Semaine 6 – Tri par tas

Dans ce TP, vous allez implémenter l'algorithme de tri par tas en utilisant une structure d'arbre implicite. Cet algorithme permet un tri beaucoup plus efficace que ceux implémentés lors des TP précédents.

1 Structure d'arbre implicite

Le tri par tas repose sur une structure appelée *tas-max* qui est un arbre binaire équilibré tassé à gauche tel que la valeur stockée dans un nœud est supperieure aux valeurs stockées dans ses enfants.

L'utilisation d'une structure d'arbre explicite comme lors des TP précédents est particulièrement inefficace notamment car l'algorithme nécessite de pouvoir accéder facilement au nœud le plus à droite de la couche la plus basse de l'arbre. À la place, une structure d'arbre implicite dans un tableau est utilisée.

Commencer par implémenter les fonctions outils très suivantes. Ces fonctions permettent d'écrire l'algorithme de tri plus naturellement comme si l'arbre était explicite.

1. Tout d'abord, trois fonctions retournant respectivement l'index du nœud parent, gauche et droit d'un nœud dont l'index est donné en paramètre.

   ```
   int parent(int i);

   int gauche(int i);

   int droit(int i);
   ```

2. Ensuite une fonction permetant l'affichage d'un tas de manière structurée afin de pouvoir vérifier visuellement pendant le tri que les propriétés du tas sont bien respéctées.

   ```
   void afficher_tas(int tab[], int n);
   ```

3. Et enfin, une fonction permettant d'échanger deux nœuds de l'arbre dont les indexes sont donnés en paramètres.

   ```
   void echange(int tab[], int a, int b);
   ```

2 Tri par tas

Le cœur de l'algorithme de tri par tas est dans la procédure de tamisage qui est utilisée afin de transformer le tableau de données initiales en un tas puis tout au long de la procédure de tri afin de maintenir cette structure au fur et à mesure que les éléments sont retirés.

L'algorithme est implémenté dans trois fonctions :

1. Commencez par implémenter la procédure de tamisage qui fait décendre dans l'arbre la racine jusqu'à ce qu'elle soit à ça position. *Attention : L'arbre obtenu ne sera un tas que si la racine est le seul nœud à ne pas respecter les propriétés de tas au moment de l'appel.*

   ```
   void tamiser(int tab[], int deb, int fin);
   ```

2. Écrivez maintenant la fonction permettant de transformer un tableau d'entiers en une structure de tas implicite par tamisages successifs.

   ```
   void creer_tas(int tab[], int n);
   ```

3. Et enfin, écrivez la fonction de tri par tas à l'aide des fonction précédentes. Cette fonction commence par initialiser le tas, puis tant que celui-ci n'est pas vide, retire la racine qui est le plus grand élément restant et restore la propriété de tas.

   ```
   void tri_par_tas(int tab[], int n);
   ```

N'oubliez pas de tester toutes vos fonctions. La fonction d'affichage du tas implémentée dans le premier exercice est ici particulièrement utile lors de l'écriture du tri afin de vérifier que la structure de tas est maintenue correctement.