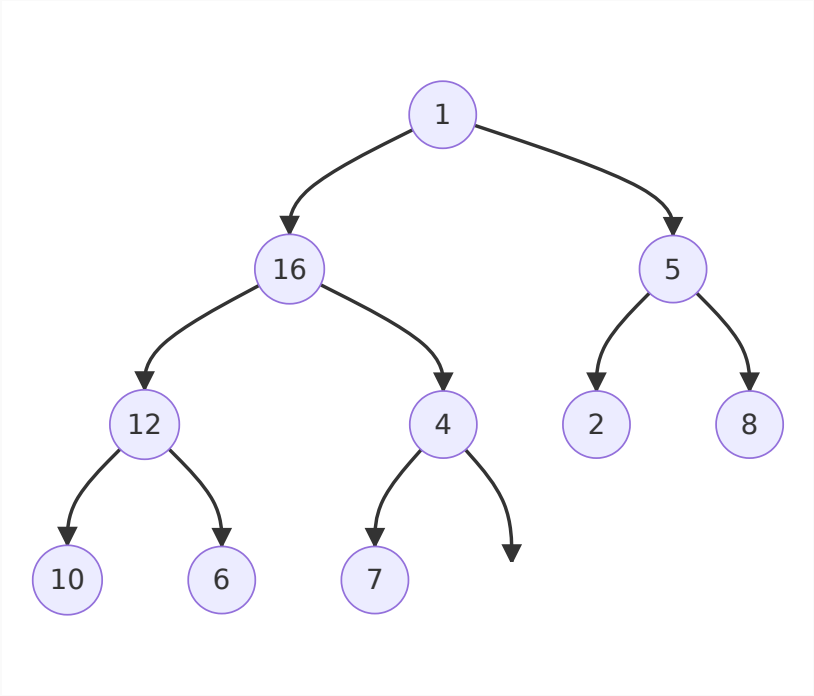
| 1 | 16 | 5 | 12 | 4 | 2 | 8 | 10 | 6 | 7 |

- Quel est l'arbre que cela représente?

Exemple de tri par tas (1/13)

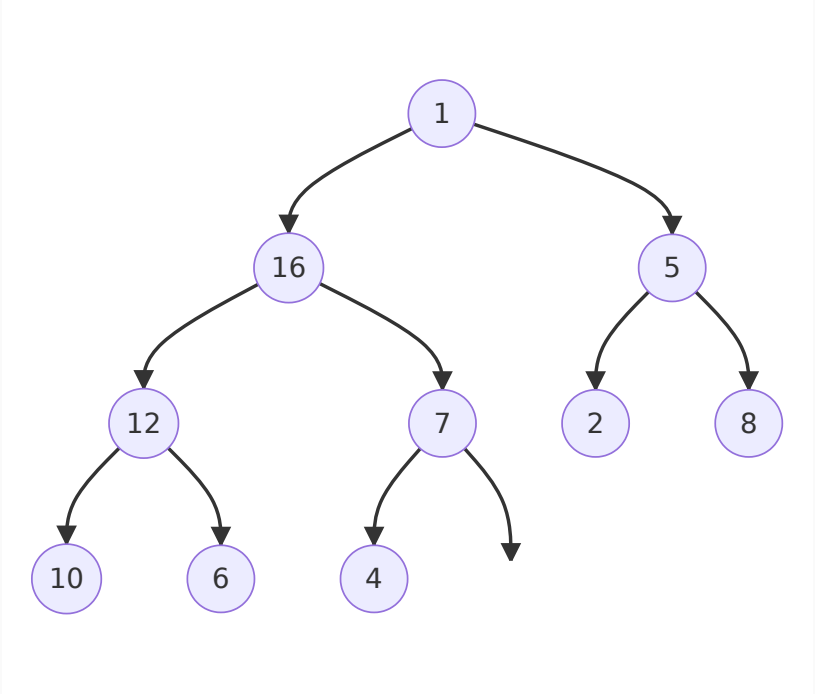
| 1 | 16 | 5 | 12 | 4 | 2 | 8 | 10 | 6 | 7 |

- Quel est l'arbre que cela représente?



But: Transformer l'arbre en tas.

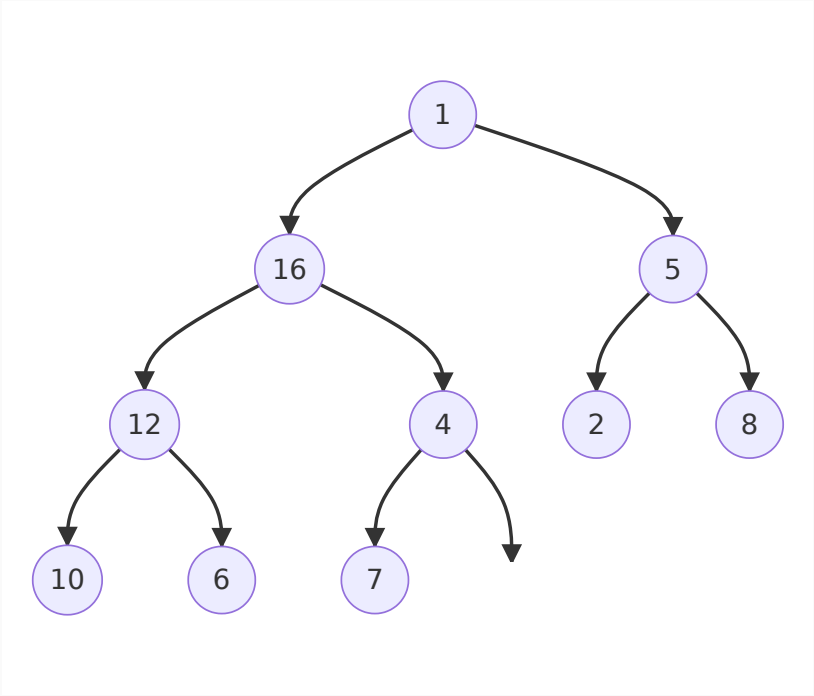
- On commence à l'indice $N/2 = 5$: 4.
- 7 > 4 (enfant > parent).
- intervertir 4 et 7.



Exemple de tri par tas (1/13)

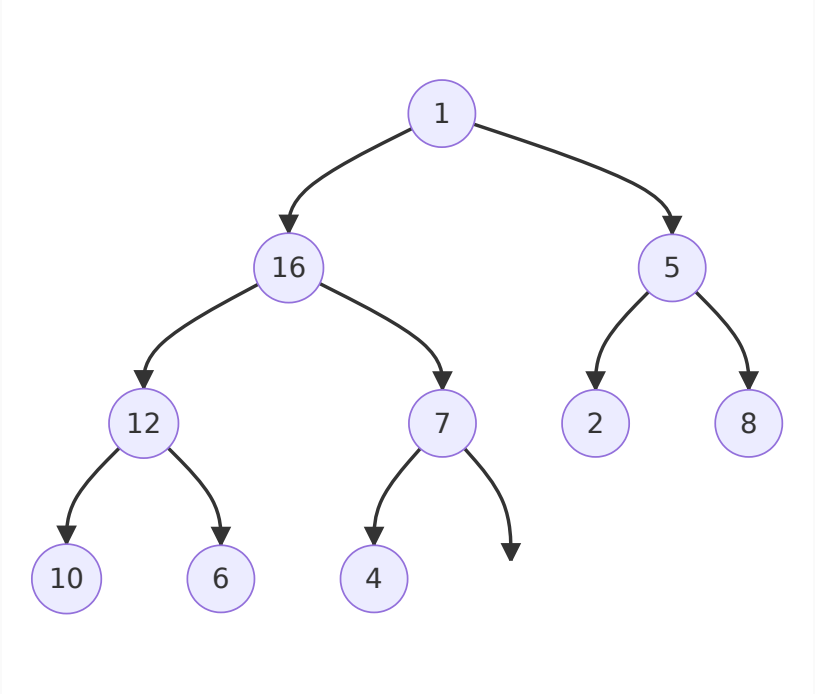
| 1 | 16 | 5 | 12 | 4 | 2 | 8 | 10 | 6 | 7 |

- Quel est l'arbre que cela représente?



But: Transformer l'arbre en tas.

- On commence à l'indice $N/2 = 5$: 4.
- 7 > 4 (enfant > parent).
- intervertir 4 et 7.



* *

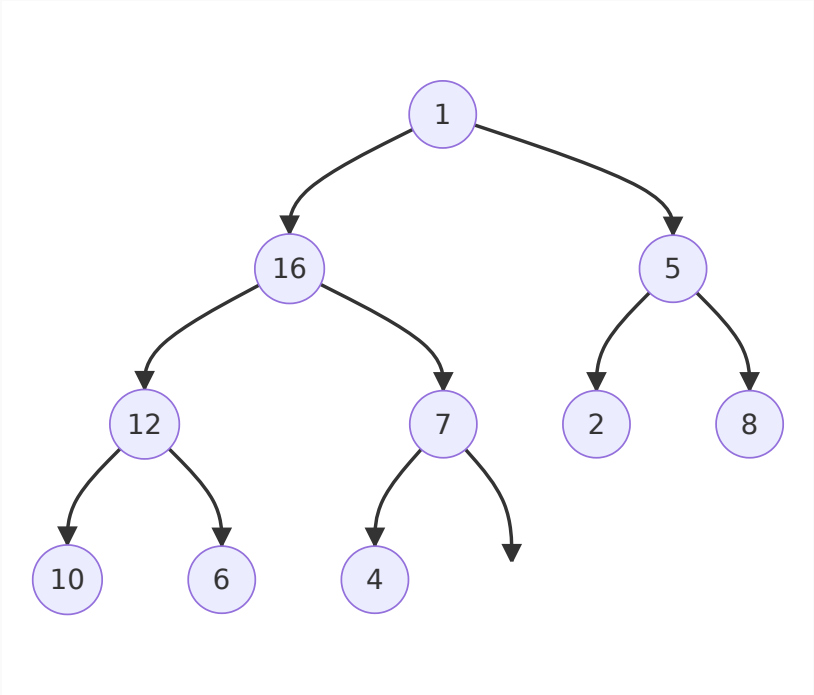
| 1 | 16 | 5 | 12 | 7 | 2 | 8 | 10 | 6 | 4 |

Exemple de tri par tas (2/13)

| 1 | 16 | 5 | 12 | 7 | 2 | 8 | 10 | 6 | 4 |

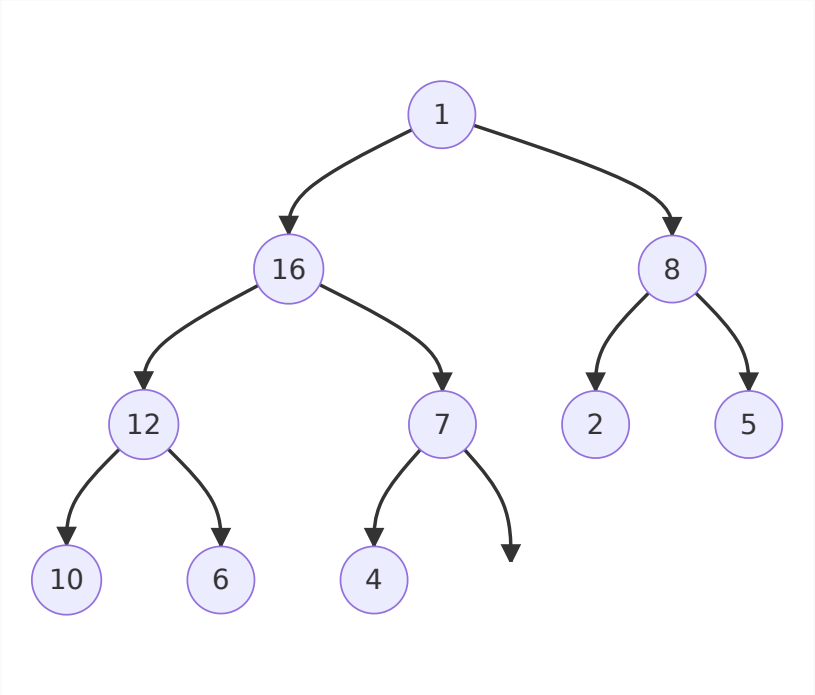
But: Transformer l'arbre en tas.

- On continue à l'indice $N/2 - 1 = 4$: 12.
- Déjà un tas, rien à faire.



But: Transformer l'arbre en tas.

- On continue à l'indice $N/2 - 2 = 3$: 5.
- 5 < 8, échanger 8 et 5 (aka `max(2, 5, 8)`)

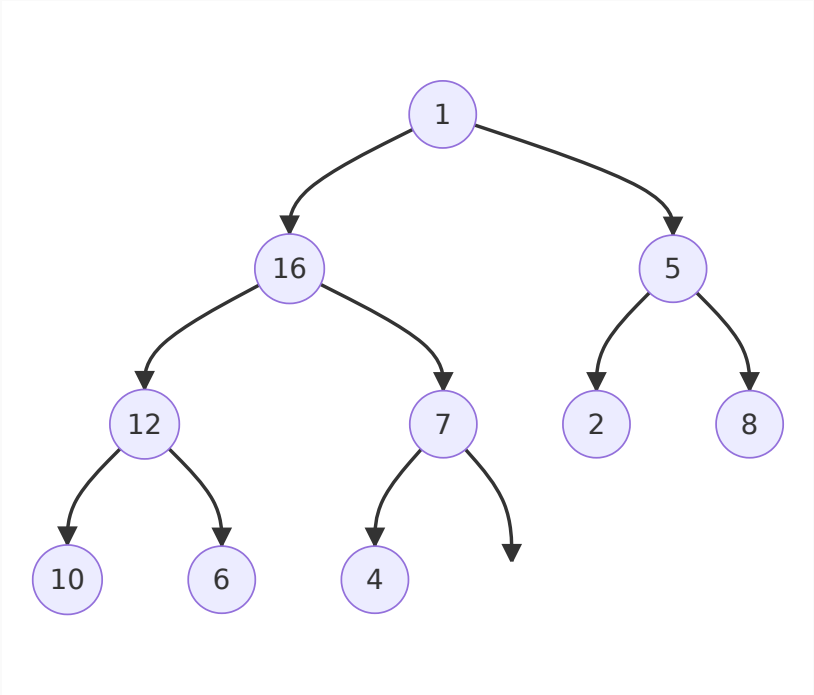


Exemple de tri par tas (2/13)

| 1 | 16 | 5 | 12 | 7 | 2 | 8 | 10 | 6 | 4 |

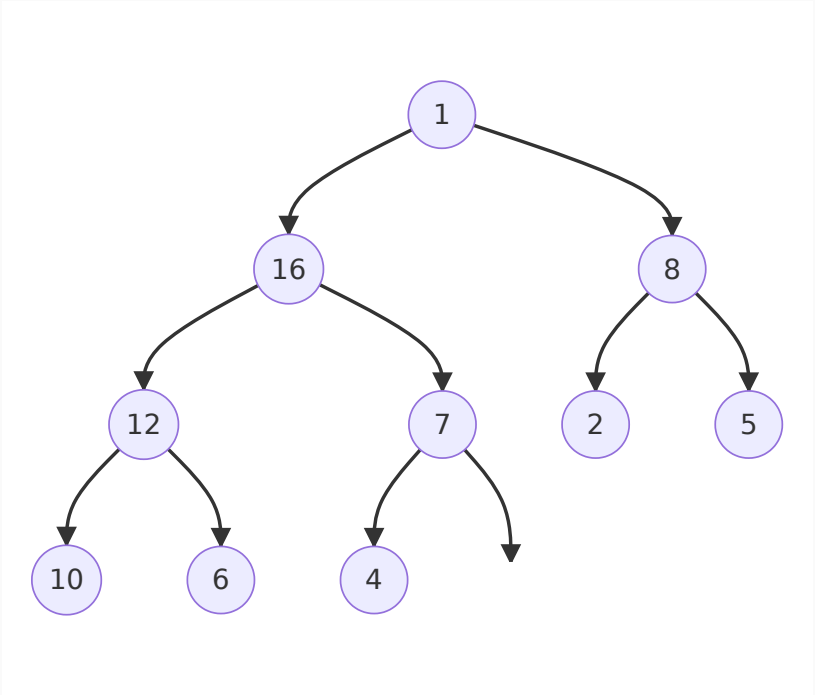
But: Transformer l'arbre en tas.

- On continue à l'indice $N/2 - 1 = 4$: 12.
- Déjà un tas, rien à faire.



But: Transformer l'arbre en tas.

- On continue à l'indice $N/2 - 2 = 3$: 5.
- 5 < 8, échanger 8 et 5 (aka `max(2, 5, 8)`)



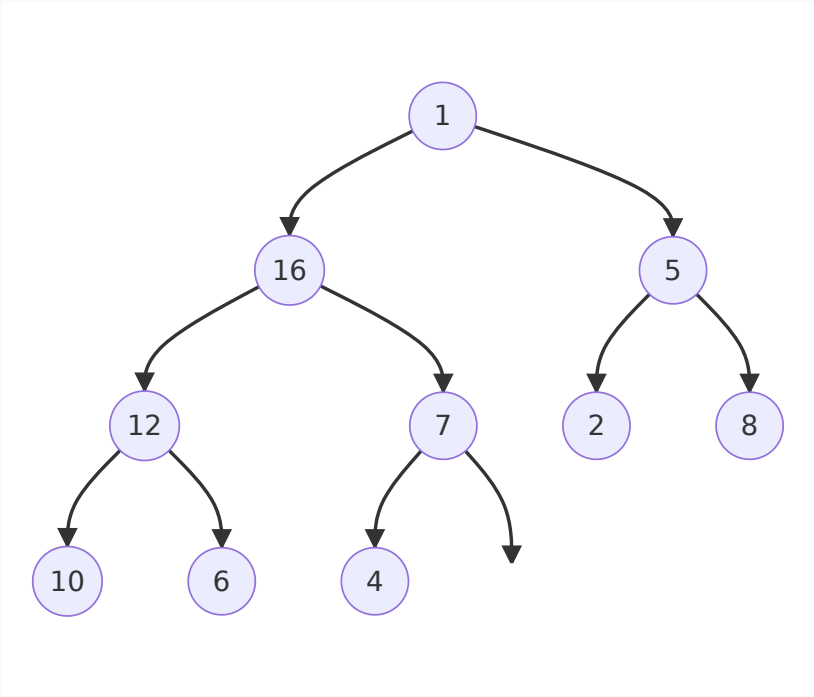
| 1 | 16 | 8 | 12 | 7 | 2 | 5 | 10 | 6 | 4 |

Exemple de tri par tas (3/13)

| 1 | 16 | 5 | 12 | 7 | 2 | 8 | 10 | 6 | 4 |

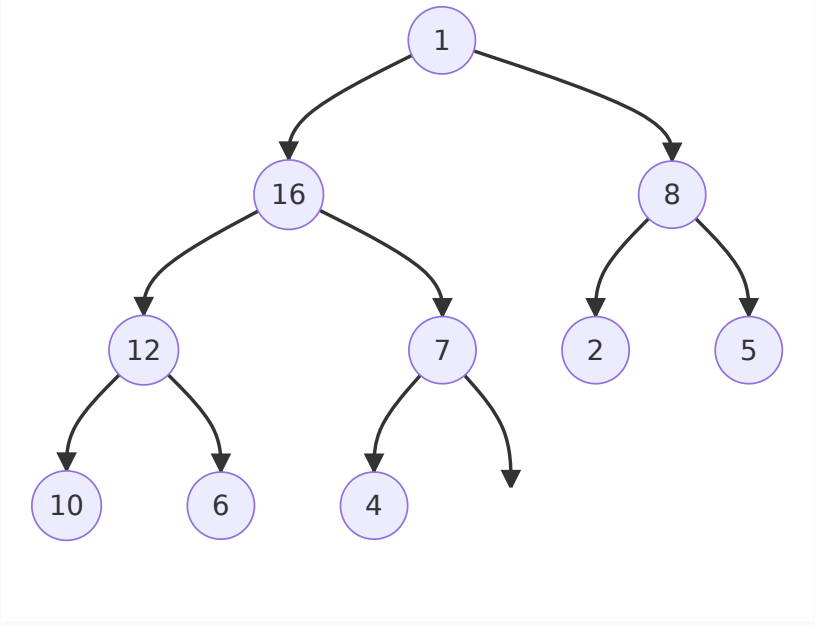
But: Transformer l'arbre en tas.

- Indice $N/2 - 1 = 4$: 12.
- Déjà un tas, rien à faire.



But: Transformer l'arbre en tas.

- Indice $N/2 - 2 = 3$: 5.
- `5 < 8`, `5 <=> max(2, 5, 8)`



*

| 1 | 16 | 8 | 12 | 7 | 2 | 5 | 10 | 6 | 4 |

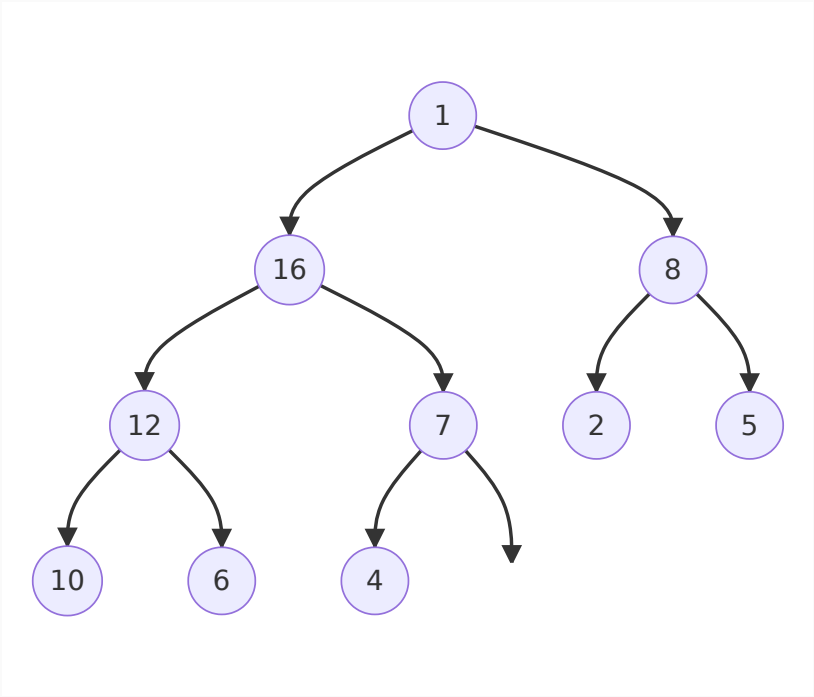
*

Exemple de tri par tas (4/13)

| 1 | 16 | 8 | 12 | 7 | 2 | 5 | 10 | 6 | 4 |

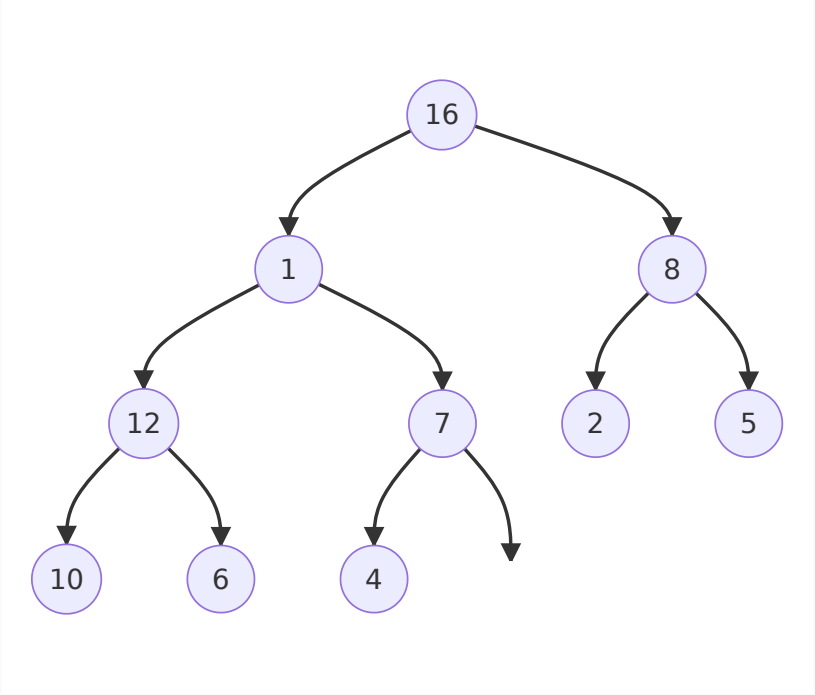
But: Transformer l'arbre en tas.

- Indice $N/2 - 3 = 1$: 16.
- Déjà un tas, rien à faire.



But: Transformer l'arbre en tas.

- Indice $N/2 - 4 = 1$: 1.
- `1 < 16 && 1 < 8`, `1 <=> max(1, 16, 8)`



* *

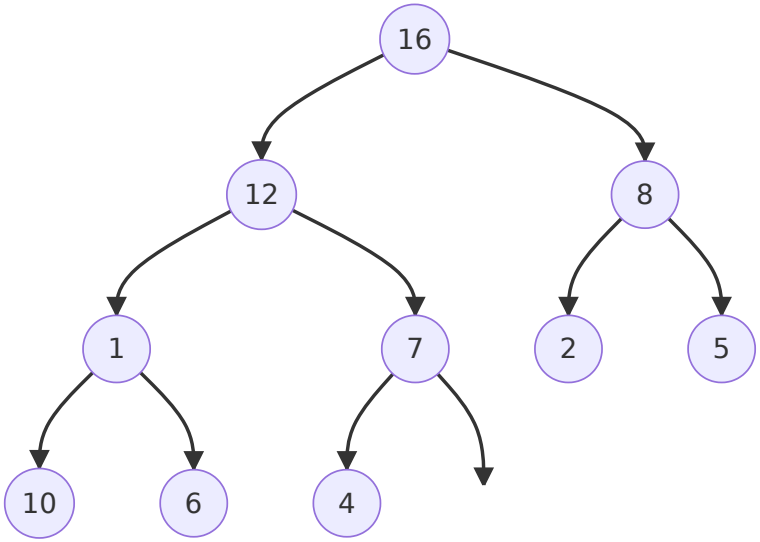
| 16 | 1 | 8 | 12 | 7 | 2 | 5 | 10 | 6 | 4 |

Exemple de tri par tas (5/13)

| 16 | 1 | 8 | 12 | 7 | 2 | 5 | 10 | 6 | 4 |

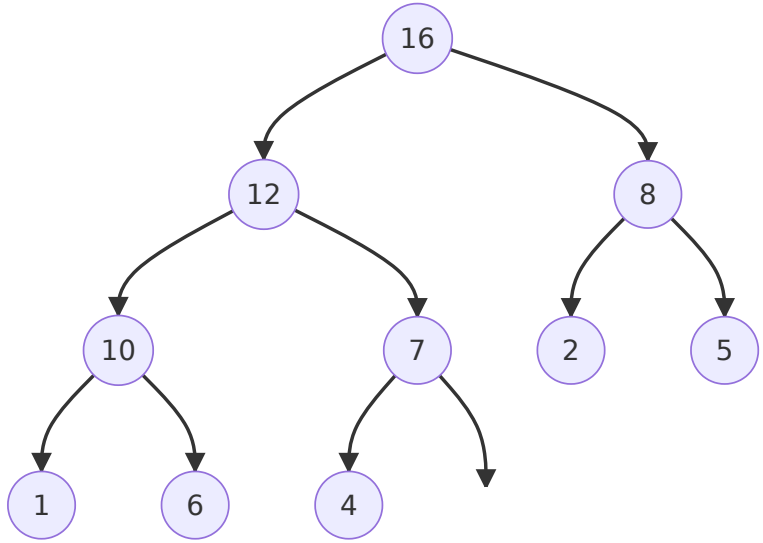
But: Transformer l'arbre en tas.

- Recommencer avec 1.
- `1 <=> max(1, 12, 7)`.



But: Transformer l'arbre en tas.

- Recommencer avec 1.
- `1 <=> max(1, 10, 6)`.



* * *

| 16 | 12 | 8 | 10 | 7 | 2 | 5 | 1 | 6 | 4 |

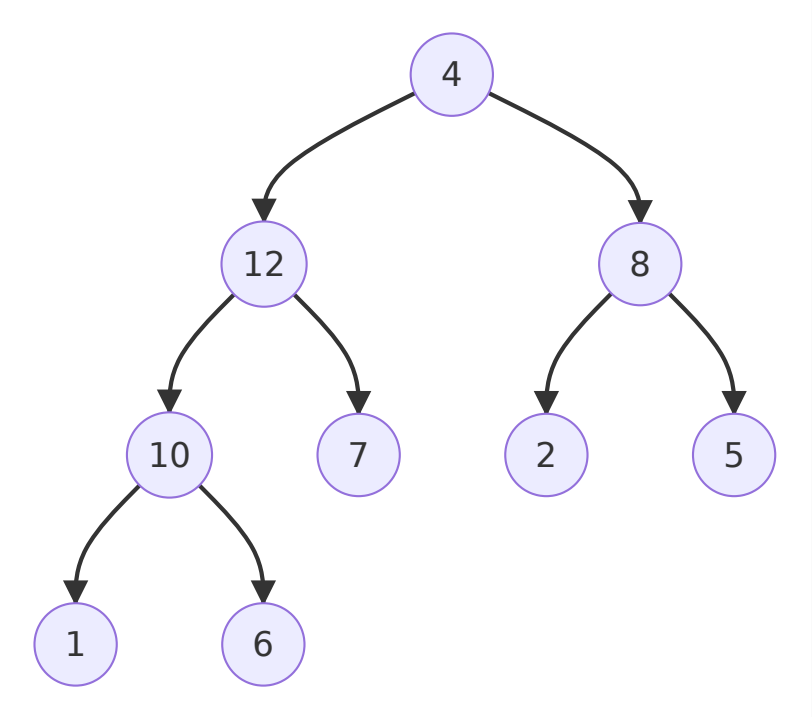
- L'arbre est un tas.

Exemple de tri par tas (6/13)

| 16 | 12 | 8 | 10 | 7 | 2 | 5 | 1 | 6 | 4 |

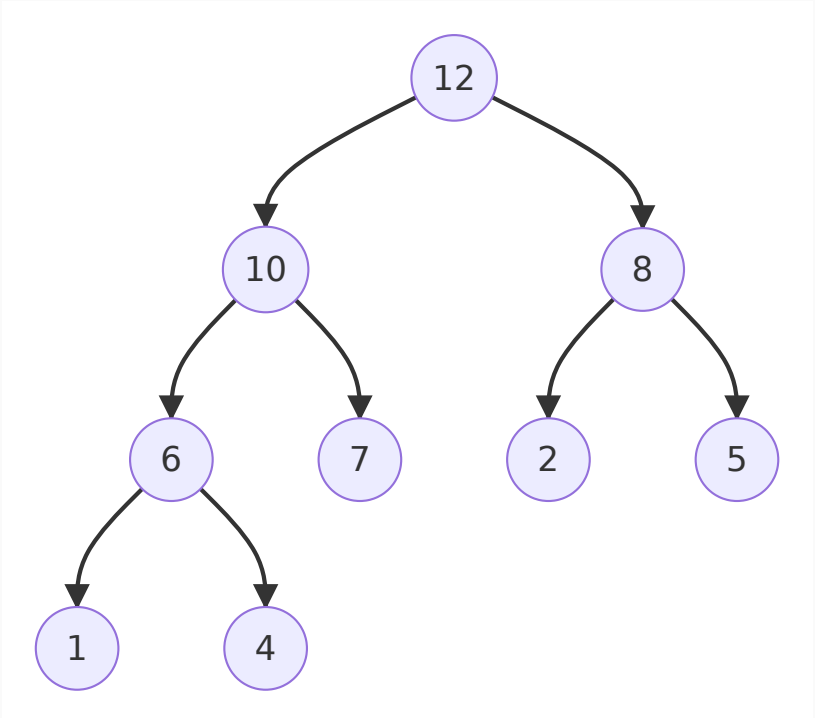
But: Trier les tas.

- Échanger la racine, 16 (`max` de l'arbre) avec 4.
- Traiter la racine.



But: Trier les tas.

- `4 <=> max(4, 12, 8).`
- `4 <=> max(4, 10, 7).`
- `4 <=> max(4, 1, 6).`



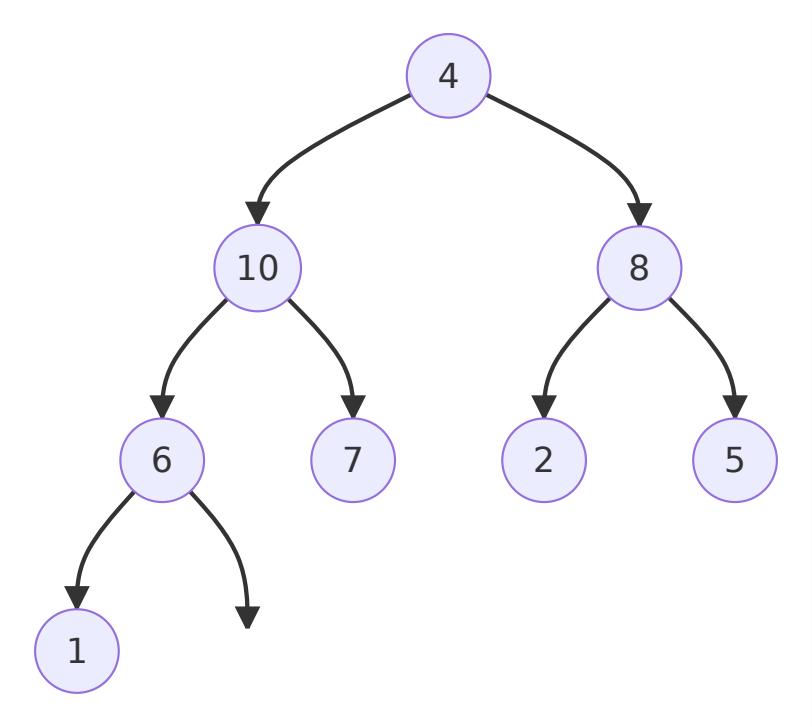
| 12 | 10 | 8 | 6 | 7 | 2 | 5 | 1 | 4 || 16

Exemple de tri par tas (7/13)

| 12 | 10 | 8 | 6 | 7 | 2 | 5 | 1 | 4 || 16

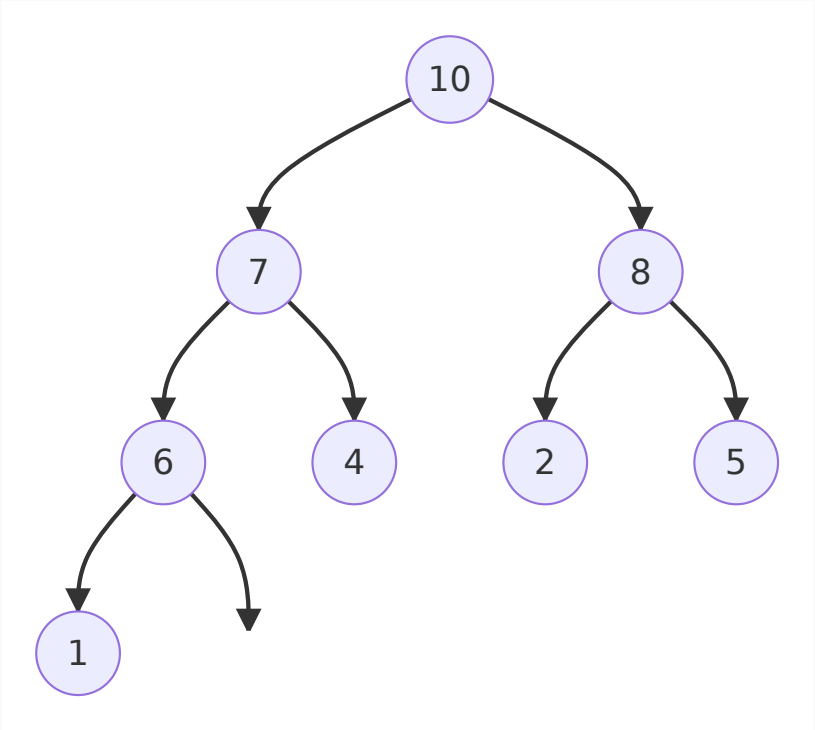
But: Trier les tas.

- Échanger la racine, 12 (`max` de l'arbre) avec 4.
- Traiter la racine.



But: Trier les tas.

- `4 <=> max(4, 10, 8).`
- `4 <=> max(4, 6, 7).`



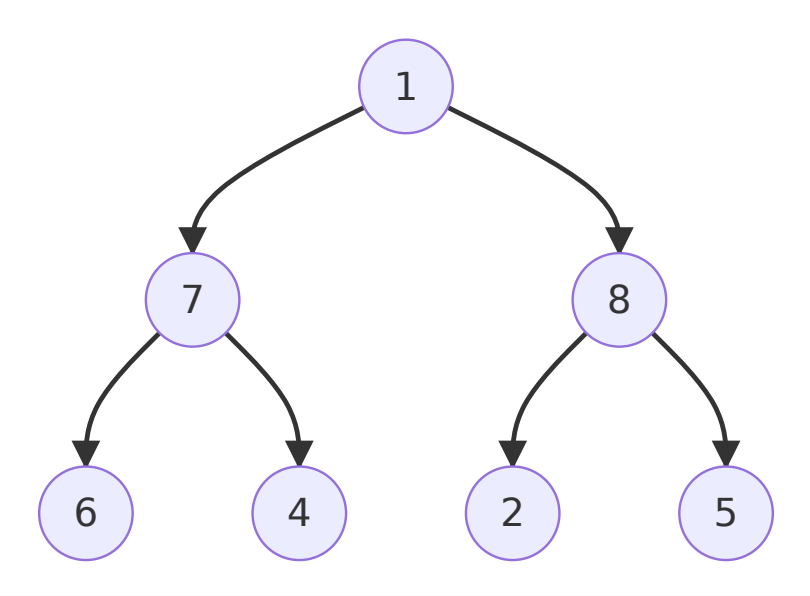
| 10 | 7 | 8 | 6 | 4 | 2 | 5 | 1 || 12 | 16

Exemple de tri par tas (8/13)

| 10 | 7 | 8 | 6 | 4 | 2 | 5 | 1 || 12 | 16

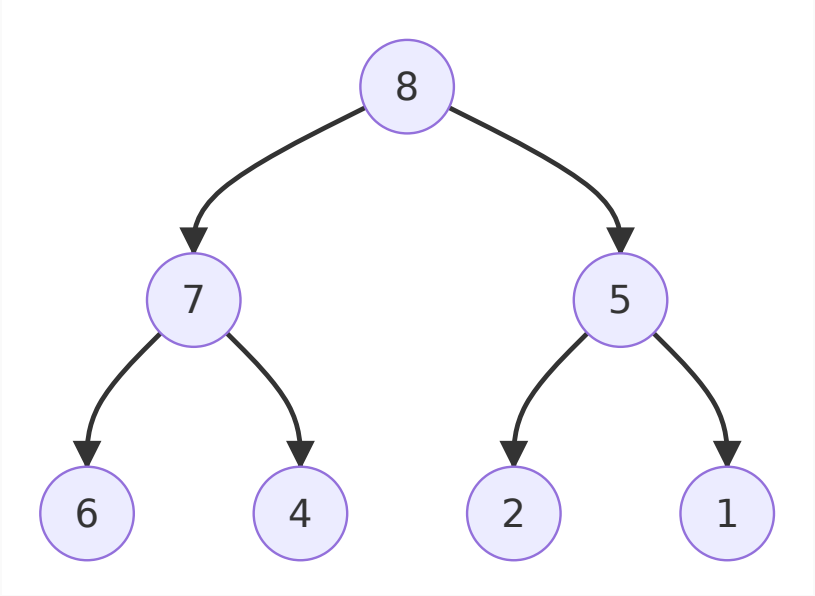
But: Trier les tas.

- Échanger la racine, 10 (`max` de l'arbre) avec 1.
- Traiter la racine.



But: Trier les tas.

- `1 <=> max(1, 7, 8).`
- `5 <=> max(1, 2, 5).`



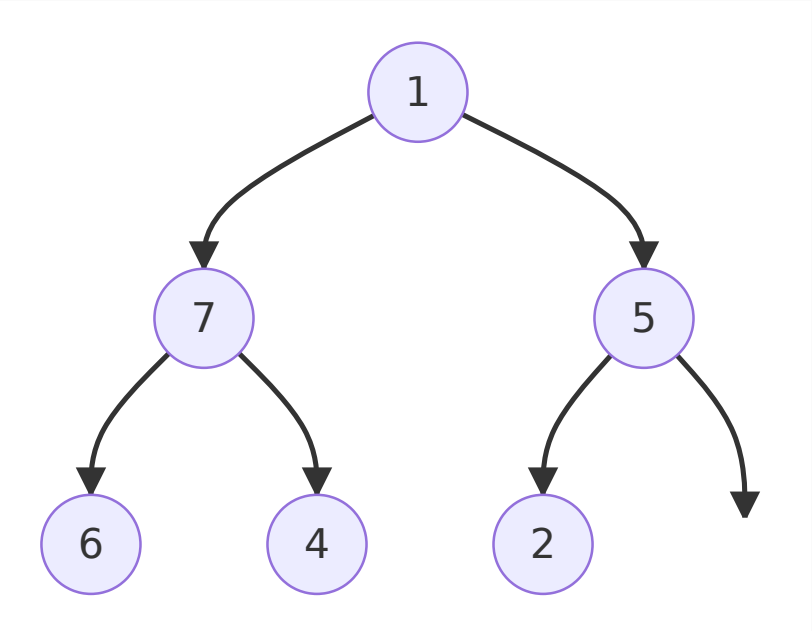
| 8 | 7 | 5 | 6 | 4 | 2 | 1 || 10 | 12 | 16

Exemple de tri par tas (9/13)

| 8 | 7 | 5 | 6 | 4 | 2 | 1 || 10 | 12 | 16

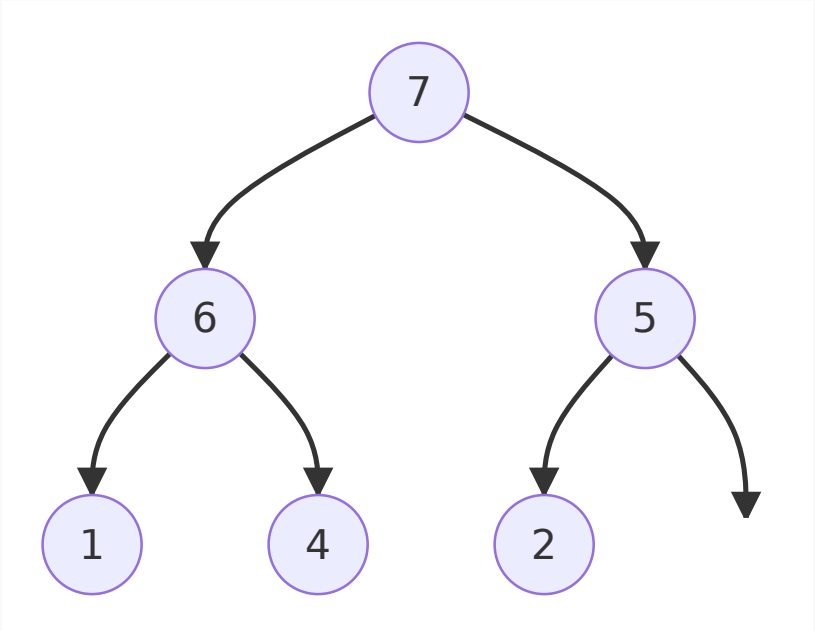
But: Trier les tas.

- Échanger la racine, 8 (`max` de l'arbre) avec 1.
- Traiter la racine.



But: Trier les tas.

- `1 <=> max(1, 7, 5).`
- `1 <=> max(1, 6, 4).`



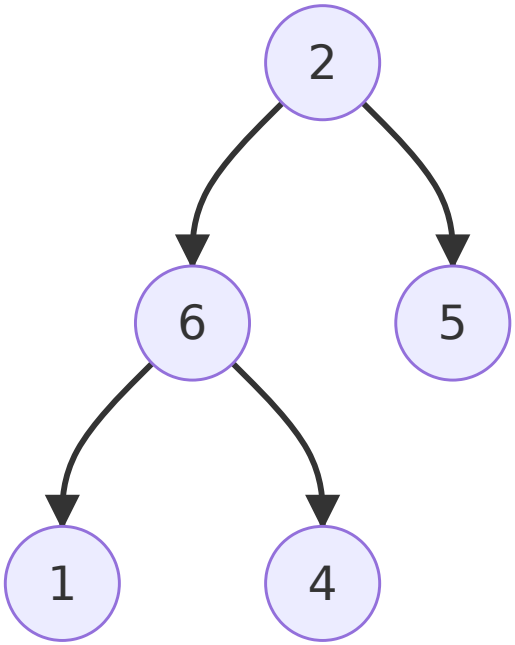
| 7 | 6 | 5 | 1 | 4 | 2 || 8 | 10 | 12 | 16

Exemple de tri par tas (10/13)

| 7 | 6 | 5 | 1 | 4 | 2 || 8 | 10 | 12 | 16

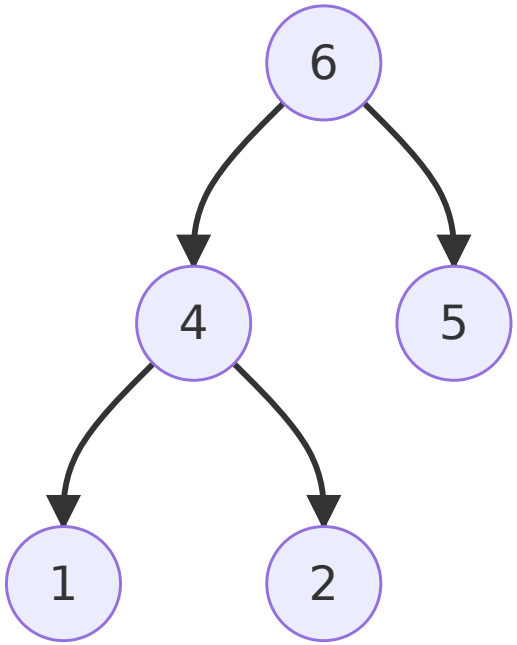
But: Trier les tas.

- Échanger la racine, 7 (`max` de l'arbre) avec 2.
- Traiter la racine.



But: Trier les tas.

- `2 <=> max(2, 6, 5)`.
- `2 <=> max(2, 1, 4)`.

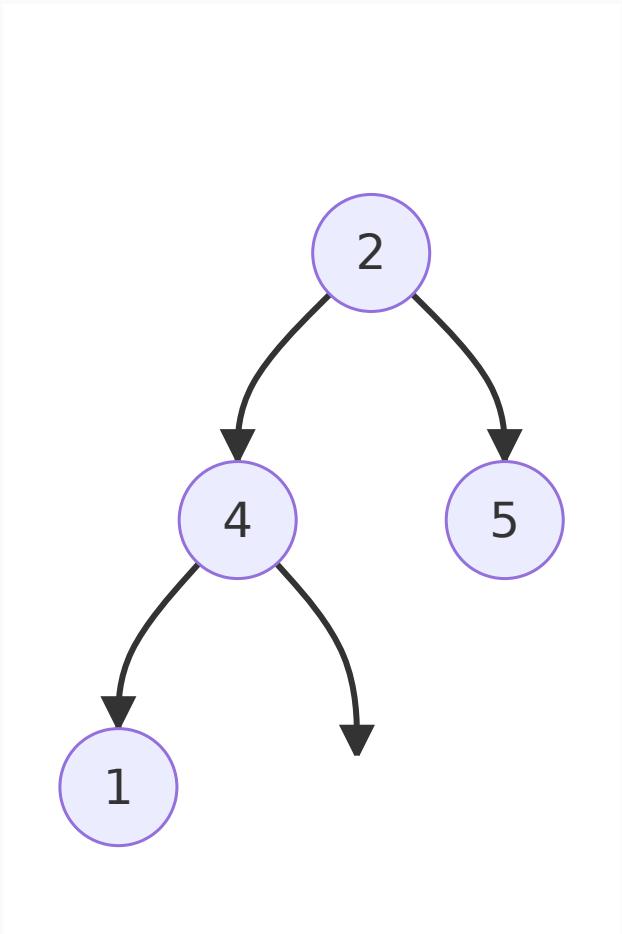


Exemple de tri par tas (11/13)

`| 6 | 4 | 5 | 1 | 2 || 8 | 10 | 12 | 16`

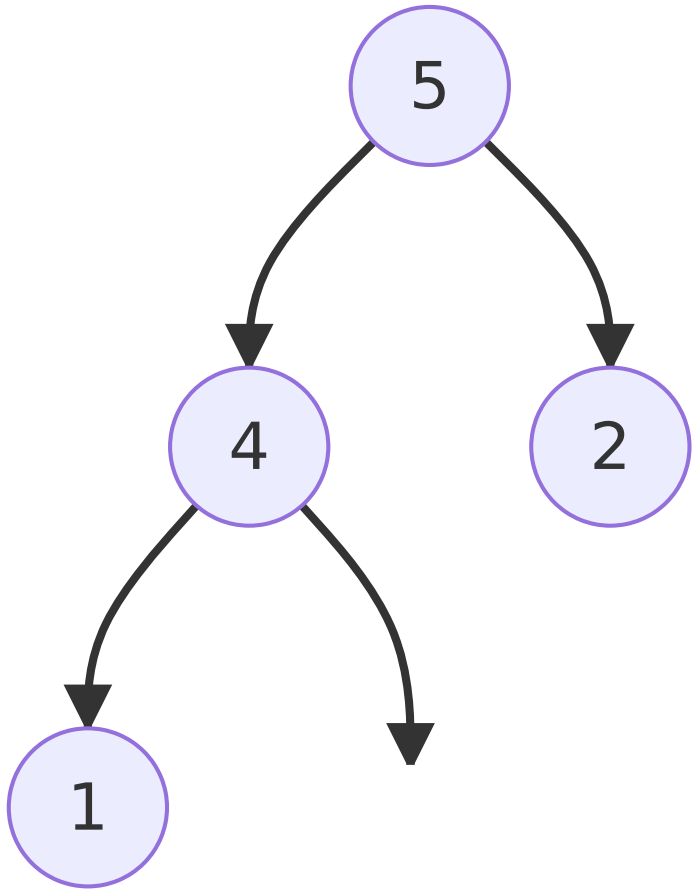
But: Trier les tas.

- Échanger la racine, 6 (`max` de l'arbre) avec 2.
- Traiter la racine.



But: Trier les tas.

- `2 <=> max(2, 4, 5).`
- `2 <=> max(2, 1, 4).`

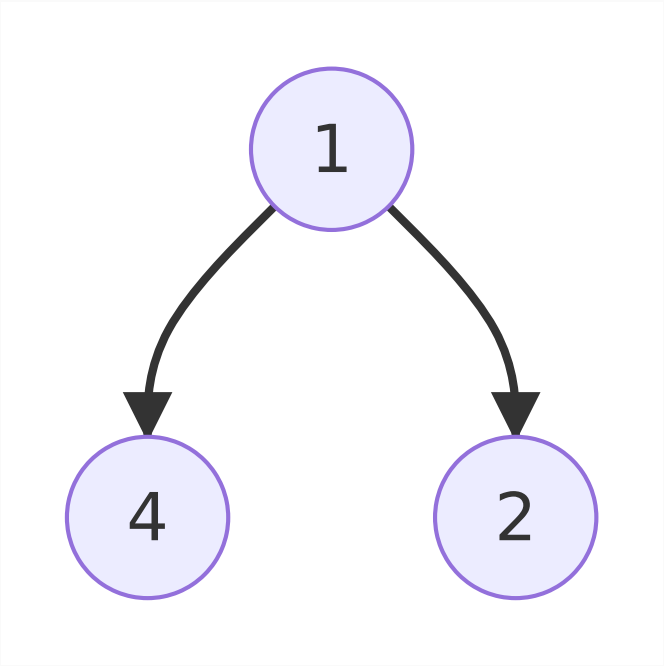


Exemple de tri par tas (12/13)

| 5 | 4 | 2 | 1 || 6 | 8 | 10 | 12 | 16

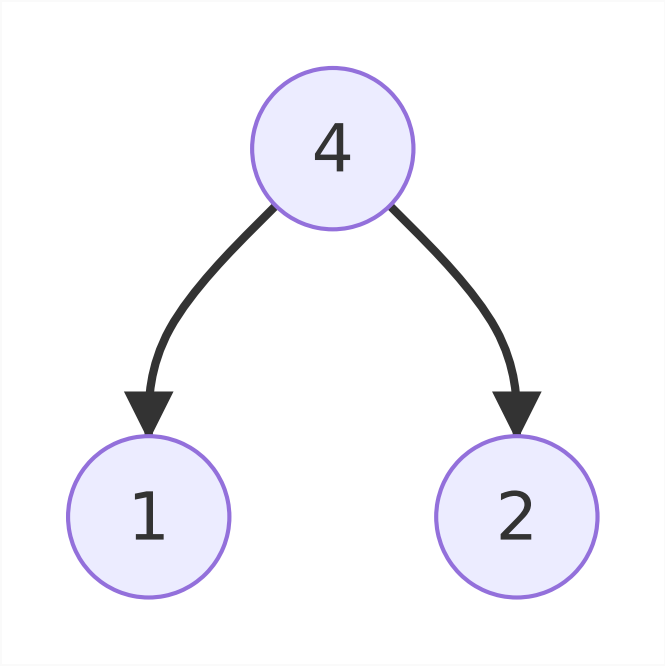
But: Trier les tas.

- Échanger la racine, 5 (`max` de l'arbre) avec 1.
- Traiter la racine.



But: Trier les tas.

- `1 <=> max(1, 4, 2)`.



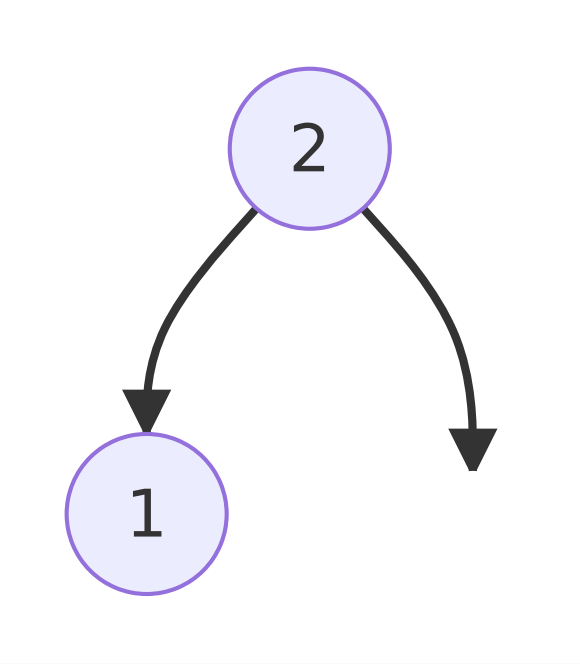
| 4 | 1 | 2 || 5 | 6 | 8 | 10 | 12 | 16

Exemple de tri par tas (13/13)

| 4 | 1 | 2 || 5 | 6 | 8 | 10 | 12 | 16

But: Trier les tas.

- Échanger la racine, 4 (`max` de l'arbre) avec 2.
- Traiter la racine.



But: Trier les tas. Plus rien à trier

- On fait les 2 dernières étapes en vitesse.
- Échange 2 avec 1.
- Il reste que 1. GGWP!

| 1 | 2 | 4 | 5 | 6 | 8 | 10 | 12 | 16

Exercice (10min)

- Trier par tas le tableau

  | 1 | 2 | 4 | 5 | 6 | 8 | 10 | 12 | 16

- Mettez autant de détails que possible.
- Que constatez-vous?
- Postez le résultat sur matrix.

L'algorithme du tri par tas (1/2)

Deux étapes

1. Entassement: transformer l'arbre en tas.
2. Échanger la racine avec le dernier élément et entasser la racine.

Pseudo-code d'entassement de l'arbre (15 min, matrix)

L'algorithme du tri par tas (1/2)

Deux étapes

1. Entassement: transformer l'arbre en tas.
2. Échanger la racine avec le dernier élément et entasser la racine.

Pseudo-code d'entassement de l'arbre (15 min, matrix)

```
rien tri_par_tas(tab)
    entassement(tab)
    échanger(tab[0], tab[size(tab)-1])
    pour i de size(tab)-1 à 2
        tamisage(tab, 0)
        échanger(tab[0], tab[i-1])

rien entassement(tab)
    pour i de size(tab)/2-1 à 0
        tamisage(tab, i)

rien tamisage(tab, i)
    ind_max = ind_max(tab, i, gauche(i), droite(i))
    si i != ind_max
        échanger(tab[i], tab[ind_max])
        tamisage(tab, ind_max)
```

L'algorithme du tri par tas (2/2)

- Fonctions utilitaires

```
entier ind_max(tab, i, g, d)
    ind_max = i
    si tab[ind_max] < tab[g] && size(tab) > g
        ind_max = g
    si tab[ind_mx] < tab[d] && size(tab) > d
        ind_max = d
    retourne ind_max

entier gauche(i)
    retourne 2 * i + 1

entier droite(i)
    retourne 2 * i + 2
```