TAS (ARBRE BINAIRE PARFAIT PARTIELLEMENT ORDONNE)

Les éléments d'un arbre parfait sont bien rangés :

- On commence par le niveau $l = 0$ (racine) et on passe au niveau $l + 1$ après avoir rempli le niveau l.
- Pour chaque niveau, on remplit de gauche à droite.
  Pour un arbre binaire parfait de n nœuds, on peut utiliser un tableau $T[i]$, $1 \leq i \leq n$.

  Nœud i est dans l'indice i du tableau T :
  - la racine est à l'indice 1
  - le père du nœud i est à l'indice $\lfloor i/2 \rfloor$ (pere(i))
  - le fils gauche du nœud i à l'indice $2 \times i$ (gauche(i))
  - le fils droit du nœud i à l'indice $2 \times i + 1$ (droit (i))

Si le tableau $T[i]$, $0 \leq i < n$.
Nœud i est dans l'indice i du tableau T :

- la racine est à l'indice 0
- le père du nœud i est à l'indice $\lfloor (i-1)/2 \rfloor$ (pere(i))
- le fils gauche du nœud i à l'indice $2 \times i + 1$ (gauche(i))
- le fils droit du nœud i à l'indice $2 \times i + 2$ (droit (i))

- *Nous pensons comme un arbre mais nous l'implémentons dans un tableau*
- Il est possible d'avoir ≥ 2 fils, les fonctions pour trouver les indices des fils et du père seraient différentes.

Les noeuds sont partiellement ordonnés

TAS *(heap)* est un arbre binaire parfait qui vérifie en plus les conditions suivantes :

$$T[i] \geq T[gauche(i)] \qquad \text{et} \qquad T[i] \geq T[droit(i)]$$

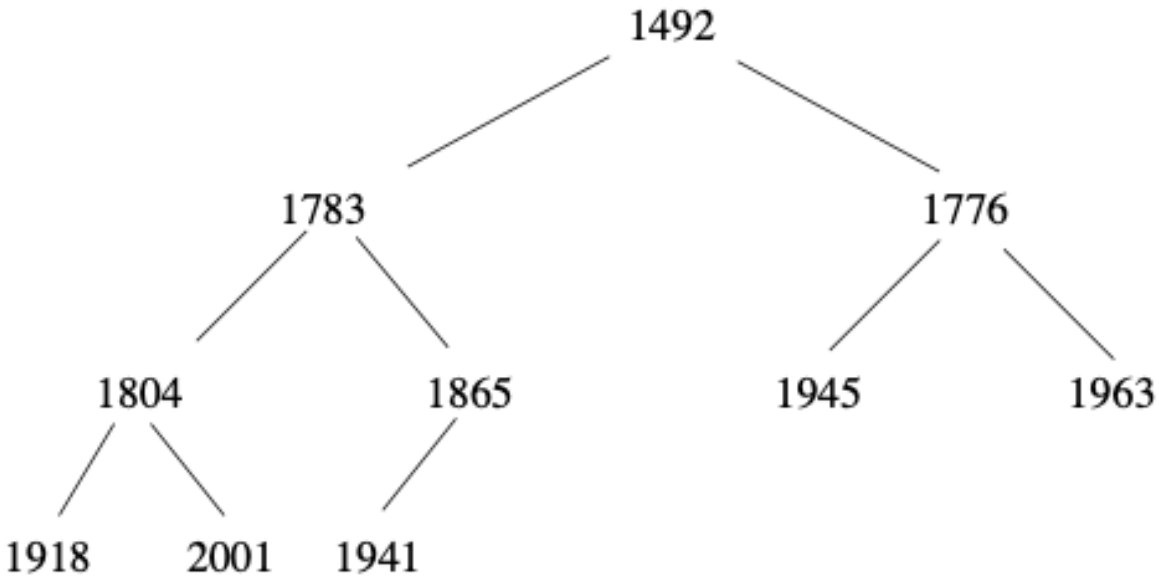
Pour chaque sous arbre du tas, la valeur de la racine est supérieures aux valeurs de ses fils.

Un tel tas est appelé tas **MAX**, car la valeur maximale du tas se trouve à la racine.

Complexité d'accéder à la valeur max $= \Theta(1)$.

D'une façon symétrique, un tas MIN peut être construit où

$$T[i] \leq T[gauche(i)] \qquad \text{et} \qquad T[i] \leq T[droit(i)]$$



1	1492
2	1783
3	1776
4	1804
5	1865
6	1945
7	1963
8	1918
9	2001
10	1941

Figure – TAS MIN

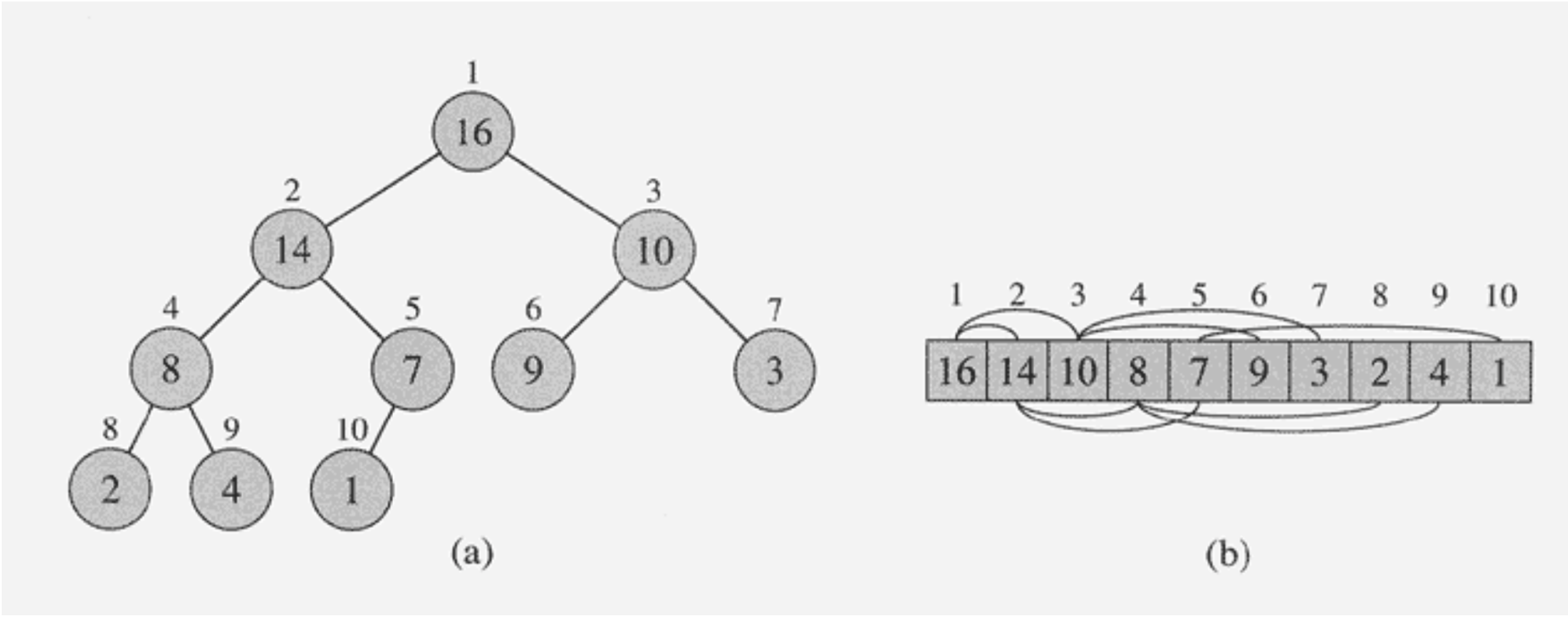


Figure – TAS MAX

Hauteur h d'un TAS de n éléments

- Au niveau l, il y a au plus 2^l nœuds.
- Hauteur (profondeur) $= max_l\{l\}$
  (Un arbre binaire avec un seul noeud est de hauteur =0)

  Les niveaux $l = 0, \cdots h-1$ sont pleins.
  Le nombre de nœuds aux niveaux $i = 0, \cdots h-1$:

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

  Au niveau h, il y a au moins 1 nœud et au plus 2^h nœuds.

$$2^h - 1 + 1 \leq n \leq 2^h - 1 + 2^h$$

$$2^h \leq n < 2^{h+1}$$

$$log_2(2^h) \leq log_2(n) < log_2(2^{h+1})$$

$$h \leq log_2 n < h + 1$$

  Les opérations proportionnelles à h sont en $\Theta(\log n)$.

Réorganiser le TAS

- $T[i]$ peut être $<$ à ses fils. Par contre les sous arbres gauche et droit sont des tas Max.
- L'algorithme réorganise le sous arbre de racine i, $1 \leq i < T.taille = n$.
- On commence par l'indice i on se dirige vers une feuille en comparant la valeur du père avec les valeurs de ses fils.
- On s'arrête lorsqu'on trouve un noeud dont la valeur est supérieure à celles de ses fils, soit on arrive à une feuille.
- Dans le pire des cas il faut vérifier à partir du niveau du noeud i ($\Theta(log_i)$) au dernier niveau $h = \Theta(logn)$

  Si i=1, n=T.taille : Complexité (pire des cas) : $\Theta(logn)$.
- On considère TAS MAX pour les exemples suivants.

ENTASSER(T,i,n)

// T est un TAS sauf éventuellement à l'indice i ; on le réorganise à partir de l'indice i et le dernier élément du TAS est à l'indice n

$l = gauche(i)$

$r = droit(i)$

if $l \leq n$ AND $T[l] > T[i]$ **then**

  $max = l$

else

  $max = i$

end if

if $r \leq n$ AND $T[r] > T[max]$ **then**

  $max = r$

end if

if $max \neq i$ **then**

  $T[i] \leftrightarrow T[max]$

  $ENTASSER(T, max, n)$

end if

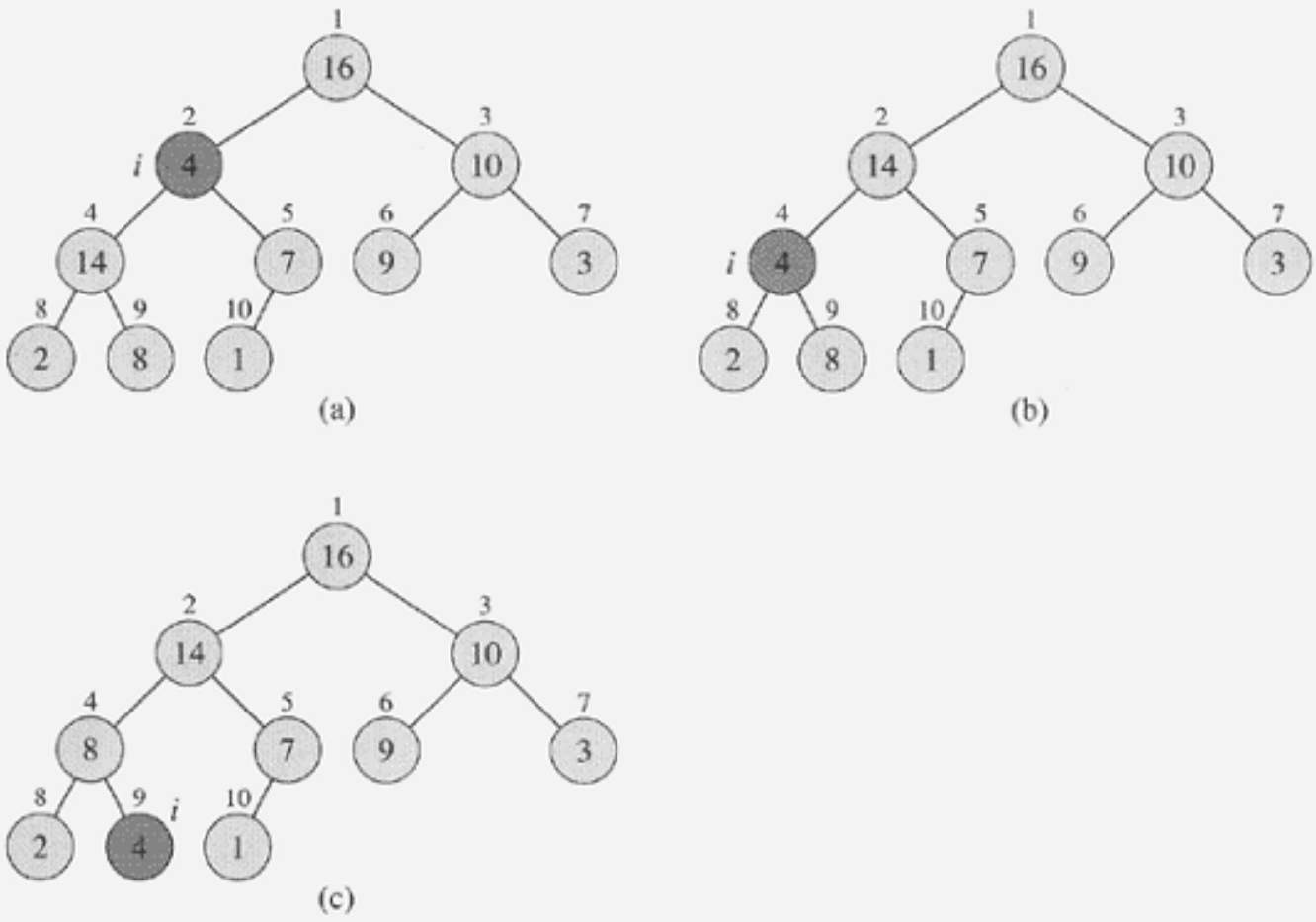


Figure 6.2 The action of MAX-HEAPIFY(A, 2), where *heap-size*$[A] = 10$. **(a)** The initial configuration, with $A[2]$ at node $i = 2$ violating the max-heap property since it is not larger than both children. The max-heap property is restored for node 2 in **(b)** by exchanging $A[2]$ with $A[4]$, which destroys the max-heap property for node 4. The recursive call MAX-HEAPIFY(A, 4) now has $i = 4$. After swapping $A[4]$ with $A[9]$, as shown in **(c)**, node 4 is fixed up, and the recursive call MAX-HEAPIFY(A, 9) yields no further change to the data structure.

Si la valeur du noeud est 15 à la place 4.
Combien de comparaisons qu'on fait ?
$T[l] > T[i]$
$T[r] > T[max]$

Le meilleur des cas ?

Construction d'un tas

On réorganise le tableau sous forme d'un TAS max

Le père de la dernière feuille est à l'indice $\lfloor n/2 \rfloor$.
Par conséquent on commence à organiser le tas par cet indice.

CONSTRUCTION(T)
$n = T.taille$
for $i = \lfloor n/2 \rfloor$ downto 1 **do**
  ENTASSER(T,i,n)
end for

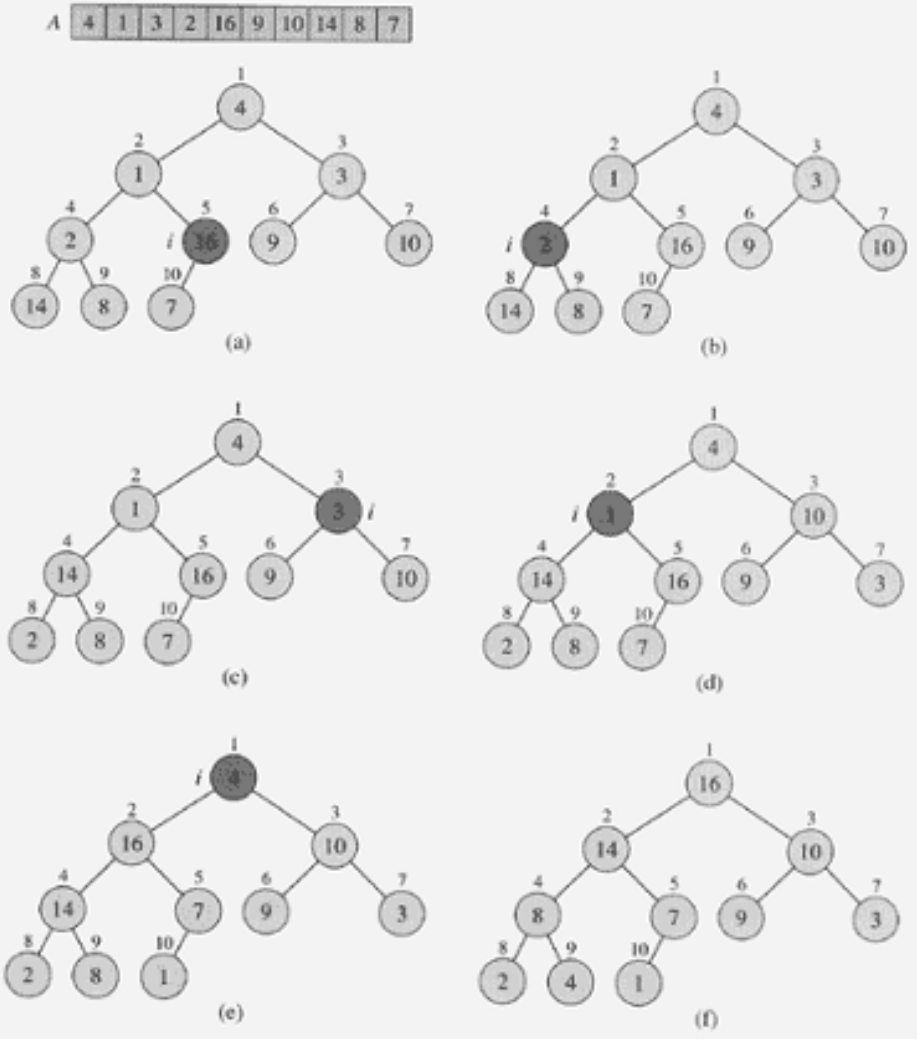


Figure 6.3 The operation of BUILD-MAX-HEAP, showing the data structure before the call to MAX-HEAPIFY in line 3 of BUILD-MAX-HEAP. **(a)** A 10-element input array A and the binary tree it represents. The figure shows that the loop index i refers to node 5 before the call MAX-HEAPIFY(A, i). **(b)** The data structure that results. The loop index i for the next iteration refers to node 4. **(c)–(e)** Subsequent iterations of the **for** loop in BUILD-MAX-HEAP. Observe that whenever MAX-HEAPIFY is called on a node, the two subtrees of that node are both max-heaps. **(f)** The max-heap after BUILD-MAX-HEAP finishes.

Complexité (le pire des cas) pour la construction d'un TAS

- On fait $i = \lfloor n/2 \rfloor \cdots 1$ itérations
- La complexité ENTASSER(T,i,n) dépend de la profondeur du noeud $i, \Theta(h(i))$.
- On peut majorer $\Theta(h(i))$ par $O(logn)$. Par conséquent la complexité dans le pire des cas est $O(nlogn)$.
- *On peut montrer que la complexité asymptotique est* $\Theta(n)$

Le père de la dernière feuille est à la profondeur $h-1$ (l'indice $n/2$) et il peut y avoir 2^{h-1}noeuds à ce niveau.
Au niveau $(h-1)$, il y aura 2^{h-1} fois $\Theta(h-(h-1))$
Au niveau $(h-2)$, il y aura 2^{h-2} fois $\Theta(h-(h-2))$
....
Au niveau 0 il y aura 1 fois $\Theta(h)$

$$\sum_{l=0}^{h-1} 2^l(h-l) = 2^h \sum_{l=0}^{h-1} \frac{(h-l)}{2^{h-l}} = 2^h \sum_{i=1}^{h} \frac{i}{2^i} \quad \text{changement de variable}$$

$$< 2^h.2 \quad \text{car} \sum_{i=0}^{\infty} \frac{i}{2^i} = \frac{1/2}{(1-1/2)^2} = 2$$

Comme $2^h = \Theta(n)$, la complexité est $\Theta(n)$.

Tri par Tas

TRI-TAS(T)
CONSTRUCTION(T)
$n = T.taille$
for $i = n$ downto 2 **do**
  // T est un TAS dans les indices de 1 à i
  // On échange les valeurs des indices 1 (MAX) et
  //i (le dernier élément du tas)
  $T[1] \leftrightarrow T[i]$
  // On met le ième élément à sa place
  //On réorganise le TAS de 1 à i-1
  ENTASSER(T,1,i-1)
end for

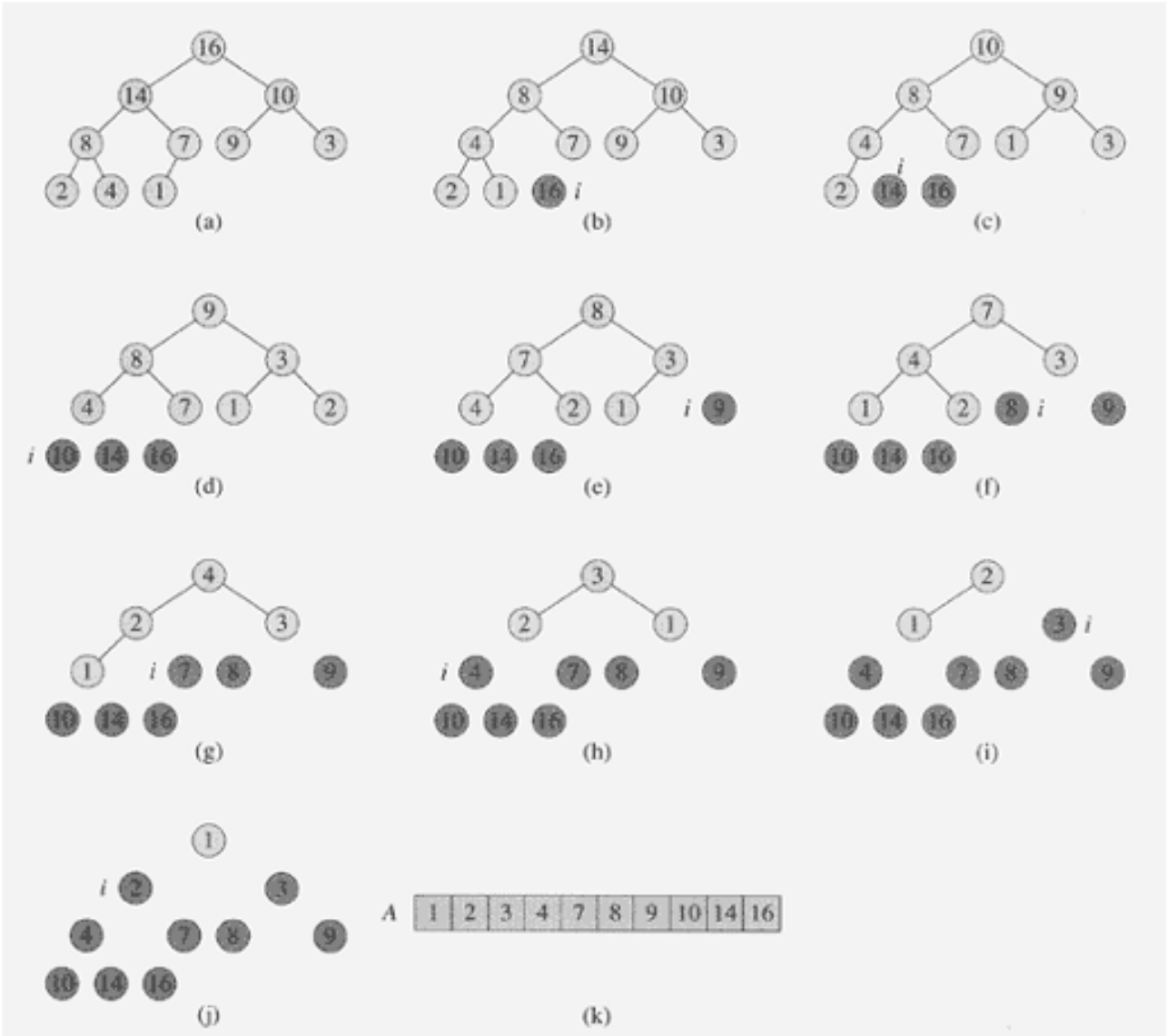


Figure 6.4 The operation of HEAPSORT. **(a)** The max-heap data structure just after it has been built by BUILD-MAX-HEAP. **(b)–(j)** The max-heap just after each call of MAX-HEAPIFY in line 5. The value of i at that time is shown. Only lightly shaded nodes remain in the heap. **(k)** The resulting sorted array A.

Complexité(pire des cas) Tri par TAS

- Complexité dans le pire des cas CONSTRUCTION(T,n) est $\Theta(n)$.
- Complexité dans le pire des cas de la boucle for est $\Theta(n.logn)$:
  Niveau h, il y a 2^h noeuds et complexité ENTASSER est $\Theta(h)$
  ....
  Niveau 1, il y a 2 noeuds et complexité ENTASSER est $\Theta(1)$

$$2.1 + 2^2.2 + 2^3.3 + \cdots 2^h.h = \sum_{l=1}^{h} 2^l l = 2\sum_{l=1}^{h} 2^{l-1} l$$

$S(x) = \sum_{l=0}^{h} x^l = \frac{x^{h+1}-1}{x-1}$; $\frac{dS(x)}{dx} = \sum_{l=1}^{h} x^{l-1} l$

$\frac{dS(x)}{dx} = \frac{(h+1)x^h(x-1)-x^{h+1}+1}{(x-1)^2}$

Nous devons évaluer $x\frac{dS(x)}{dx}$ pour x=2, après simplification : :

$$2^{h+1}(h-1) + 2; \Theta(nlogn)$$

- Complexité dans le pire des cas pour TRI par TAS :

$$max(\Theta(n), \Theta(n.logn)) = \Theta(n.logn)$$

Suppression Max et Réorganiser TAS

SUPP-MAX(T, n)
//T est un TAS MAX de taille n
// La taille du TAS serait n-1 après la suppression de max
if $n < 1$ **then**
  return ERREUR
else
  $max = T[1]$
  $T[1] = T[n]$
  $n = n - 1$
  ENTASSER(T,1, n)
  return max
end if

Complexité $\Theta(logn)$.

AJOUTER un élément et réorganiser le TAS

$T[i], 1 \leq i \leq n$ est un TAS MAX. On ajouter l'élément *cle* à l'indice $n+1$

AJOUTER(T, n, cle)
// on augmente la taille du TAS
$n = n + 1$
$T[n] = cle$
$i = n$
while $i > 1$ AND $T[pere(i)] < T[i]$ **do**
  $T[i] \leftrightarrow T[pere(i)]$
  $i = pere(i)$
end while

Complexité $\Theta(logn)$.

TAS min = File à priorité "Priority queue"
Utile pour des algorithmes où il faut ordonancer des tâches