Corrigé E.D. Algorithmes et Structures de Données n° 4

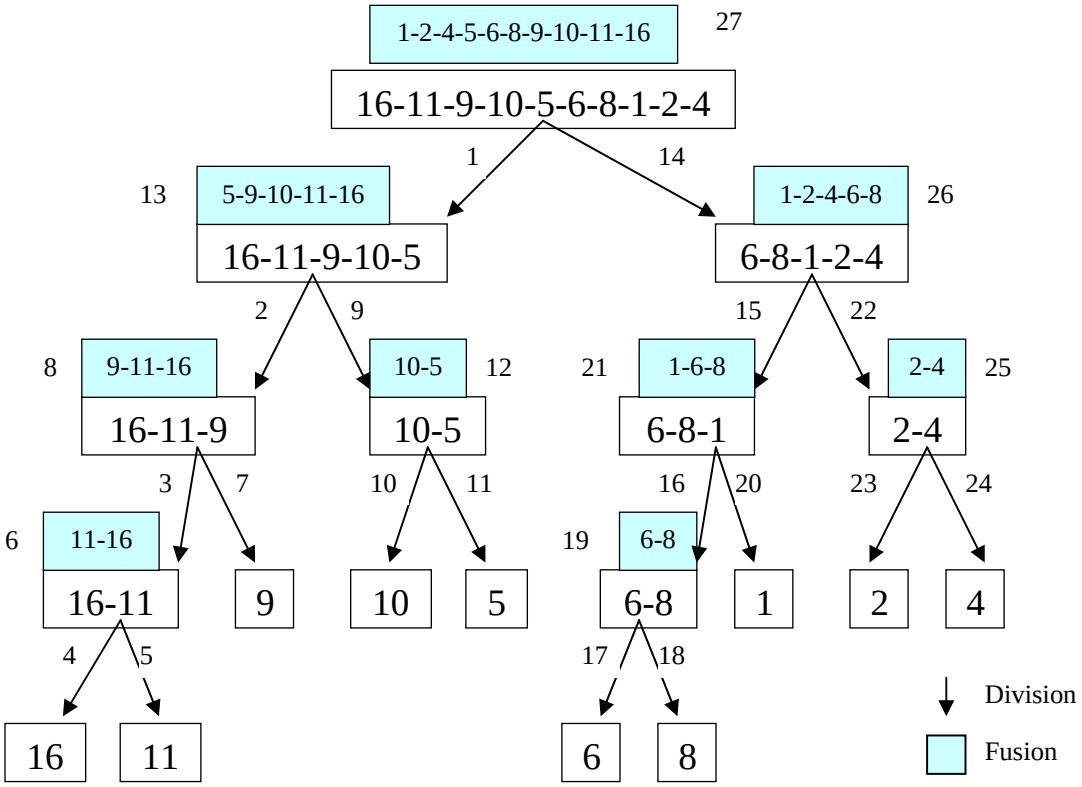
Thème : Les Tris

Exercice IV.1 Le Tri - fusion

Question 1 Appliquer l'algorithme du tri fusion à la suite de nombres suivante :

16 - 11 - 9 - 10 - 5 - 6 - 8 - 1 - 2 - 4

Dans l'illustration ci-dessous, les numéros de 1 à 27 indiquent l'ordre chronologique des opérations.



Question 2 Evaluer le nombre d'opérations nécessaires à ce tri (complexité).

Soit t(n) le nombre d'opérations du tri-fusion appliqué à un tableau de n éléments. Cette quantité t(n) peut être décomposée en :

1. la division comporte 5 opérations : g:=i;d:=j;m:=(i+j)/2; (cf. page 4.16)
2. on appelle tri-fusion pour le sous-tableau gauche qui comporte n/2 éléments, ce qui engendre un nombre d'opérations égal à t(n/2)
3. on appelle tri-fusion pour le sous-tableau droit qui comporte lui aussi n/2 éléments => t(n/2) opérations

4. on appelle fusionner que l'on sait être en O(n), on peut donc en déduire que son nombre d'opérations est proportionnel à n, nous noterons ce nombre cn, où c est une constante.

On obtient donc **t(n) = 5 + 2*t(n/2) + cn**

On souhaite arriver à des tableaux à un seul élément, soit h le nombre de divisions (et donc de fusions) nécessaires pour y arriver. On divise par 2 à chaque fois le nombre d'éléments de chacun des tableaux, nous arriverons donc à des tableaux de 1 élément lorsque n sera de l'ordre de 2^h.
$n = 2^h <=> h = \log_2 n$

Il s'agit là d'un résultat général : la hauteur d'un arbre binaire parfait de n éléments est $\log_2 n$.

Calculons t(n) :
t(n) = 5 + 2*t(n/2) + cn *(on va appliquer la formule à t(n/2))*
= 5 + 2*(5 + 2*t(n/2^2) + cn/2) + cn
= (1 + 2)*5 + 2^2*t(n/2^2) + 2*cn *(et maintenant à t(n/2^2)) etc…*

On peut montrer par récurrence la formule à l'étape i :
t(n) = (1 + 2 + ... + $2^{(i-1)}$)*5 + 2^i*t(n/2^i) + i*cn

Ce qui nous donne à l'étape h
t(n) = (1 + 2 + ... + $2^{(h-1)}$)*5 + 2^h*t(n/2^h) + h*cn

Or, $n = 2^h$, donc t(n) = (1 + 2 + ... + $2^{(h-1)}$)*5 + nt(1) + $\log_2 n$ *cn

D'autre part, (1 + 2 + ... + $2^{(h-1)}$) représente la somme des termes d'une suite géométrique de raison 2 et vaut : (1-2^h)/(1-2) =2^h – 1 = (n-1)

Finalement, t(n) = 5(n-1) +t(1)n +cn$\log_2 n$. L'algorithme est donc en O(n$\log_2 n$) (t(1) est une constante)

Question 3 Écrire la procédure **`void( fusionner (C_Tab t, Indice g, Indice d)`** définie dans le cours.

```
    static void  fusionner  (int[] t, int g, int d){
// on veut fusionner les parties t[g..m] et t[m+1..d] du tableau t
      // avec m : milieu de l'intervalle
      // ces deux parties sont chacune triee dans ordre croissant
      // on veut obtenir une partie t[g..d] triee

      int m=(g+d)/2;
      int[] xt = new int[d-g+1]; // tableau de travail
      int i = g; //indice de parcours dans t[g..m]
      int j = m+1;  //indice de parcours dans t[m+1..d]
      int k= 0;  //indice de parcours dans xt

      while ((i<=m) && (j<=d)) {
          if (t[i] <=t[j]) {
            xt[k]=t[i]; i++; k++;
```

```
        } else {
          xt[k]=t[j]; j++; k++;
        }
      }
      // ici soit i>m soit j>d c-a-d
      // on a epuise soit la partie t[g..m] soit la partie t[m+1..d]
      while (i<=m){
            xt[k]=t[i]; i++; k++;
      }
      while (j<=d) {
            xt[k]=t[j]; j++; k++;
      }
      //on a fini. On recopie xt dans t[g..d]
      for (i=0; i<xt.length; i++) t[g+i]= xt[i];
    } //fin fusionner
```

Exercice IV.2 Le tri rapide

Soit le tableau suivant :

150	200	100	300	80	30	50	40	75	180	20	200	35	10	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Question 1 Appliquer l'algorithme de tri-rapide pour trier les arêtes par ordre de poids croissants.

Au départ, tri-rapide(tab, 0,14)
pivot =150
i=0, j=14
avancer i et reculer j puis échanger leur contenu :

	i ↓												j ↓	
150	200	100	300	80	30	50	40	75	180	20	200	35	10	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

150	10	100	300	80	30	50	40	75	180	20	200	35	200	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Puis

150	10	100	35	80	30	50	40	75	180	20	200	300	200	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Puis

									j ↓	i ↓				
150	10	100	35	80	30	50	40	75	20	180	200	300	200	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

20	10	100	35	80	30	50	40	75	**150**	180	200	300	200	300
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

150 est maintenant à sa place définitive.

On va maintenant exécuter tri-rapide(tab, 0, 8) et tri-rapide(tab, 9, 14)

Voici tri-rapide(tab, 0, 8)

20	10	100	35	80	30	50	40	75
0	1	2	3	4	5	6	7	8

j i

20	10	100	35	80	30	50	40	75
0	1	2	3	4	5	6	7	8

10	**20**	100	35	80	30	50	40	75
0	1	2	3	4	5	6	7	8

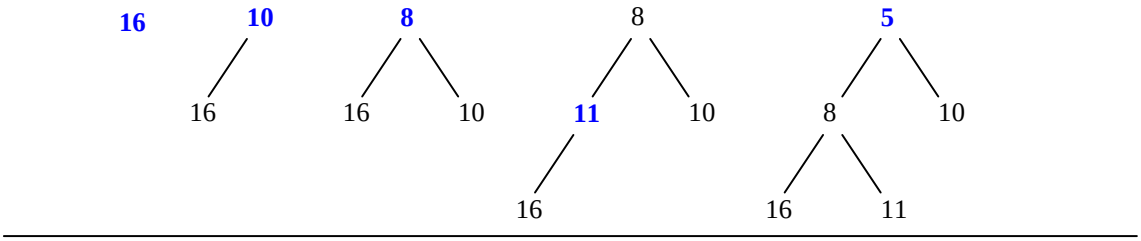
Exécute à son tour tri-rapide(tab, 0,0) et tri-rapide(tab, 2,8)
Etc …

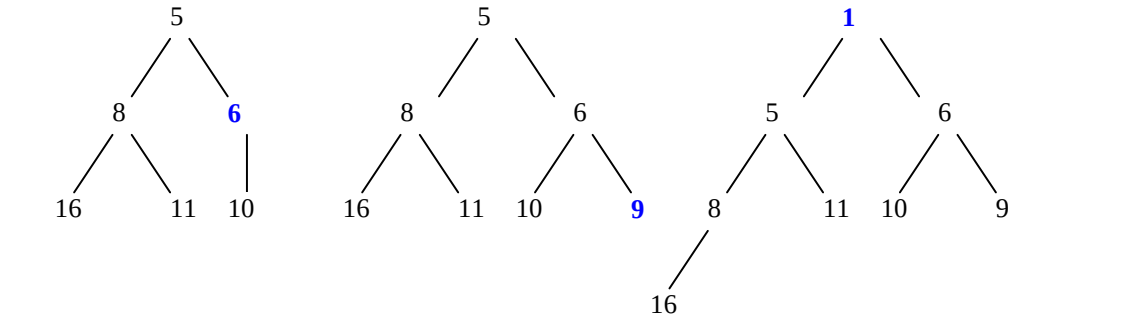
Question 2 Ecrire en pseudo-langage l'algorithme du tri-rapide.

```
procedure tri_rapide (c_cle[] tab, indice g, indice d)
      // trie la partie g..d du tableau
      indice i,j ;
      c_cle pivot;
debut
      si g<d alors
            pivot = tab(g);
            i = g;
            j = d;
            tant que (i<j) faire
                  //invariant les elem de g a i sont <= p
                  //les elem de j+1 a d sont > p
                  i := i + 1;
                  tant que((tab(i)<=pivot) et (i<j))faire i= i+1;fait
                  tant que (tab(j) > pivot) faire j = j - 1; fait
                  si (i < j) alors echanger (tab, i, j); fin si
            fait
            echanger (tab, g, j); //place definitive du pivot
            tri_rapide (tab, g, j - 1);
            tri_rapide (tab, j + 1, d);
      fin si
fin
```

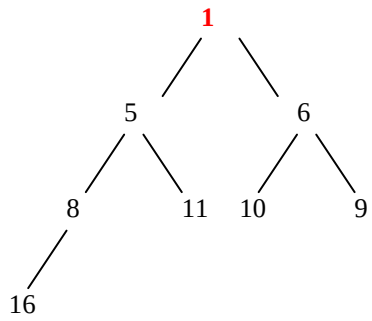
Exercice IV.3 Le tri par TAS

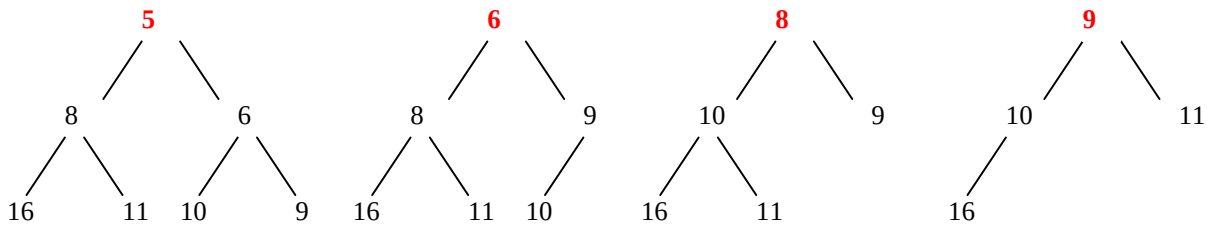
Question 1 On va appliquer les procédures d'insertion et de suppression de la racine d'un tas à l'exemple de façon à, dans un premier temps, construire un tas puis en supprimant le minimum atteindre la solution où tous les éléments sont dans l'ordre croissant. Nous allons d'abord construire le tas en 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.

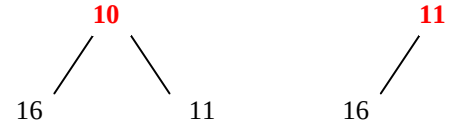




En supprimant progressivement min par min, avec la procédure supprimer_min, on passe par les tas suivants et on en tire la liste ordonnée.

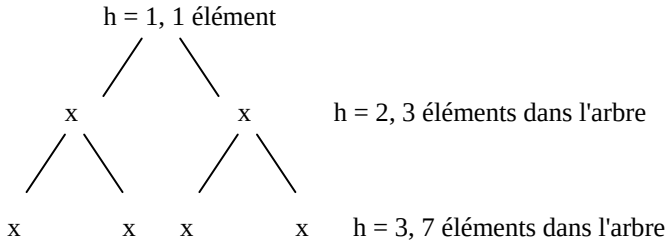






Question 2 Evaluer la complexité de l'algorithme

- procédure insérer : la complexité de l'insertion dépend de la profondeur de l'arbre (du tas). A la hauteur i, on a au plus 2^i éléments. Quelle hauteur h est nécessaire pour insérer n éléments ? Plaçons-nous dans le cas où le dernier niveau de l'arbre est plein et qu'il contient n éléments. Quelle relation existe-t-il entre h et n ?



à la profondeur h, nous avons donc $1 + 2 + 2^2 + ... + 2^{(h-1)}$ éléments, c'est-à-dire n éléments.

Donc $n = 1 + 2 + 2^2 + ... + 2^{(h-1)} = (1-2^h)/(1-2) = 2^h - 1$. On a la somme des termes d'une suite géométrique de raison 2. On en déduit que h vaut $\log_2(1+n)$.

Le nombre d'opérations de l'insertion est proportionnel à h, sa complexité est en O(h), soit en **$O(\log_2 n)$.**

- procédure supprimer_min : on retrouve la même complexité que pour l'insertion, soit **$O(\log_2 n)$.**

- procédure tri par tas : pour trier n éléments, on appelle les procédures insérer et supprimer_min n fois, la complexité totale est donc en **$O(n\log_2 n)$.**