TP N°4 : Les TRI

Les tris font partie des familles d'algorithmes les plus classiquement étudiées, car ils comptent parmi les modules indispensables au bon déroulement d'algorithmes plus évolués. Le principe général d'un algorithme de tri est d'ordonner (par ordre croissant par exemple) les objets d'une collection de données (valeurs), en fonction d'un critère de comparaison (clé – pour nous, valeurs et clés sont ici confondues : ce sont les éléments d'un tableau d'entiers). On réalise en général un tri « sur place »: les valeurs triées sont rangées dans le même tableau que les valeurs initiales (qui devient donc un paramètre d'entrée-sortie).
Il existe une multitude d'algorithmes permettant de trier des données, dans ce TP, l'étude porte sur les méthodes ci-dessous.

- Les objectifs de ce TP sont multiples :
  - Mettre en pratique et tester quelques algorithmes de tri ;
  - Étudier les algorithmes de tri et calculer leur complexité ;
  - Confronter complexité théorique et évaluation du coût d'exécution ;
  - Afin d'étudier le coût réel des algorithmes, vous les testerez sur des tableaux d'entiers de taille n croissante remplis aléatoirement. Pour qu'une mesure de temps soit fiable, il faut la faire plusieurs fois (par exemple 10 fois) pour une taille de tableau donnée. C'est à vous de choisir les valeurs de n qui vous semblent pertinentes. Pour chaque tri étudié, vous tracerez un graphe représentant le temps de calcul en fonction de n ;
  - Comparer les différentes méthodes d'un point de vue théorique et expérimental.
- Remettre un rapport détaillé de l'étude des différentes méthodes de tri ainsi que le code source et une conclusion.
- Le travail peut être fait en binôme.
- Le travail doit être remis au plus tard dans deux semaines.

1. Tri à bulles

Principe : Parcourir le tableau T de taille N du dernier au premier élément (presque . . .), avec un indice i. A chaque étape, la partie du tableau située à droite de i est considérée comme triée. On parcourt alors la partie de gauche (partie non triée) avec un indice j. Pour chaque j, si T[j − 1] > T[j], on les permute.

a. Ecrire le programme C de la procédure TriBulle suivante :

Procédure du TriBulle (E/S: un tableau T[n] d'entiers ; E/n ;entier)
Début
  Changement = vrai ; (variable booléenne)

```
   Tant que (Changement=vrai) faire
        Changement ← faux ;
        pour i←1 à n-1 faire
             si (T[i] > T[i+1]) alors
                  Permuter(T[i], T[i+1]) ;
                  Changement ←VRAI;
             Fsi ;
        Fait;
    Fait;
Fin;
```

b. Après le i^{ème} parcours du tableau, tous les i derniers éléments sont à leurs places définitives. Donc à chaque parcours de tableau, le parcours pourra s'arrêter un indice avant le précédent. L'algorithme devient :

```
Procédure du TriBulleOpt (E/S: un tableau T[n] d'entiers ; E/n ;entier)
Début  m←n-1 ;
   Changement = vrai ; (variable booléenne)
   Tant que (Changement=vrai) faire
        Changement ← faux ;
        pour i←1 à m faire
             si (T[i] > T[i+1]) alors
                  Permuter(T[i], T[i+1]) ;
                  Changement ←VRAI;
             Fsi ;
        Fait;
        m←m-1;
    Fait;
Fin;
```

Ecrire le programme C de TriBulleOpt puis comparer les deux algorithmes TriBulle et TriBulleOpt au meilleur et pire cas.

2. Tri Gnome

Principe : Dans le tri gnome, on commence par le début du tableau, on compare deux éléments consécutifs (i, i+1) : s'ils sont dans l'ordre on se déplace d'un cran vers la fin du tableau (incrémente) ou on s'arrête si la fin est atteinte ; sinon, on les permute et on se déplace d'un cran vers le début du tableau (décrémente) ou si on est au début du tableau alors on se déplace d'un cran vers la fin (incrémente). (Ex4 TD4)
Ecrire le programme C et donner sa complexité théorique au meilleur et pire cas.

3. Tri par distribution

On utilise un tri par distribution (appelé aussi tri par base) pour trier des entiers selon leur chiffre le moins significatif (chiffre des unités), puis pour trier la liste obtenue selon le chiffre des dizaines puis selon le chiffre des centaines ...ect. La liste des entiers 141, 232, 045, 112, 143 va être triée selon le chiffre des unités, on obtient la liste 141, 232, 112, 143, 045 qui à son tour va être triée selon le chiffre des dizaines, on obtient la liste 112, 232, 141, 143, 045 puis va être triée selon le chiffre des centaines et on obtient la liste des entiers triée selon l'ordre croissant 045, 112, 141, 143, 232.

Soit un tableau T contenant n entiers entre 0 et $10^k - 1$, où k est une constante entière. On veut appliquer un tri par base, à ces entiers.

a. Ecrire la fonction clé(E/ x, i : entier) : entier ; qui retourne soit le chiffre des unités, soit le chiffre des dizaines, soit le chiffre des centaines …

Exemple : clé (143, 0)=3, clé (143, 1)=4, clé (143, 2)=1

b. Ecrire la fonction TriAux(T, n, i) qui réordonne les éléments de T tels que : clé(T[1], i) ≤ clé(T[2], i) ≤ . . . ≤ clé(T[n],i). TriAux doit s'exécuter en un temps linéaire en fonction de la taille n du tableau.

c. En utilisant la procédure TriAux, écrire la fonction TriBase(T, n, k) du tri par base du tableau T.

d. Ecrire le programme C en utilisant les fonctions définies précédemment.

4. Tri rapide

Le tri rapide est fondé sur une approche "diviser pour régner" que l'on peut décomposer en 3 étapes. On considère que nous avons un tableau Tab de taille n. On notera un "sous-tableau" de Tab, Tab[p…r], p étant l'indice du début du sous-tableau et r l'indice de la fin du sous tableau.

Diviser : le sous-tableau Tab[p…r] est partitionné (c-à-d réarrangé) en 2 sous-tableaux non vides A[p..q] et A[q+1..r] de telle sorte que chaque élément du tableau A[p..q] soit inférieur ou égal à chaque élément de A[q+1…r]. L'indice q est calculé pendant la procédure de partitionnement.

- ✓ Régner : Les 2 sous-tableaux A[p..q] et A[q+1..r] sont triés par des appels récursifs à la méthode principale de tri-rapide.
- ✓ Combiner : le tri rapide effectue le tri sur place. Cela implique qu'il n'y a aucun travail supplémentaire pour les fusionner : le sous-tableau Tab[p..r] tout entier est maintenant trié.

Le tri rapide va comporter 2 fonctions triRapide et partitionner qui renvoie un entier. L'entier renvoyé par partitionner est l'indice que l'on recherche dans l'étape

"Diviser". L'appel initial sur le tableau tab sera triRapide(tab, 0, n-1). Voici les algorithmes :

```
Procedure triRapide(E/tab : tableau[n] entiers ; p, r :entier)
Var q : entier;
début
   si (p < r) alors
      q ← partitionner(tab, p, r);
      triRapide(tab, p, q);
      triRapide(tab, q+1, r);
   fsi ;
fin ;
fonction partitionner(E/tab : tableau[n] entiers ; d, f :entier) : entier
var  eltPivot : entier ;  // on considère que tab est un tableau d'entiers
       i, j, x : entier ;
début
  eltPivot ← tab[p];
  i← d ; j← f ;
  Répéter
     Tant que (j≤d) et (tab[j]≤ eltPivot) faire  j← j+1 ;  fait ;
     Tant que (i≥f) et (tab[i]> eltPivot)  faire  i← i-1 ;    fait ;
     si (j<i) alors x←tab[j] ; tab[j]← tab[i] ;  tab[i]←x; j←j+1; i←i-1; fsi ;
 jusqu'à (j>i)
 retourner (i) ;
Fin ;
```

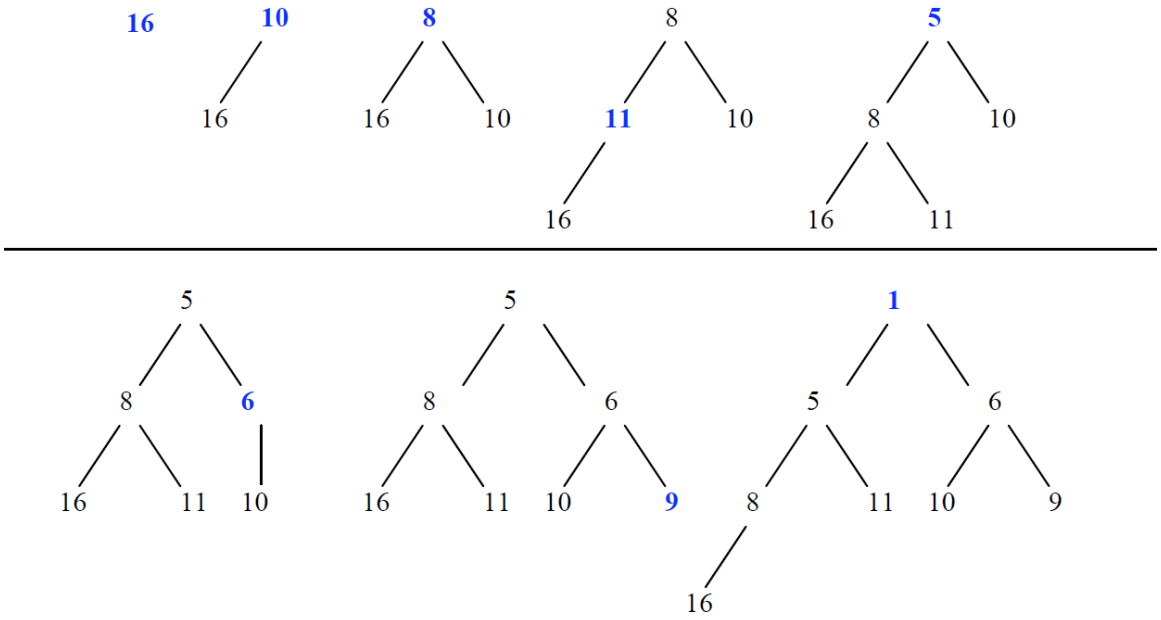
a. Ecrire le programme C du tri rapide. Donner et résoudre l'équation de récurrence du tri rapide.

5. Tri par tas

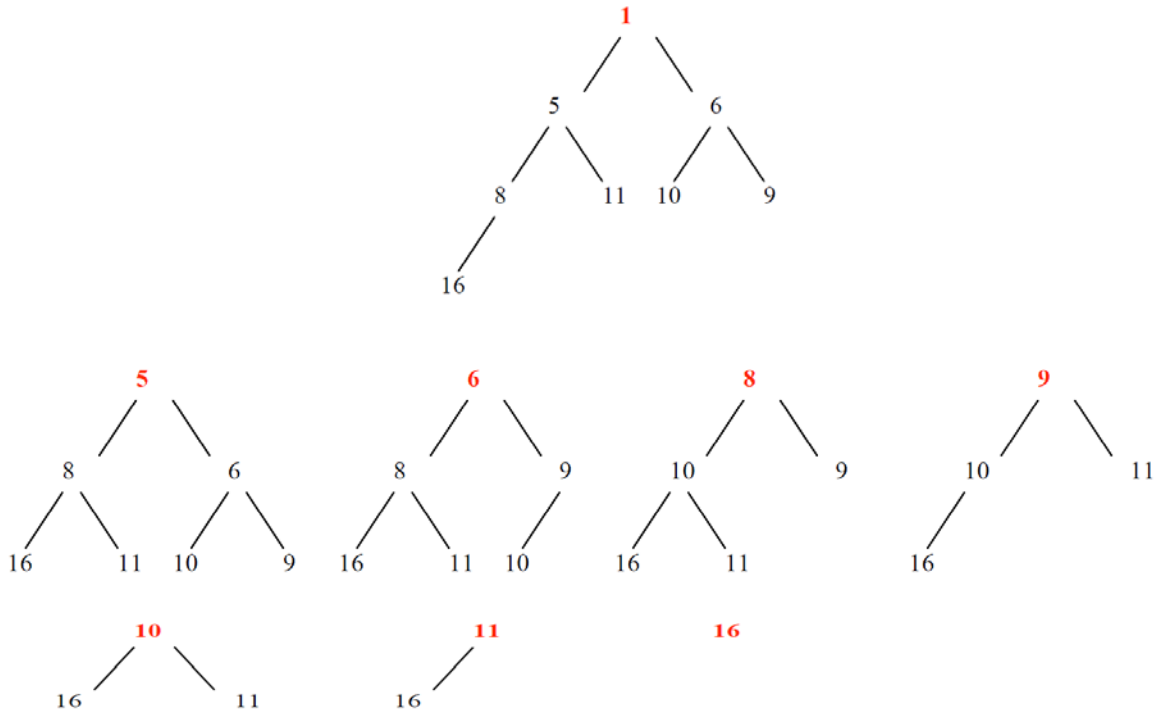
Le tri par tas se base sur une structure de données particulière : le tas. Il s'agit d'une représentation d'un arbre binaire sous forme de tableau. L'arbre est presque complet : il est rempli à tous les niveaux, sauf potentiellement le dernier. Le parcours de l'arbre se fait par un calcul d'indice sur le tableau. Pour un nœud d'indice i, on a le père à l'indice $\lfloor i/2 \rfloor$, le fils gauche à l'indice 2i et le fils droit à l'indice 2i+1.

On va appliquer les procédures d'insertion et de suppression de la racine d'un tas à l'exemple de façon à, dans un premier temps, construire un tas puis en supprimant le minimum atteindre la solution où tous les éléments sont dans l'ordre croissant. Nous allons d'abord construire le tas en insérant dans ce tas les

éléments de l'exemple les uns après les autres. Voici la représentation des différentes étapes pour la liste 16-10-8-11-5-6-9-1. A chaque étape, l'élément ajouté est indiqué en bleu.



En supprimant progressivement min par min, avec la procédure supprimer_min, on passe par les tas suivants et on en tire la liste ordonnée.



Ecrire le programme C du tri par tas puis évaluer la complexité de l'algorithme.