TP #19.2 — Tas

Exercice 1. *Implémentation d'un tas en OCaml*

On se donne le type `'a heap` suivant :

```
type 'a heap = { dat : 'a array ;
                 gt : ('a -> 'a -> bool) ;
                 mutable cnt : int }
```

d'un tas « max » relativement à une fonction de comparaison `gt` (c'est à dire que l'élément `t` à la racine du tas doit être tel que pour tout autre élément `e` du tas l'on a `gt t e` qui s'évalue à `true`).

1. Écrivez une fonction OCaml :

   ```
   make : 'a -> ('a -> 'a -> bool) -> int -> 'a heap
   ```

   telle que `make e gt n` crée un tas vide de capacité `n`, dont le tableau a été rempli avec les valeurs `e`.
2. Écrivez des fonctions OCaml :

   ```
   _sift_up : 'a heap -> int -> unit
   _sift_down : 'a heap -> int -> unit
   ```

   qui implémentent les algorithmes de « percolation » d'un tas. (Il peut être pratique de définir au préalable des fonctions auxiliaire de calcul des enfants & parents, ainsi qu'effectuant l'échange de deux éléments d'un tableau.)
3. Testez.
4. Écrivez des fonctions OCaml :

   ```
   push : 'a heap -> 'a -> unit
   pop  : 'a heap -> 'a
   ```

   qui implémentent les opérations correspondantes d'une file de priorité impérative à capacité fixe. N'oubliez pas de traiter les cas d'erreur associés.
5. Testez. Vérifiez notamment « en boîte noire » que ces fonctions implémentent correctement la fonctionnalité d'une file de priorité.
6. Écrivez une fonction OCaml :

   ```
   heap_sort : 'a array -> ('a -> 'a -> bool) -> unit
   ```

   telle que `heapsort a gt` trie *en place* (c'est à dire sans aucune recopie) son argument `a` pour l'ordre croissant donné par `gt`, en utilisant un tri par tas.
7. Testez, notamment avec des types et des ordres variés.

Exercice 2. *Implémentation du tri par tas en C*

1. Écrivez une fonction C de signature :
   ```
   void heap_sort(int *a, size_t n)
   ```
   qui trie son argument `a` en place pour l'ordre croissant sur les entiers.
   On ne demande pas ici d'implémenter *toutes* les fonctions d'une file de priorité (ou même d'un tas), mais seulement celles nécessaire au tri.
   Pour plus de variété, on demande cependant d'implémenter (au moins) des versions *non récursives* de chaque fonction.