L'exercice 1 fait parti du cours, vous devez donc savoir répondre aux questions de cet exercice. Même si le sujet utilise principalement des tas-max, vous devez être capables d'implémenter les fonctions correspondantes pour les tas-min.
Pour représenter les tas-max en OCaml, on utilise le type ci-contre. Le champ `n` contient le nombre de noeuds dans l'arbre. Le champ `elem` est un tableau dont la taille `m` $\geqslant$ `n` est fixée au moment de la création du tas-max.

```
type 'a tas_max = {
    mutable n: int;
    elem: 'a array;
};;
```

```
type 'a file_prio =
  ('a * int) tas_max;;
```

Une file de priorité max est représentée par une variable de type `'a file_prio`. Les valeurs de la file sont de type `'a` et les priorités de type `int`. On pourra utiliser les fonctions « `fst: 'a * 'b -> 'a` » et « `snd: 'a * 'b -> 'b` » qui renvoient respectivement le premier et le deuxième élément d'un couple.

# Exercice 1. Questions de cours

### Opérations élémentaires sur les files de priorité

1. Écrire une fonction « `echange: 'a tas_max -> int -> int -> unit` » qui prend en entrée un tas-max ainsi que deux entiers $i, j$, et échange les éléments d'indices $i$ et $j$ dans le tableau `elem`.
2. Écrire une fonction « `make: int -> 'a -> 'a tas_max` » qui prend en entrée `m` ainsi qu'un élément « `e: 'a` », et renvoie un tas-max vide. L'argument `e` sera utilisé pour initialiser les cases du tableau (qui est de taille `m`).
3. Écrire une fonction « `perco_haut: 'a file_prio -> int -> unit` » qui prend en entrée une file de priorité `fp` ainsi qu'un indice `i0` (dont la priorité vient d'augmenter), et effectue une percolation vers le haut sur l'élément `fp.elem.(i0)`.
4. Même question pour une percolation vers le bas : « `perco_bas: 'a file_prio -> int -> unit` ».
5. Écrire une fonction « `ajout: 'a file_prio -> 'a -> int -> unit` » qui ajoute un élément à une file de priorité. Dans le cas où toutes les cases du tableau sont utilisées, on déclenchera une erreur (on pourrait également augmenter la taille du tableau, mais ce n'est pas demandé ici).
6. Écrire une fonction « `extrait_max: 'a file_prio -> ('a * int)` » qui extrait l'élément de priorité maximale dans une file de priorité.

### Tri par tas

7. À l'aide des fonctions précédentes, écrire un algorithme de tri de type « `int array -> int array` ». Cet algorithme s'appelle le ***tri par tas***.

### Complexité des opérations

8. Donner les complexités des fonctions demandées dans les questions précédentes.

### Piles LIFO et files FIFO

9. Expliquer comment implémenter une pile avec priorité LIFO (last in, first out).
10. Expliquer comment implémenter une file avec priorité FIFO (first in, first out).

# Exercice 2. Construction d'un tas en temps linéaire

Dans cet exercice, on considère un tableau `tab` contenant $n$ entiers et on souhaite construire un tas-max contenant tous les éléments de `tab`.

1. Expliquer comment construire ce tas-max en temps $\mathcal{O}(n \log n)$.

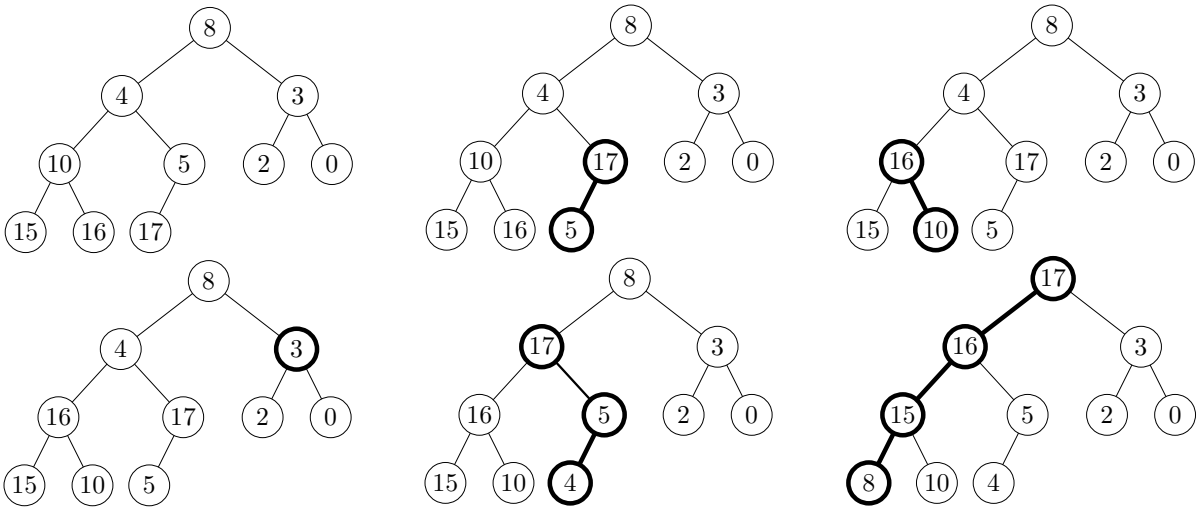
En fait il est possible de construire le tas-max en temps $\Theta(n)$.

2. Soit $A$ un arbre binaire presque-complet. On suppose que les sous-arbres droit et gauche sont des tas-max. En revanche, on ne suppose pas que la racine est supérieure à ses deux fils. Expliquer comment obtenir un tas-max en temps $\mathcal{O}(\ell)$ où $\ell$ est la hauteur de l'arbre.

Dans la suite, on interprète `tab` comme un arbre binaire presque-complet à l'aide de la numérotation de Sosa. On appelle sous-arbre d'indice $i$ le sous-arbre dont la racine est étiquetée par l'élément d'indice $i$ du tableau. On note $F \subseteq [\![0, n-1]\!]$ l'ensemble des indices des feuilles de l'arbre.

3. Que vaut $F$ ?

Afin de construire un tas-max à partir de `tab`, on applique la transformation de la question 2 au sous-arbre d'indice $i$ pour chaque $i \in [\![0, n-1]\!] \setminus F$. Les indices seront parcourus dans l'ordre décroissant afin de garantir que l'hypothèse de la question 2 est vérifiée :




4. Écrire cet algorithme en OCaml.
   **Indication.** Pour pouvoir réutiliser la fonction `perco_bas` de l'exercice 1, on pourra d'abord créer une file de priorité `fp` dont les valeurs sont arbitraires et dont les priorités sont les éléments de `tab`. On pourra également utiliser la fonction « `Array.map: ('a -> 'b) -> 'a array -> 'b array` » qui prend en entrée une fonction `f` ainsi qu'un tableau `[|`$e_0$`;` $e_1$`; ...;` $e_{n-1}$`|]` et renvoie le tableau `[|f` $e_0$`; f` $e_1$`; ...; f` $e_{n-1}$`|]`.
5. Obtient-on nécessairement le même arbre lorsqu'on applique l'algorithme de la question 1 et l'algorithme de la question 4 ?
6. Montrer que l'algorithme de la question 4 s'exécute en temps $\Theta(n)$.

# Exercice 3. Quelques questions de programmation

### Vérification de la propriété de tas-max.

1. Écrire une fonction de type « `'a tas_max -> bool` » qui teste si l'objet en entrée représente un tas-max.

**Représentation alternative des tas-max.** On introduit un autre type pour représenter les tas-max :

```
type 'a arbre = | Vide | N of 'a * 'a arbre * 'a arbre;;
```

2. Écrire une fonction de type « `'a tas_max -> 'a arbre` » qui effectue le conversion.
   **Indication (essayez de résoudre la question sans lire ce qui suit).** On pourra écrire une fonction intermédiaire qui prend en entrée un entier `i` et renvoie le sous-arbre dont la racine est étiquetée par l'élément d'indice `i` du tableau `elem`.
3. Écrire une fonction de type « `'a arbre -> 'a tas_max` » qui effectue le conversion inverse.
   **Indication (essayez de résoudre la question sans lire ce qui suit).** On pourra écrire une fonction intermédiaire qui prend en entrée un entier `i` ainsi qu'un arbre `arb` et remplit le tableau `elem` en considérant que la racine de `arb` est d'indice `i` dans `elem`.

**Fusion de listes triées** Étant données `k` listes d'entiers triées $\texttt{L}_1, \ldots, \texttt{L}_\texttt{k}$, on souhaite fusionner ces listes ; c'est à dire obtenir une liste qui soit triée et qui contienne tous les éléments présents dans les $\texttt{L}_\texttt{i}$. L'idée est de partir d'une liste `L` vide, de supprimer le plus petit élément `x` présent dans les $\texttt{L}_\texttt{i}$ et d'ajouter `x` à `L`. Il suffit alors de répéter cette procédure tant que les $\texttt{L}_\texttt{i}$ ne sont pas toutes vides.

Pour trouver le minimum, on va utiliser une file de priorité. À chaque instant, la file de priorité contiendra au plus `k` éléments, un pour chaque $\texttt{L}_\texttt{i}$ non vide. En choisissant correctement les valeurs des priorités, on peut obtenir en temps $\mathcal{O}(\texttt{log(k)})$ l'indice `i` tel que le premier élément de $\texttt{L}_\texttt{i}$ est le minimum recherché. Si le nombre total d'éléments est `n`, le temps d'exécution de cette procédure est en $\mathcal{O}(\texttt{n log(k)})$.

4. Grâce aux fonctions de l'exercice 1, implémenter l'algorithme décrit ci-dessus.
   **Indication (essayez de résoudre la question sans lire ce qui suit).** On suppose que les listes sont stockées dans un tableau. Pour chaque $\texttt{L}_\texttt{i}$ non vide, la file de priorité contient un élément dont la valeur est `i` et dont la priorité est l'opposée du premier élément de $\texttt{L}_\texttt{i}$.

# Exercice 4. Tas-max $d$-aires dynamiques

Soit $d \geqslant 1$ un entier. Un arbre $d$-aire est un arbre dont chaque noeud a au plus $d$ fils. Un tas-max $d$-aire est un arbre $d$-aire presque complet tel que l'étiquette de chaque noeud est supérieure aux étiquettes de ses fils.

Proposer un moyen de représenter les tas-max $d$-aires à l'aide d'un tableau et implémenter les opérations d'ajout et de suppression d'un noeud. Dans le cas où on essaye d'ajouter un élément alors que le tableau est plein, on doublera la taille du tableau avant de faire l'ajout.

**Indications (essayez de résoudre l'exercice sans lire ce qui suit).** On pourra utiliser une variante de la numérotation de Sosa : le tableau `tab` contient le parcours en largeur de l'arbre. Ainsi pour chaque indice `i` valide (faire un dessin pour se convaincre que les formules données ci-dessous sont correctes) :

- S'il existe, le père de `tab.(i)` est `tab.((i-1)/d)`.
- S'il existe, le fils numéro $\texttt{k} \in [\![1, \texttt{d}]\!]$ de `tab.(i)` est `tab.(d*i+k)`.