Tas

Dans ce TD, on va s'intéresser aux tas, qui sont des arbres binaires que leur structure spécifique permet de représenter directement dans des tableaux.

1. **Représentation par un tableau**
   a. Dessiner les arbres binaires parfaits dont le parcours en largeur correspond à ces suites d'entiers :
      — 1, 5, 7, 6, 5, 8, 9, 12, 7, 5, 8, 9
      — 4, 5, 8, 6, 6, 9, 8, 7, 5, 11
   b. Parmi les arbres représentés ci-dessus, quels sont ceux qui sont des arbres tournois ?
   c. Soit `i` un indice dans un tableau représentant un arbre parfait. Quel est l'indice de son nœud parent ?
   d. Soit `i` un indice dans un tableau représentant un arbre parfait. Quels sont les indices de son fils gauche et de son fils droit ?
   e. En partant d'un tas vide, on ajoute dans l'ordre les valeurs suivantes : 5, 13, 4, 14, 1, 20. Dessiner le tas obtenu après ajout de ces valeurs, et donner le tableau qui le représente.
   f. On enlève maintenant à notre tas sa valeur minimale. Dessiner le tas obtenu après cette suppression, et donner le tableau qui le représente.
2. **Vérification** — Dans cet exercice et les exercices qui suivront, on utilisera le type C suivant :

```c
typedef struct {
   int taille;      /* taille courante */
   int arbre[MAX];  /* stockage des étiquettes */
} Tas;
```

   où `MAX` désigne une constante.
   Écrire une fonction `int estTas(int tab[], int taille)` qui vérifie qu'un tableau `tab` de taille `taille` représente un tas.
3. **Fils minimal** — Écrire une fonction `int Fils(Tas T, int indice)` qui renvoie `-1` si le nœud représenté par la case `indice` n'a pas de fils, et qui renvoie, sinon, l'indice du nœud fils contenant la plus petite étiquette.
4. **Change** — Écrire une fonction `void Change(Tas *T, int indice, int valeur)` qui remplace la valeur de la case `indice` par `valeur` et effectue différents échanges de case afin de respecter la structure de tas. On vérifiera que `indice` est inférieur à la taille du tas `T`.
5. **Ajout** — Écrire une fonction `int Ajout(Tas *T, int valeur)` qui ajoute un nœud de valeur `valeur` au tas `*T`, tout en maintenant la structure de tas de `*T`. La fonction renverra `1` si l'ajout est correct et `0` sinon.
6. **Extraire l'élément minimal** — Écrire une fonction `int ExtraireMin(Tas *T)` qui extrait la valeur minimale du tas `*T` et la renvoie, tout en maintenant la structure de tas de `*T`.
7. **Tri en tas** — Réécrire les fonctions précédentes avec la structure suivante :

```c
typedef struct {
   int taille;      /* taille courante */
   int *arbre;      /* stockage des étiquettes */
   int Max;         /* taille maximale du tas */
} Tas;
```

   En utilisant ces fonctions, écrire une fonction `void TriTas(int T[], int taille)` qui effectue un tri du tableau `T` en ordre **décroissant**.