8 Tri par tas

8.4 Heapisation

Opération heapify (A) met les éléments du tableau $A[1..n]$ dans l'ordre de tas. Triviale ?

$$H \leftarrow \emptyset\,;\ \textbf{for}\ i \leftarrow 1,\dots,n\ \textbf{do}\ \textsc{insert}(A[i],H,i)\,;\ A \leftarrow H$$

$\Rightarrow$ prend $\Theta(n\log n)$ au pire

Meilleure solution :

```
HEAPIFY(A) // tableau arbitraire A[1..n]
for i ← ⌈n/2⌉, ..., 1 do SINK(A[i], i, A, n)
```

Théorème 8.1. HEAPIFY *met les éléments dans l'ordre de tas en temps* $O(n)$.

Démonstration. SINK prend $O(h)$ temps où h est la hauteur du nœud qui correspond à l'indice i dans la représentation arborescente du tas. Il y a $\leq \lceil n/2^h \rceil$ nœuds internes avec hauteur h. Donc le temps de calcul est borné par $T(n) \leq \sum_{h=1}^{1+\lfloor \lg n \rfloor} \lceil \frac{n}{2^h} \rceil \times O(h) = O(n) \cdot \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \right) = O(n)$. ■

8.5 Tri par tas

W(fr)

On peut utiliser une file de priorité pour tri : mettre tous éléments dans la file (heapify), et retirer l'un après l'autre dans l'ordre croissant. Avec un tas binaire, on peut faire le tri en place. Après heapify, on maintient l'ordre de tas dans le préfixe $A[1..i]$ en une boucle $i \leftarrow n, n-1, \dots 2$. Le suffixe $A[i..n]$ est toujours trié en ordre décroissant. À chaque itération, après avoir échangé $A[1] \leftrightarrow A[i]$, on rétablit l'ordre de tas en $A[1..i-1]$ pour la prochaine itération.

```
   HEAPSORT(A[1..n])                                   // tableau non-trié
H1 heapify(A)
H2 for i ← n, ... 2 do
H3     v ← A[i] ; A[i] ← A[1]                          // échange A[i] ↔ A[1]
H4     SINK(v, 1, A, i − 1)                            // rétablissement de l'ordre de tas en A[1..i − 1]
```

Ordre des éléments. On finira avec l'ordre décroissant — pour l'ordre croissant :

- ⋆ renverser le résultat en place : $i \leftarrow 1; j \leftarrow n;$ **while**$(i < j)\{A[i] \leftrightarrow A[j]; i \leftarrow i+1; j \leftarrow j-1\}$
- ⋆ ou implanter l'ordre de max-tas dans le code

Temps de calcul et mémoire. Tri par tas finit en $O(n\log n)$ temps. C'est un **tri en place** parce qu'il utilise seulement $O(1)$ espace additionnelle (quelques variables à part du tableau à trier).