TP3 Caml : Tri par tas

1 Tri par sélection

1. Ecrire une fonction `min_liste : int list -> int * int list` qui renvoie le plus petit élément d'une liste et la liste de départ privée de ce plus petit élément. Cette fonction renvoie une erreur pour la liste vide.
Exemple :

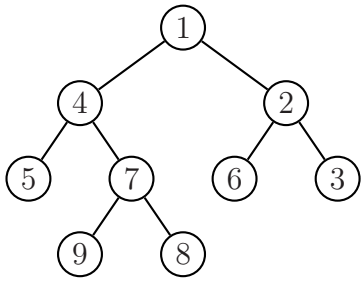
```
#min_list [2;4;6;8;1;3;7;5;9] ;;
- : int * int list = 1, [2; 4; 6; 8; 3; 5; 7; 9]
```

2. En déduire une fonction `tri_selection : int list -> int list` qui trie une liste d'entiers par ordre croissant.

3. Quelle est la complexité de cet algorithme de tri ?

2 Tri par tas

Pour améliorer l'algorithme précédent on propose de mettre en œuvre une structure de données facilitant la recherche du minimum d'un ensemble : un *tas*.
Un tas est un arbre binaire *ordonné* : tous les nœuds autre que la racine ont une valeur plus grande que leur père.
Exemple :



On utilisera le type suivant :

```
type Tas =
   Vide
  | Noeud of int * Tas * Tas ;;
```

4. Ecrire une fonction `min_tas : Tas -> int` qui renvoie le plus petit élément d'un tas ou une erreur si le tas est vide.

5. Ecrire une fonction `enleve_min : Tas -> Tas` qui retire le plus petit élément d'un tas et renvoie le nouveau tas.

6. Ecrire une fonction `ajouter : int -> Tas -> Tas` qui ajoute un élément à un tas. Pour cette fonction il faudra faire un choix : insérer dans le fils gauche ou le fils droit du nœud courant. Afin de garder un tas équilibré on propose de toujours insérer dans le fils gauche mais de permuter ensuite les deux fils.

7. Ecrire une fonction `ajouter_liste : int list -> Tas` qui construit un tas à partir d'une liste d'entiers par ajouts successifs à partir d'un tas vide.

8. Ecrire une fonction `vider : Tas -> int list` qui vide un tas dans une liste en rendant une liste triée par ordre croissant.

9. En déduire une fonction `tri_par_tas : int list -> int list` qui trie une liste d'entiers par ordre croissant.

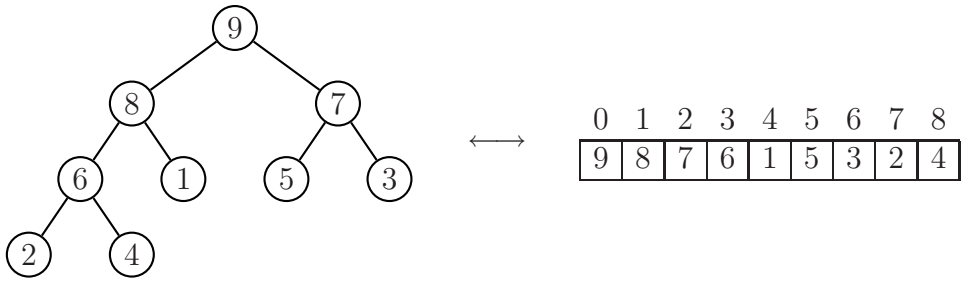
10. Quelle est la complexité de cet algorithme de tri ?

3 Tri par tas dans un tableau

Dans la pratique, le tri par tas est réalisé dans un tableau qui représente l'arbre binaire associé au tas. On peut ainsi trier *sur place* un tableau : on n'utilise pas d'autres cases mémoires que celles qui contiennent le tableau à trier.
Pour pouvoir être stocké dans un tableau, les tas étudiés ici sont des arbres *presque complets* : tous les niveaux sont remplis sauf éventuellement le dernier qui est rempli en partant de la gauche jusqu'à un certain point (*cf* exemple suivant).
Désormais le père d'un nœud (autre que la racine) sera **plus grand** que ses fils.
On utilisera le type suivant :

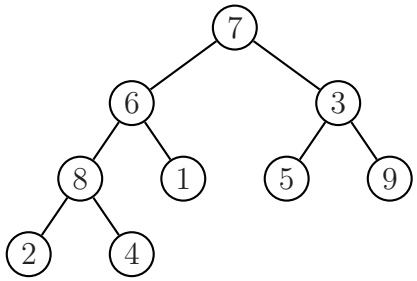
```
type Tas_vect = { tab : int vect ; mutable taille : int } ;;
```

Le champ `taille` représente le nombre de nœuds du tas : celui-ci peut être inférieur au nombre d'élément du tableau `tab`. La racine du tas est `tab.(0)`. Pour un nœud d'indice i donné, son fils gauche a l'indice $2i+1$ et son fils droit l'indice $2i+2$. Le tableau est ainsi rempli niveaux par niveaux, de gauche à droite.
Exemple: { `tab=[|9;8;7;6;1;5;3;2;4;0;0;0;0|] ; taille=9` }



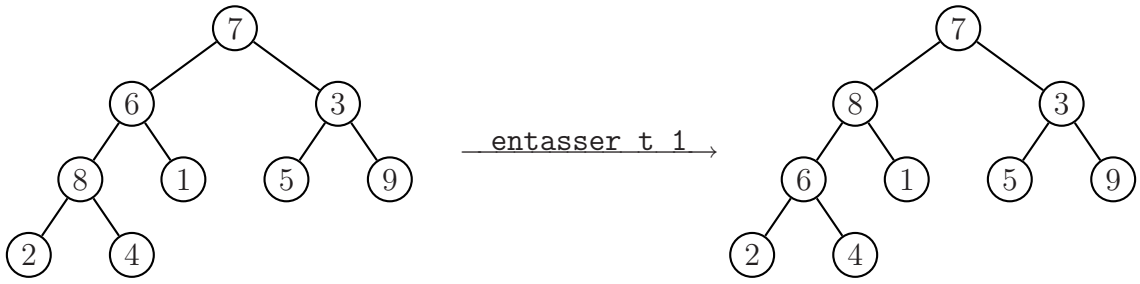
11. Ecrire une fonction `max_noeud : Tas_vect -> int -> int` qui prend un tas t et un indice i et renvoie l'indice du plus grand élément parmis le nœud d'indice i et ses (éventuels) fils, sans supposer que les nœuds du tas sont ordonnés.

Exemple :

si tab = 

, `max_noeud t 1` = 3

12. Ecrire une fonction `entasser : Tas_vect -> int -> unit` qui prends en argument un tas t non nécessairement ordonné et un indice i. On suppose que les arbres binaires enracinés aux niveaux des fils du nœud d'indice i sont ordonnés. Le rôle de la fonction `entasser` et de modifier le sous-arbre enraciné en i de façon à ce qu'il devienne un arbre binaire ordonné (en utilisant la fonction `max_noeud`).
Exemple :



13. En déduire une fonction `construire_tas : int vect -> Tas_vect` qui construit un tas à partir d'un tableau d'entiers en appelant la fonction `entasser` sur les indices du tableau en commençant par les feuilles.

14. Ecrire une fonction `retire_max : Tas_vect -> unit` qui retire l'élément maximum d'un tas en permutant cet élément avec la dernière valeur du tableau, en décrémentant le champ `taille` et mettant à jour la structure du tas.

15. En déduire une fonction `vide_tas : Tas_vect -> unit` qui vide un tas en appelant la fonction `retire_max` jusqu'à ce que le champ `taille` du tas soit 1.

16. En déduire une fonction `tri_par_tas_vect : int vect -> int vect` qui tri un tableau d'entiers par ordre croissant.