Algorithme 2

Tas

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

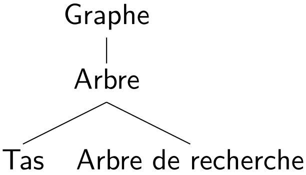
Licence 2 Informatique – Semestre 4

Plan

Plan

Contexte du cours

- Structures de données non linéaires : graphes, arbres, tas, arbres de recherche et algorithmiques associés.



- Applications :
  - Internet : recherche internet, routage, etc.
  - Informatique bas niveau : conception de circuit, système de fichiers, etc.
  - Multimédia : compression d'images, de sons, de vidéos.
  - etc.

Raisons pour étudier l'algorithmique

- Résoudre des problèmes difficiles.
- Stimulation intellectuelle.
- Être un meilleur programmeur.

Programme du cours

- Tas : tri par tas, file de priorités.
- Arbre de recherche : parcours, arbre auto-équilibré, rouge-noir, B-arbre, splay.
- Graphe : parcours, arbres couvrants, plus court chemin, flots, couplage.

Spécificités pédagogiques

- Un QCM noté de 10 minutes en milieu de chaque séance de CM.
- Pour chaque algorithme :
  - mécanisme algorithmique / pseudo-code
  - invariant pour prouver la validité
  - analyse de complexité
- TD et TP associés au CM : préparation (pseudo-code, invariant, complexité), puis implémentation et validation.
- Une évaluation blanche et facultative proposée à chaque séance de TP.

Environnements particuliers

Question

Une question simple pour chaque thème d'un CM (3 minutes par question).

Notion algorithmique

Théorie algorithmique : récapitulatif sur les tris, technique d'analyse de complexité, technique de preuve de validité, etc.

Notion mathématique

Rappel ou résultat basique mathématique.

Curiosité

Point intéressant ou notion avancée (hors programme).

Évaluation

- QCMs en séance de CM ($5 \times 2 = 10$ %, les 5 meilleures notes).
- 2 projets individuels ($5 + 10 = 15$ %) : 5 séances de TP, évaluation automatique (validité, avertissements du compilateur, fuite mémoire et accès mémoire invalide, indentation).
- Épreuve de TP de 3h (25 %).
- DS 1 et DS 2 ($25 + 25 = 50$ %) de 1h30 et 2h.
- Test de seconde chance (optionnel) de 3h : 33 % de la note finale (on ignorera la note obtenue si elle est moins bonne).

Résumé du syllabus

- Tous les supports PDF sont sur `http://lccanon.free.fr/teaching/` (lien accessible sur le cours Moodle).
- Le cours Moodle regroupe les annonces, les rendus des TP et projets et les notes (clé d'inscription : `algo2`).
- Voir le syllabus en ligne pour toutes les informations sur ce cours.
  - Description du contenu du cours.
  - Calendrier.
  - Modalités d'évaluation.
  - Rappel des règles.

Plan

Propriétés des algorithmes de tri

Notion algorithmique

Complexité en temps
- **au pire cas** Nombre d'opérations maximal garanti pour tous les tableaux.
- **en moyenne** Nombre d'opérations pour des tableaux usuels.
- **meilleur cas** Nombre d'opérations pour des tableaux déjà triés (ou caractère *adaptatif* pour des tableaux partiellement triés).

Complexité en mémoire Utilisation de données supplémentaires (ou *en place*).

Stabilité L'ordre des éléments égaux est conservé.

En ligne Tri progressif d'éléments au fur et à mesure de leurs arrivées.

Exemple d'algorithmes de tri

Notion algorithmique

Tri	pire cas	moyenne	meilleur cas	en place	stable	en ligne
insertion	$O(n^2)$	$O(n^2)$	$O(n)$	✓	✓	✓
bulles	$O(n^2)$	$O(n^2)$	$O(n)$	✓	✓	✗
fusion	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	✗	✓	✗
rapide	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	✓	✗	✗

Contexte d'utilisation des tas

Utilisation pour le tri :

- tri en place, comme le tri rapide ;
- efficace (complexité dans le pire cas), comme le tri fusion.

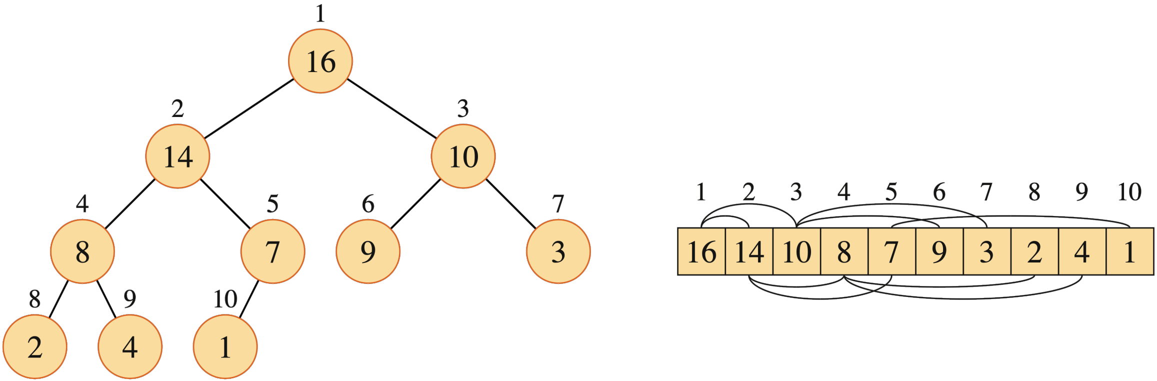
Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Exemple de tas



(a) (b)

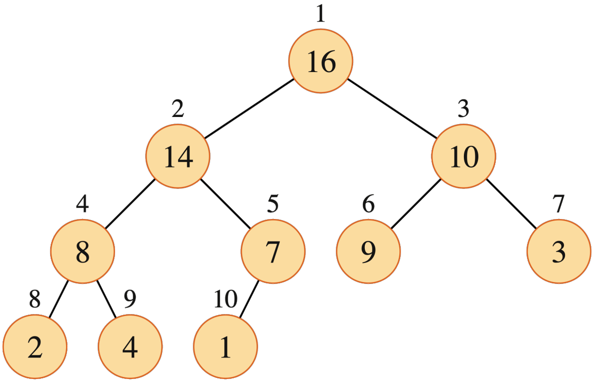
Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

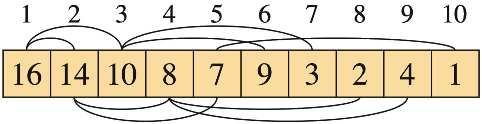
Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droit}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Exemple de tas



(a)



(b)

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

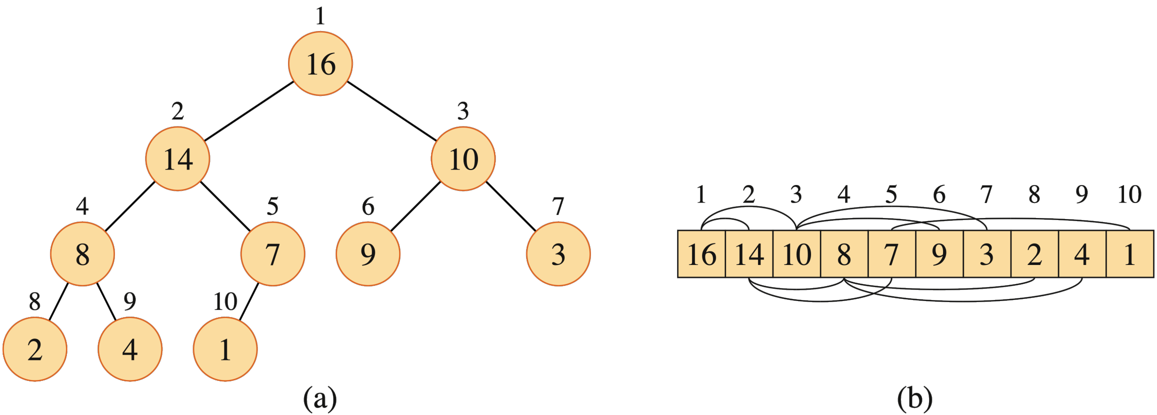
Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Définitions

- Un *arbre binaire* est un ensemble de nœuds, chacun connecté à au plus un *parent* et deux *enfants* (gauche et droit).
- La *racine* est le seul nœud sans parent et les *feuilles* sont les nœuds sans enfants.
- Un *tas* (*heap*) est un arbre binaire presque complet (tous les niveaux sont complets, excepté le dernier où toutes les feuilles sont rangées à gauche).
- La *hauteur* d'un nœud est le plus grand nombre d'arcs qui le sépare d'une feuille qui en est une descendante. La hauteur d'un tas est celle de sa racine.
- Un tas peut être stocké dans un tableau A de longueur $A.n$ tel que :
  - La racine est $A[1]$.
  - Le parent d'un nœud i est $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
  - Le fils gauche d'un nœud i est $\textsc{Gauche}(i) = 2i$.
  - Le fils droit d'un nœud i est $\textsc{Droite}(i) = 2i + 1$.
- Propriété de *tas max* : $\forall i, A[\textsc{Parent}(i)] \geq A[i]$ (sens inverse pour un *tas min*).

Exemple de tas



Question

Lequel des tableaux suivants ne représente pas un tas max ?

1. (19,18,17,16,15,14,13,12,11,10)
2. (10,10,10,10,10,10,10,10,10,10)
3. (34,30,29,27,25,17,16,19,22,24)
4. (30,27,23,17,16,15,13,14,18,11)

Question

Lequel des tableaux suivants ne représente pas un tas max ?

1. (19,18,17,16,15,14,13,12,11,10)
2. (10,10,10,10,10,10,10,10,10,10)
3. (34,30,29,27,25,17,16,19,22,24)
4. (30,27,23,17,16,15,13,14,18,11) ✓car $17 < 18$

Plan

Description d'Entasser-Max (*heapify*)

- Algorithme utilisé dans la manipulation des tas max.
- Suppose que les sous-arbres du nœud i sont des tas max.
- Garantit que le sous-arbre enraciné en i est un tas max.

Pseudo-code d'ENTASSER-MAX

```
ENTASSER-MAX(A, i)
--------------------------------------------
g ← GAUCHE(i)
d ← DROITE(i)
si g ≤ A.n et A[g] > A[i] alors
   max ← g
sinon max ← i
si d ≤ A.n et A[d] > A[max] alors
   max ← d
si max ≠ i alors
   échanger A[i] et A[max]
   ENTASSER-MAX(A, max)
```

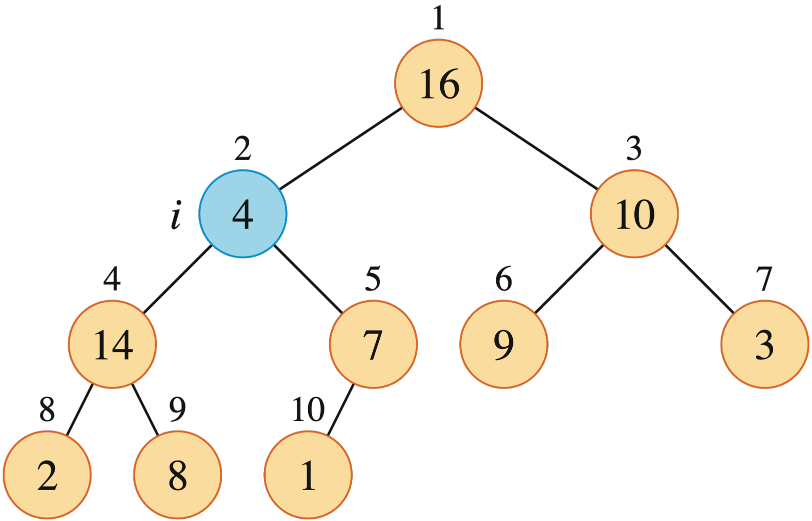
- Compare $A[i]$, $A[\text{GAUCHE}(i)]$ et $A[\text{DROITE}(i)]$.
- Si nécessaire, échange avec le plus grand des enfants pour préserver la propriété de tas max.
- Continue ce processus jusqu'à ce qu'aucun échange ne soit nécessaire ou possible.

Pseudo-code d'ENTASSER-MAX

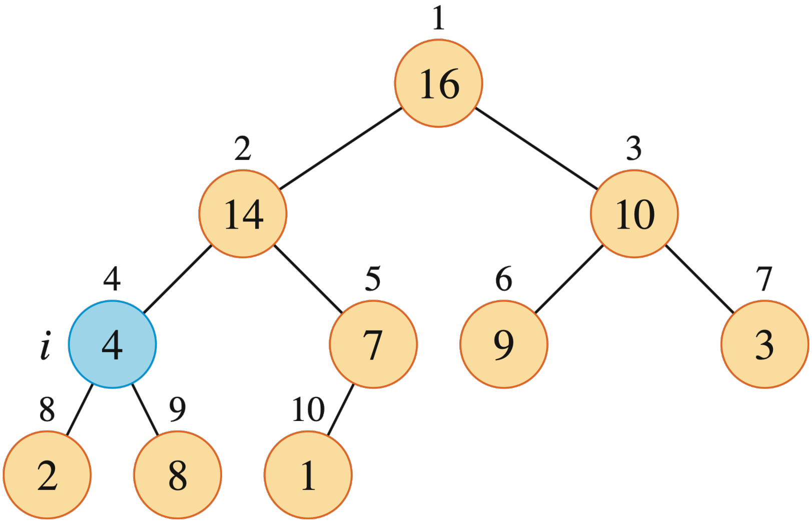
```
ENTASSER-MAX(A, i)

g ← GAUCHE(i)
d ← DROITE(i)
si g ≤ A.n et A[g] > A[i] alors
   max ← g
sinon max ← i
si d ≤ A.n et A[d] > A[max] alors
   max ← d
si max ≠ i alors
   échanger A[i] et A[max]
   ENTASSER-MAX(A, max)
```

- Compare $A[i]$, $A[\text{GAUCHE}(i)]$ et $A[\text{DROITE}(i)]$.
- Si nécessaire, échange avec le plus grand des enfants pour préserver la propriété de tas max.
- Continue ce processus jusqu'à ce qu'aucun échange ne soit nécessaire ou possible.

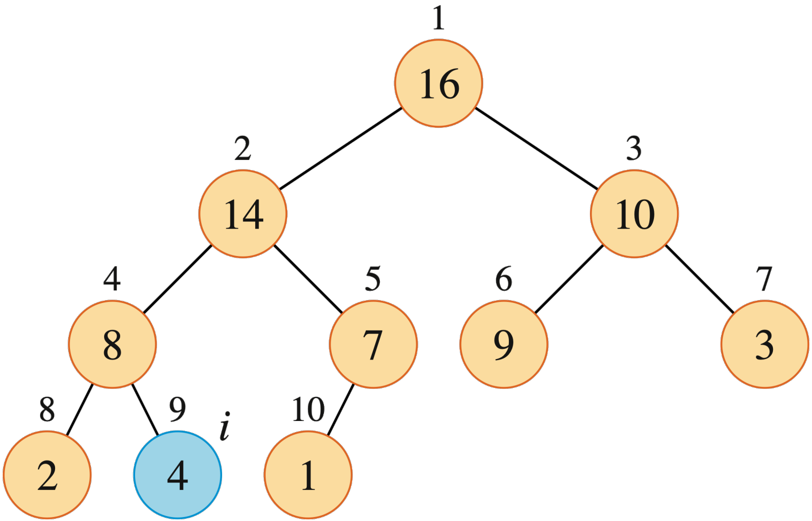
Exemple



Exemple



Exemple



Notation de Landau

Notion algorithmique

Symbole	description	exemples
$O(g(n))$	Fonctions f telles que $\exists c, n_0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0$	$10n = O(n^2)$, $n^2 + n = O(n^2)$
$\Omega(g(n))$	Fonctions f telles que $\exists c, n_0 : 0 \leq cg(n) \leq f(n), \forall n \geq n_0$	$n^2 - n = \Omega(n^2)$, $n^3 = \Omega(n^2)$
$\Theta(g(n))$	Fonctions f telles que $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$	$\frac{n^2}{10} + n = \Theta(n^2)$

L'optimal est donc la complexité constante $O(1)$ (c'est-à-dire $\Theta(1)$ car la complexité d'une opération est rarement 0).

Analyse de complexité

- Temps d'exécution ou nombre d'opérations pour chaque nœud considéré : $\Theta(1)$ (2 calculs d'index, 4 comparaisons, 5 affectations et 1 échange).
- Pour un sous-arbre de taille n, appel récursif avec un sous-arbre de taille au plus $2n/3$: sous-arbre de gauche complet plus haut que celui de droite.
- $T(n) \leq T(2n/3) + \Theta(1)$.
- Cas 2 du théorème général : $T(n) = O(\log n)$.

Question

Lequel des tableaux suivants correspond à l'action $\textsc{Entasser-Max}(A, 3)$ sur le tableau $A = (27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 8, 0)$?

1. (27,17,3,16,13,10,1,5,7,12,4,8,8,0)
2. (27,17,10,16,13,8,1,5,7,12,4,8,3,0)
3. (27,17,10,16,13,3,1,5,7,12,4,8,8,0)
4. (27,17,10,16,13,8,1,5,7,12,4,3,8,0)

Question

Lequel des tableaux suivants correspond à l'action $\textsc{Entasser-Max}(A, 3)$ sur le tableau $A = (27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 8, 0)$?

1. (27,17,3,16,13,10,1,5,7,12,4,8,8,0)
2. (27,17,10,16,13,8,1,5,7,12,4,8,3,0)
3. (27,17,10,16,13,3,1,5,7,12,4,8,8,0)
4. (27,17,10,16,13,8,1,5,7,12,4,3,8,0) ✓

Plan

Contruction-Tas-Max

Algorithme qui construit un tas max à partir d'un tableau de valeurs dans un ordre arbitraire :

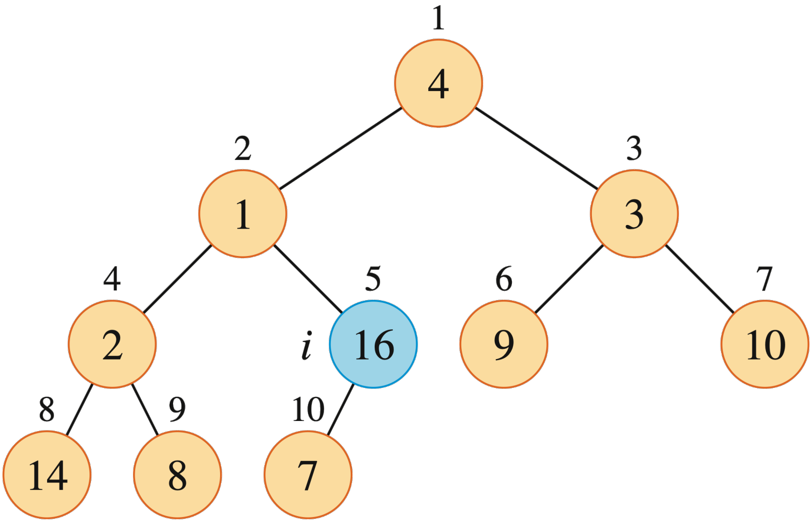
Construire-Tas-Max(A)

pour $i = \lfloor n/2 \rfloor$ **decr jusqu'à** 1
    Entasser-Max(A, i)

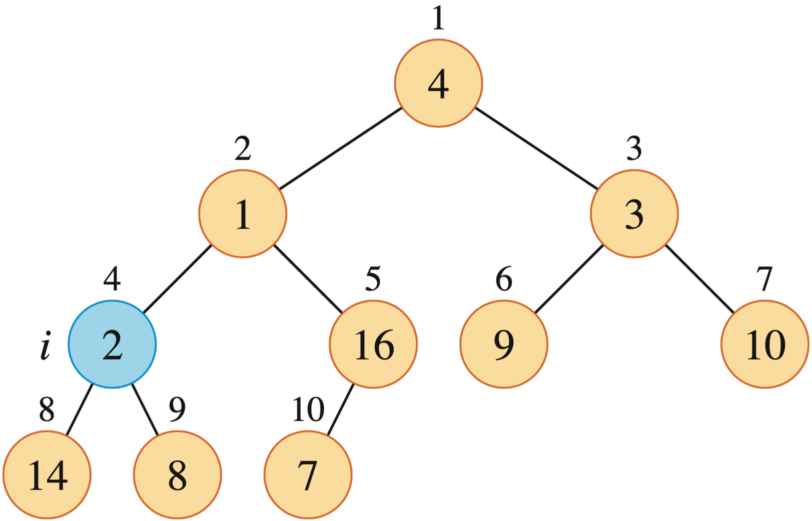
Exemple

- On considère le tableau $(4, 1, 3, 2, 16, 9, 10, 14, 8, 7)$
- La boucle commencera à l'indice 5.
- On entassera successivement les valeurs 16, 2, 3, 1, puis 4.

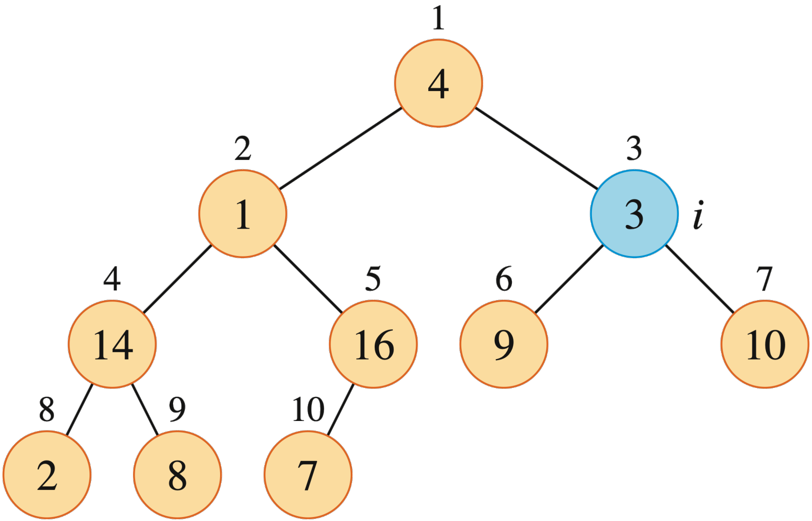
Exemple



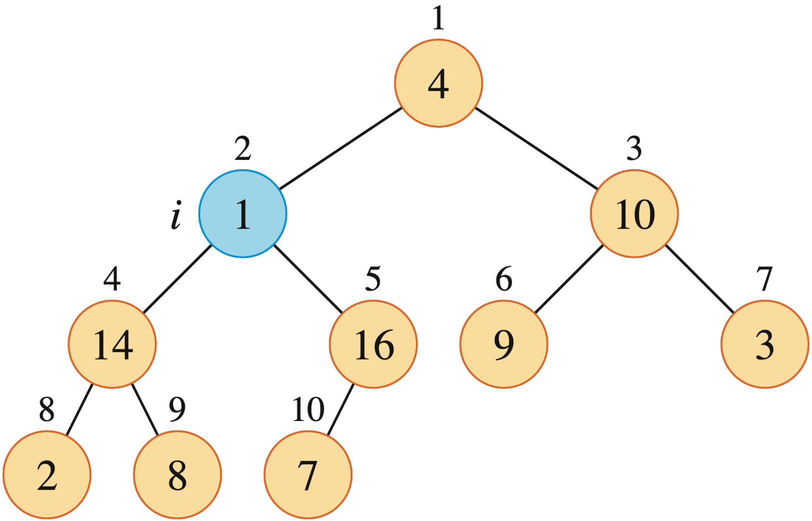
Exemple



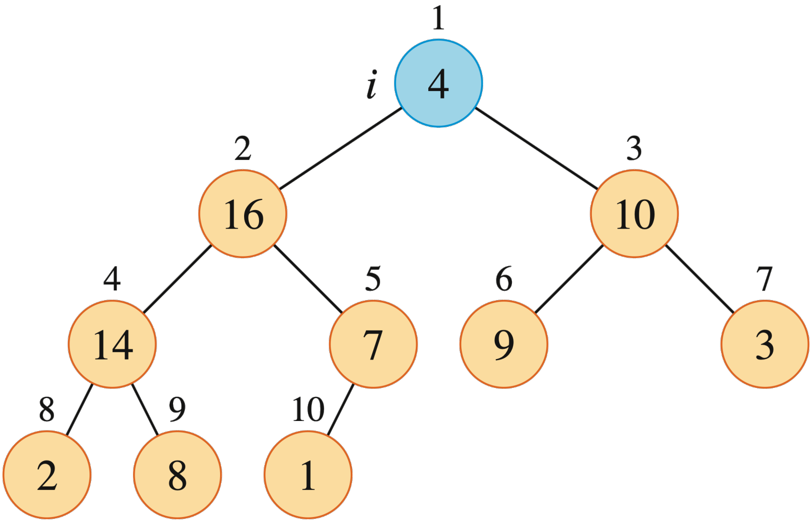
Exemple



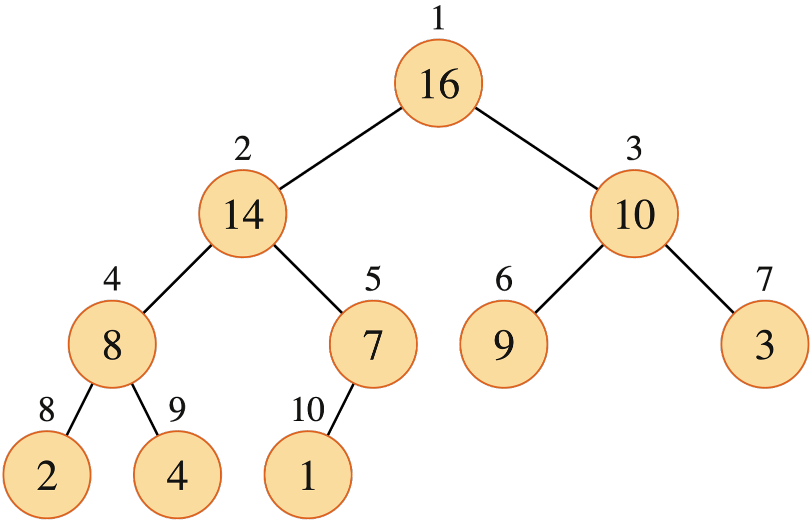
Exemple



Exemple



Exemple



Technique de preuve de validité

Notion algorithmique

On peut prouver qu'un algorithme fonctionne correctement grâce à un *invariant de boucle* : c'est une propriété qui est vrai au début de chaque itération. On procède ensuite par récurrence/induction :

Initialisation On prouve que l'invariant est vrai à la première itération de la boucle.

Conservation On suppose que l'invariant est vrai pour une itération donnée et on prouve qu'il reste vrai à l'itération suivante.

Terminaison On vérifie qu'à la dernière itération, on peut déduire la validité de l'algorithme à partir de l'invariant.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle, chaque noeud $i+1$, $i+2$, ..., n est la racine d'un tas max.

Initialisation pour $i = \lfloor n/2 \rfloor$ Tous les nœuds d'indice supérieur sont des feuilles (pas prouvé ici), et donc des tas max.

Conservation On suppose que l'invariant est vrai pour un indice i. Les nœuds $\textsc{Gauche}(i)$ et $\textsc{Droite}(i)$ sont donc des racines de tas max. Dans ce cas, $\textsc{Entasser-Max}$ garantit que le nœud i est une racine d'un tas max, ce qui établit l'invariant pour l'itération suivante.

Terminaison À la fin, $i = 0$. D'après l'invariant, tous les nœuds sont des racines de tas mas, y compris le nœud 1.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle, chaque noeud $i+1$, $i+2$, …, n est la racine d'un tas max.

Initialisation pour $i = \lfloor n/2 \rfloor$ Tous les nœuds d'indice supérieur sont des feuilles (pas prouvé ici), et donc des tas max.

Conservation On suppose que l'invariant est vrai pour un indice i. Les nœuds GAUCHE(i) et DROITE(i) sont donc des racines de tas max. Dans ce cas, ENTASSER-MAX garantit que le nœud i est une racine d'un tas max, ce qui établit l'invariant pour l'itération suivante.

Terminaison À la fin, $i = 0$. D'après l'invariant, tous les nœuds sont des racines de tas mas, y compris le nœud 1.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle, chaque noeud $i+1$, $i+2$, ..., n est la racine d'un tas max.

Initialisation pour $i = \lfloor n/2 \rfloor$ Tous les nœuds d'indice supérieur sont des feuilles (pas prouvé ici), et donc des tas max.

Conservation On suppose que l'invariant est vrai pour un indice i. Les nœuds $\textsc{Gauche}(i)$ et $\textsc{Droite}(i)$ sont donc des racines de tas max. Dans ce cas, $\textsc{Entasser-Max}$ garantit que le nœud i est une racine d'un tas max, ce qui établit l'invariant pour l'itération suivante.

Terminaison À la fin, $i = 0$. D'après l'invariant, tous les nœuds sont des racines de tas mas, y compris le nœud 1.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle, chaque noeud $i+1$, $i+2$, ..., n est la racine d'un tas max.

Initialisation pour $i = \lfloor n/2 \rfloor$ Tous les nœuds d'indice supérieur sont des feuilles (pas prouvé ici), et donc des tas max.

Conservation On suppose que l'invariant est vrai pour un indice i. Les nœuds $\textsc{Gauche}(i)$ et $\textsc{Droite}(i)$ sont donc des racines de tas max. Dans ce cas, $\textsc{Entasser-Max}$ garantit que le nœud i est une racine d'un tas max, ce qui établit l'invariant pour l'itération suivante.

Terminaison À la fin, $i = 0$. D'après l'invariant, tous les nœuds sont des racines de tas mas, y compris le nœud 1.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - Entasser-Max prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de Construire-Tas-Max est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - Entasser-Max prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de Construire-Tas-Max est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - ENTASSER-MAX prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de CONSTRUIRE-TAS-MAX est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - ENTASSER-MAX prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de CONSTRUIRE-TAS-MAX est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - Entasser-Max prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de Construire-Tas-Max est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - Entasser-Max prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de Construire-Tas-Max est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Analyse de complexité

- Majorant naïf du temps d'exécution : $n/2$ itérations coûtant $O(\log n)$ mène à $O(n \log n)$.
- Observations de départ :
  - il existe au plus $\lceil n/2^{h+1} \rceil$ nœuds dont la hauteur est h ;
  - $\textsc{Entasser-Max}$ prend un temps $O(h)$ pour un nœud de hauteur h ;
  - la hauteur d'un tas est $\lfloor \log_2 n \rfloor$.
- Le coût total de $\textsc{Construire-Tas-Max}$ est donc inférieur à :

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

- Le coût est donc linéaire.

Notion mathématique

Pour un réel $x \neq 1$:

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$

Si $|x| < 1$:

$$\sum_{k=0}^{\infty} x^k = \lim_{k \to \infty} \frac{x^{k+1} - 1}{x - 1} = \frac{1}{1 - x}$$

Après dérivation et multiplication par x :

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1 - x)^2}$$

En substituant $x = 1/2$:

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{1/2}{(1/2)^2} = 2$$

Question

Quel est le tas max issu de l'action CONSTRUIRE-TAS-MAX sur le tableau $(5, 3, 17, 10, 84, 19, 6, 22, 9)$?

1. $(84, 22, 19, 17, 3, 10, 6, 5, 9)$
2. $(84, 19, 22, 17, 3, 10, 6, 5, 9)$
3. $(84, 22, 19, 10, 3, 17, 6, 5, 9)$
4. $(84, 19, 22, 10, 3, 17, 6, 5, 9)$

Question

Quel est le tas max issu de l'action Construire-Tas-Max sur le tableau $(5, 3, 17, 10, 84, 19, 6, 22, 9)$?

1. $(84, 22, 19, 17, 3, 10, 6, 5, 9)$
2. $(84, 19, 22, 17, 3, 10, 6, 5, 9)$
3. $(84, 22, 19, 10, 3, 17, 6, 5, 9)$ ✓
4. $(84, 19, 22, 10, 3, 17, 6, 5, 9)$

Plan

Introduction

Structure de tas

Conservation de la structure de tas

Construction d’un tas

Tri par tas

Conclusion et référence

TRI-PAR-TAS

Algorithme qui tri un tableau de valeurs dans un ordre arbitraire :

TRI-PAR-TAS(A)

CONSTRUIRE-TAS-MAX(A)
pour $i = n$ **decr jusqu'à** 2
  échanger $A[1]$ et $A[i]$
  $A.n \leftarrow A.n - 1$
  ENTASSER-MAX($A, 1$)

Fonctionnement :

- Construit (en place) un tas max à partir du tableau A.
- À chaque itération, extrait la valeur maximale du tas ($A[1]$) et la place à sa position finale ($A[i]$) par échange.
- Après chaque échange, les nœuds 2 et 3 sont toujours des racines de tas max, mais plus forcément le nœud 1 (ENTASSER-MAX).
- S'arrête lorsque le tas considéré n'a plus qu'une seule valeur, la valeur minimale.

TRI-PAR-TAS

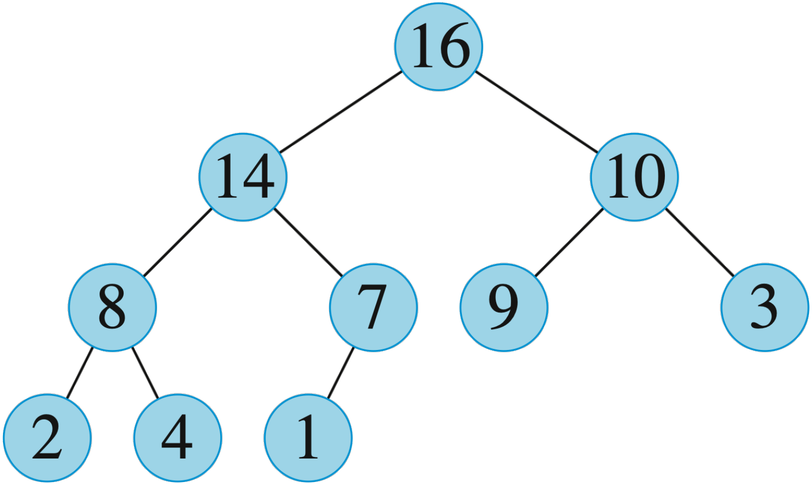
Algorithme qui tri un tableau de valeurs dans un ordre arbitraire :

```
TRI-PAR-TAS(A)
CONSTRUIRE-TAS-MAX(A)
pour i = n decr jusqu'à 2
    échanger A[1] et A[i]
    A.n ← A.n − 1
    ENTASSER-MAX(A, 1)
```

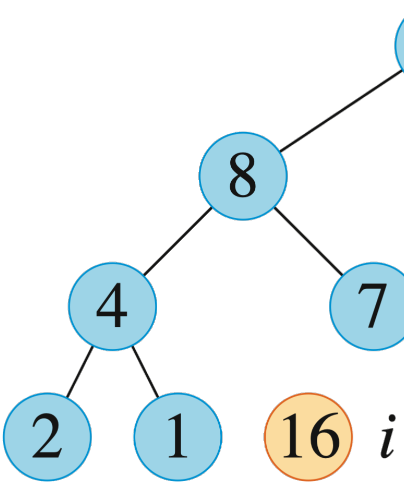
Fonctionnement :

- Construit (en place) un tas max à partir du tableau A.
- À chaque itération, extrait la valeur maximale du tas ($A[1]$) et la place à sa position finale ($A[i]$) par échange.
- Après chaque échange, les nœuds 2 et 3 sont toujours des racines de tas max, mais plus forcément le nœud 1 (ENTASSER-MAX).
- S'arrête lorsque le tas considéré n'a plus qu'une seule valeur, la valeur minimale.

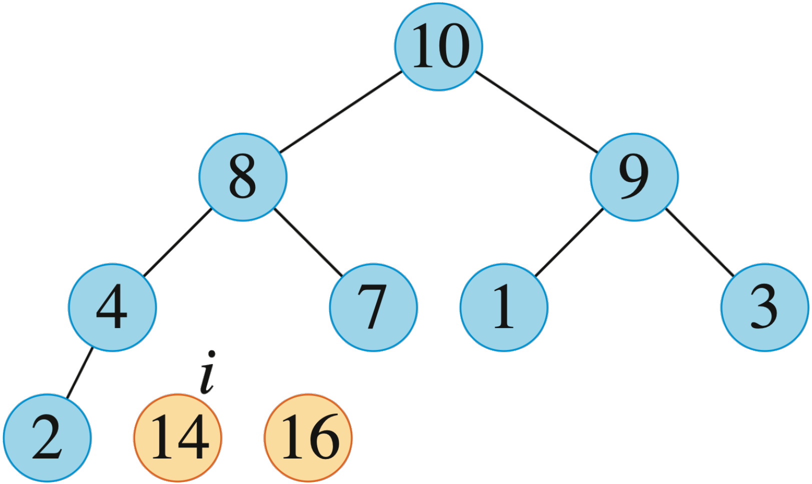
Exemple



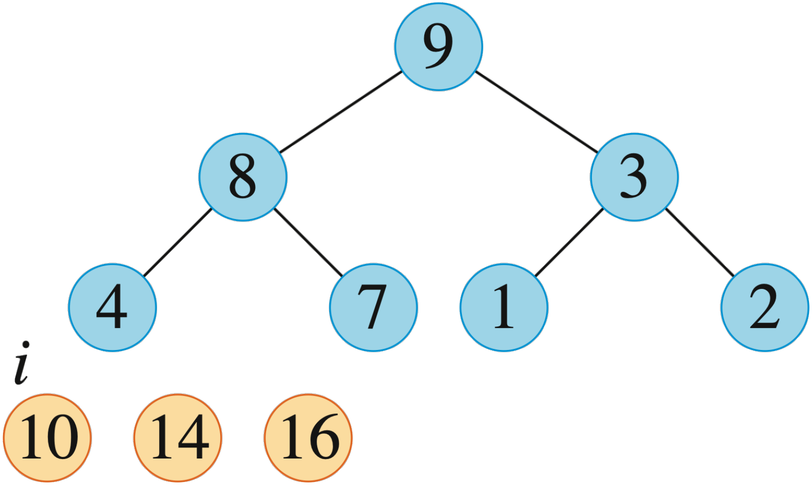
Exemple



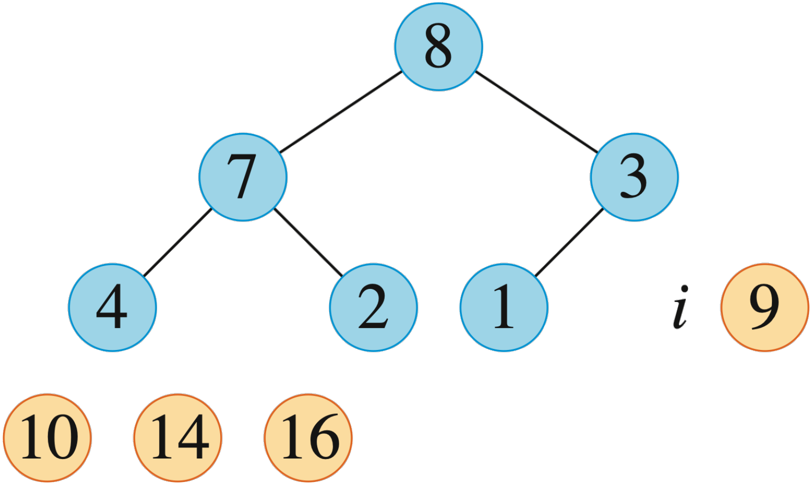
Exemple



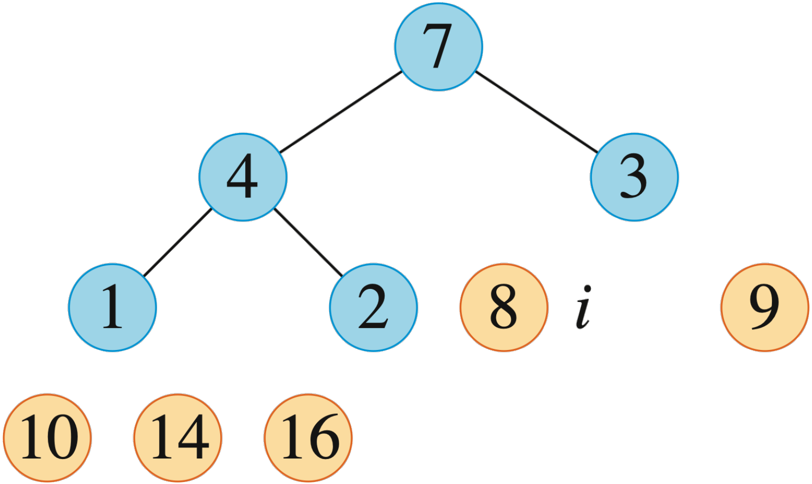
Exemple



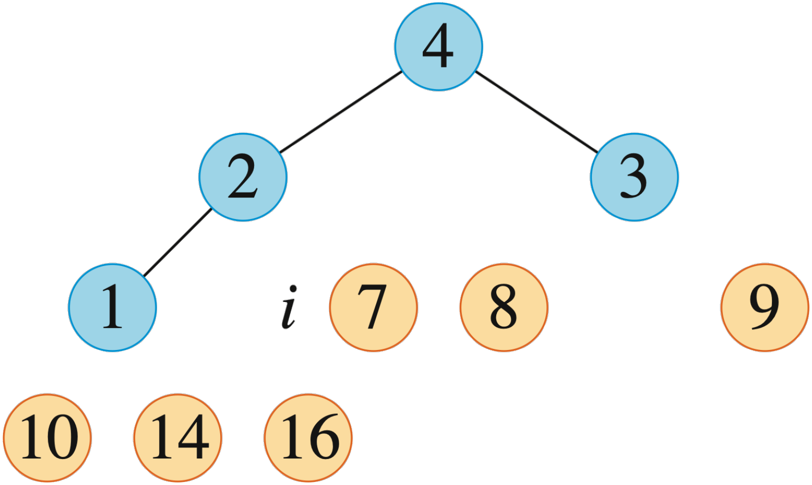
Exemple



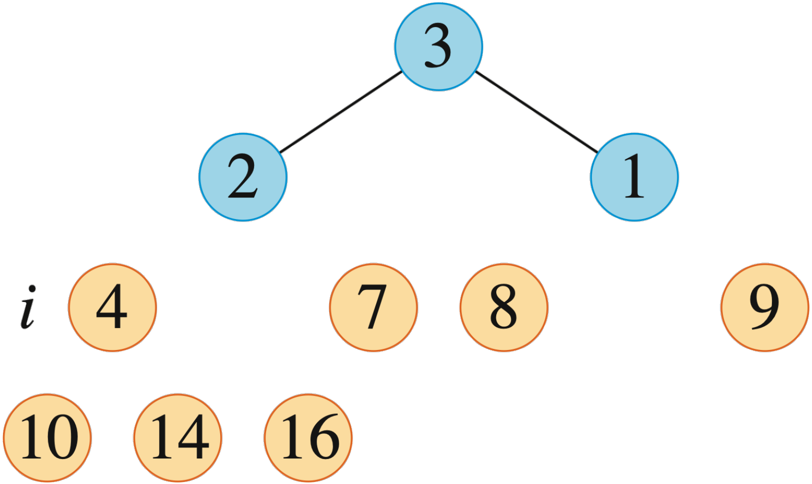
Exemple



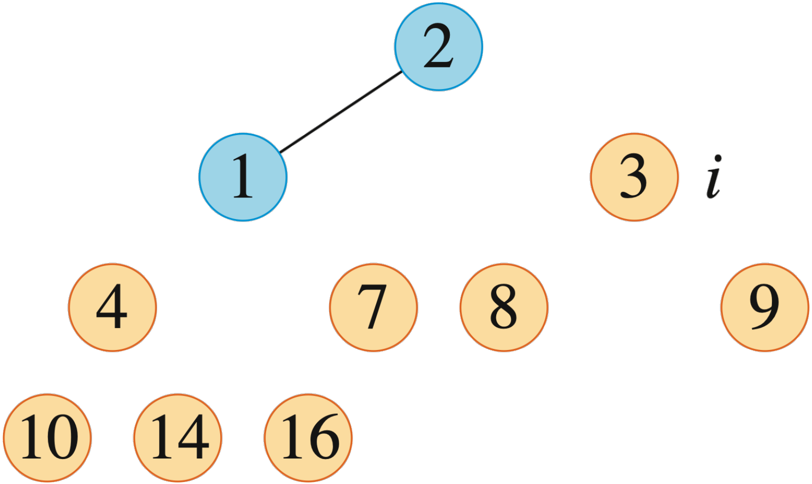
Exemple



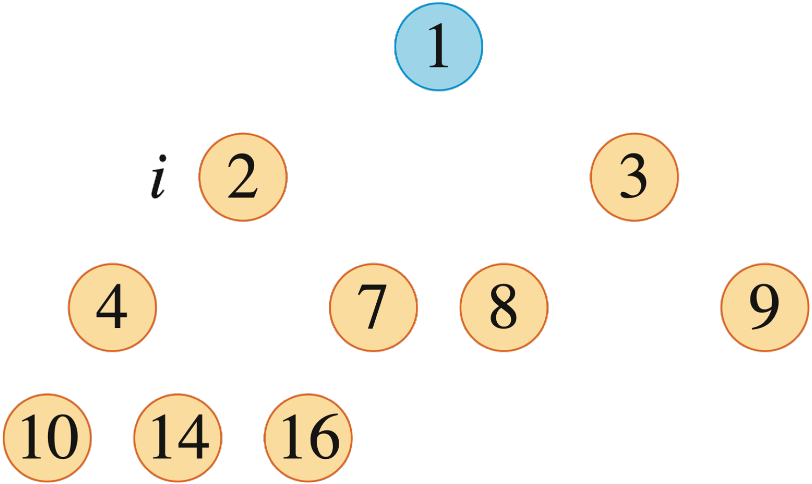
Exemple



Exemple



Exemple



Analyse de complexité

- CONSTRUIRE-TAS-MAX(A) : $O(n)$.
- $n - 1$ itérations.
- Échange des valeurs : $\Theta(1)$.
- ENTASSER-MAX : $O(\log n)$.
- Temps total : $O(n \log n)$.

Dans la pratique

Curiosité

- Meilleur cas du tri par tas en $O(n \log n)$ seulement.
- Mauvaise gestion du cache : les données manipulées en même temps ont des localités différentes.
- Le tri rapide est souvent meilleur mais sensible à certains motifs (pire cas).
- *introsort* (utilisé dans de nombreuses bibliothèques C, C++, .NET, Go, Java) se base sur le tri rapide et évite les pires cas en utilisant le tri par tas en dernier recourt.

Question

Combien d'échanges sont réalisés pour trier le tableau $(5, 13, 2, 25)$ avec TRI-PAR-TAS ?

1. 8
2. 9
3. 10
4. 11

Question

Combien d'échanges sont réalisés pour trier le tableau $(5, 13, 2, 25)$ avec Tri-Par-Tas ?

1. 8 ✓(3 pour la construction du tas, 3 pour le tri, 2 pour l'entassement)
2. 9
3. 10
4. 11

Plan

Résumé

Contenu

- Tas : structure de données adaptée pour trouver l'élément maximal.
- Tri par tas (*heapsort*) : tri en place avec une complexité optimale dans le pire cas.
- Méthode centrale : ENTASSER-MAX pour construire un tas, puis extraire les valeurs maximales.

Prochaines échéances

- QCM à la prochaine séance de CM (numéro étudiant nécessaire).
- Mini-projet : début la semaine du 10/2, à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.

Résumé

Contenu

- Tas : structure de données adaptée pour trouver l'élément maximal.
- Tri par tas (*heapsort*) : tri en place avec une complexité optimale dans le pire cas.
- Méthode centrale : ENTASSER-MAX pour construire un tas, puis extraire les valeurs maximales.

Prochaines échéances

- QCM à la prochaine séance de CM (numéro étudiant nécessaire).
- Mini-projet : début la semaine du 10/2, à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.

Introduction à l'algorithmique, Cormen, Leiserson, Rivest, Stein

LA référence, LE Cormen. Cité plus de 10 000 fois dans la littérature scientifique. Actualisé en 2022 (édition initiale en 2001).