TD n° 4 : Tri fusion sur fichier

a) On considère le fichier non trié contenant la séquence d'éléments suivante :

65 5 89 56 7 15 28 2 98 33

Donnez le contenu des 3 fichiers (appelés A, B et X dans les diapositives de cours) intervenant dans le tri fusion du fichier ci-dessus, après chaque étape d'éclatement et chaque étape de fusion.

Fichier original : (65) (5) (89) (56) (7) (15) (28) (2) (98) (33)
Remarque : les parenthèses ne sont pas réellement dans le fichier, je les mets juste pour indiquer les différentes monotonies. On a donc au départ 10 monotonies de longueur 1.

Eclatement sur deux fichiers (1 monotonie sur 2 est copiée dans le fichier A, 1 sur 2 est copiée dans le fichier B) :
Fichier A : (65) (89) (7) (28) (98)
Fichier B : (5) (56) (15) (2) (33)

Fusion des monotonies 2 à 2 : la première du fichier A avec la première du fichier B, etc. On obtient 5 monotonies de longueur 2 que l'on écrit dans le fichier X.
Fichier X : (5 65) (56 89) (7 15) (2 28) (33 98)

Eclatement sur 2 fichiers (on écrase le contenu précédent des fichiers A et B) :
Fichier A : (5 65) (7 15) (33 98)
Fichier B : (56 89) (2 28)

Fusion dans le fichier X (on écrase le contenu précédent du fichier X) : on obtient 2 monotonies de longueur 4 et une de longueur 2.
Fichier X : (5 56 65 89) (2 7 15 28) (33 98)

Eclatement sur 2 fichiers (on écrase le contenu précédent des fichiers A et B) :
Fichier A : (5 56 65 89) (33 98)
Fichier B : (2 7 15 28)

Fusion dans le fichier X (on écrase le contenu précédent du fichier X) : on obtient 1 monotonies de longueur 8 et une de longueur 2.
Fichier X : (2 5 7 15 28 56 65 89) (33 98)

Eclatement sur 2 fichiers (on écrase le contenu précédent des fichiers A et B) :
Fichier A : (2 5 7 15 28 56 65 89)
Fichier B : (33 98)

Fusion dans le fichier X (on écrase le contenu précédent du fichier X) : on obtient 1 monotonies de longueur 10 : le fichier est complètement trié.
Fichier X : (2 5 7 15 28 33 56 65 89 98)

b) Combien de comparaisons d'éléments effectue-t-on au pire lorsqu'on fusionne une monotonie de longueur L_a avec une autre monotonie de longueur L_b ?

Le cas le pire est celui de 2 monotonies dont les éléments sont « entremêlés », c'est-à-dire le cas où on n'épuise pas une monotonie « prématurément ».

Exemple 1 : fusion de {1, 6} avec {8, 23, 51}, 2 comparaisons, puis on sait qu'on peut recopier directement la deuxième monotonie sans faire davantage de comparaisons.

Exemple 2 : fusion de {1, 8} avec {2, 6, 23} : on doit faire 4 comparaisons.

On a le cas le pire quand le dernier élément de A ne peut être « éliminé » (des éléments à traiter) que par une comparaison avec le dernier élément de B, et réciproquement. Plus formellement, les éléments 1..(L_b-1) de B sont inférieurs au dernier élément de A, et réciproquement, les éléments 1..(L_a-1) sont inférieurs au dernier élément de B.

Lors de la fusion des deux monotonies, on écrit $L_a + L_b$ éléments dans le fichier X. Dans le cas le pire, on n'écrit jamais plus d'un élément à la fois. On a donc $L_a + L_b$ étapes d'écriture. Chaque écriture d'élément est précédée d'une comparaison, sauf pour le dernier élément. **On a donc au pire $L_a + L_b$ - 1 comparaisons à effectuer**.

c) Combien de comparaisons d'éléments effectue-t-on au pire lorsqu'on trie par fusion un fichier de *n* éléments, dans le cas particulier où *n* est une puissance de 2 ($n=2^p$) ?

L'intérêt de choisir n puissance de 2 est qu'on aura à chaque étape un nombre pair de monotonies à fusionner, et que les monotonies d'une étape donnée auront toutes la même longueur, même la dernière. Dans le cas plus général où n n'est pas une puissance de 2 (cf question a pour un exemple), FicA peut contenir une monotonie de plus (éventuellement incomplète) que FicB. Et si au contraire FicA et FicB contiennent le même nombre de monotonies, alors la dernière monotonie de FicB peut être incomplète.

Première passe : On a n monotonies de longueur 1 à fusionner 2 à 2, soit n/2 opérations de fusion. On a donc (n/2)*(1+1-1) = n/2 comparaisons à effectuer.

Deuxième passe : On a n/2 monotonies de longueur 2, à fusionner 2 à 2, soit n/4 opérations de fusion. On va donc effectuer (n/4)*(2+2-1) = 3n/4 comparaisons.

Troisième passe : On a n/4 monotonies de longueur 4 à fusionner 2 à 2, soit (n/8)*(4+4-1)=7n/8 comparaisons à réaliser.

k-ieme passe :
A chaque passe, on multiplie par deux la longueur des monotonies et on divise par deux le nombre de monotonies. Avant la k-ieme passe, on a $n/2^{k-1}$ monotonies de longueur 2^{k-1} à fusionner 2 à 2, soit $n/2^k$ opérations de fusion à effectuer. On a donc $(n/2^k)*(2^{k-1} + 2^{k-1} -1) = (n/2^k)*(2^k - 1)$ comparaisons à réaliser. A l'issue de la k-ieme passe, on a $n/2^k$ monotonies de longueur 2^k.

On s'arrête quand on n'a plus qu'une monotonie de longueur n : $2^{k_{final}} = n = 2^p$, soit $k_{final} = p$. On doit donc effectuer p passes sur le fichier.

Le nombre total de comparaisons est donc :

$$C = \sum_{k=1}^{p} \frac{n(2^k - 1)}{2^k} = n\sum_{k=1}^{p} \frac{(2^k - 1)}{2^k} = n\left(\sum_{k=1}^{p} 1 - \sum_{k=1}^{p} \frac{1}{2^k}\right) = n\left(p - \sum_{k=1}^{p} \frac{1}{2^k}\right)$$

On reconnaît la somme partielle d'une suite géométrique de raison q=1/2. Soit S cette somme.

$$
\begin{array}{rcll}
S & = & 1/2 + 1/4 + 1/8 + \ldots + 1/2^p & \\
qS & = & 1/4 + 1/8 + \ldots + 1/2^p + 1/2^{p+1} & \\
\multicolumn{4}{c}{\text{----------}} \\
S - qS & = & 1/2 - 1/2^{p+1} & \text{(les autres termes s'éliminent)} \\
S & = & (1/2 - 1/2^{p+1}) / (1 - q) & \\
S & = & 1 - 1/2^p & \text{(sachant que q=1/2)}
\end{array}
$$

Donc $C = n\left(p - 1 + \frac{1}{2^p}\right) = n\left(\log_2(n) - 1 + \frac{1}{n}\right)$

Au final, on obtient donc $C = n\log_2(n) - n + 1 = O(n\log_2(n))$ comparaisons pour trier par fusion un fichier de n éléments, dans le cas particulier où n est une puissance de 2.

Remarque : dans le cas général, c'est-à-dire pour n quelconque, il est possible de montrer que l'ordre de grandeur est inchangé, l'algorithme est toujours $O(n\log_2(n))$. Pour en savoir plus, voir le chapitre 4 de Cormen et al., Introduction à l'algorithmique, 2e édition. L'ouvrage est disponible à la BU.

d) Ecrire en langage algorithmique la procédure de tri par fusion d'un fichier, en supposant que vous disposez déjà des procédures « eclatement » et « fusion » (voir les entêtes ci-dessous). La procédure de tri doit appeler les procédures « eclatement » et « fusion » jusqu'à ce que le fichier soit complètement trié.

Procédure eclatement(nomFicX : chaîne de caractères, lg : entier, nomFicA : chaîne de caractères, nomFicB : chaîne de caractères)
Précondition : le fichier appelé nomFicX contient des monotonies de longueur lg, sauf peut-être la dernière qui peut être plus courte
Postcondition : Les monotonies de nomFicX sont réparties (1 sur 2) dans des fichiers appelés nomFicA et nomFicB : : la première monotonie est copiée dans le fichier A, la seconde dans le fichier B, la troisième dans le A, etc. Si ces fichiers existaient déjà, leur contenu est écrasé. Le fichier A peut recueillir une monotonie de plus le fichier B, et cette dernière monotonie peut être de longueur inférieure à lg. Si au contraire ficA et ficB recueillent le même nombre de monotonies, la dernière monotonie écrite dans ficB peut être de longueur inférieure à lg.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg
{Les *chaînes de caractères* contenant les noms des fichiers ne vont pas être affectées par la procédure, d'où le passage en mode donnée, mais bien sûr, les *fichiers* désignés par nomFicA et nomFicB vont être affectés, comme cela est précisé dans les post-conditions.}

Procédure fusion(nomFicA : chaîne de caractères, nomFicB : chaîne de caractères, lg : entier, nomFicX : chaîne de caractères, nbMonoDansX : entier

Préconditions : les fichiers appelés nomFicA et nomFicB contiennent des monotonies de longueur lg. FicA peut contenir une monotonie de plus (éventuellement incomplète) que FicB. Si au contraire FicA et FicB contiennent le même nombre de monotonies, alors la dernière monotonie de FicB peut être incomplète.
Postconditions : le fichier appelé nomFicX contient nbMonoApres monotonies de longueur 2*lg, la dernière pouvant être plus courte. Ces monotonies résultent de la fusion 2 à 2 des monotonies de FicA et de FicB. Si FicX existait déjà, son ancien contenu est écrasé.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg
Paramètre en mode résultat : nbMonoDansX

Procédure tri_par_fusion(nomFic : chaîne de caractères)
Précondition : le fichier appelé nomFic contient des Elements
Postcondition : le fichier appelé nomFic contient les mêmes éléments, mais triés.
Paramètres en mode donnée : nomFic
Variables locales :
    longueur, nbMonotonies : entier
    nomA, nomB : chaînes de caractères
Début
    longueur ← 1
    nomA ← « fichierAnnexeA.txt »
    nomB ← « fichierAnnexeB.txt »
    Répéter
        eclatement(nomFic, longueur, nomA, nomB)
        fusion(nomA, nomB, longueur, nomFic, nbMonotonies)
        longueur ← longueur * 2
    Jusqu'à ce que (nbMonotonies = 1)
Fin tri_par_fusion

e) Donnez le code de la procédure d'éclatement en langage C.

```
/* Precondition : le fichier appele nomFicX contient des monotonies de
   longueur lg, sauf peut-etre la derniere qui peut etre plus courte.

   Postconditions : Les monotonies de nomFicX sont reparties dans des
   fichiers appeles nomFicA et nomFicB : la premiere monotonie est copiee
   dans le fichier A, la seconde dans le fichier B, la troisieme dans A, etc.
   Si ces fichiers existaient deja, leur contenu est ecrase. Le fichier A
   peut recueillir une monotonie de plus que le fichier B, et cette
   derniere monotonie peut etre de longueur inferieure a lg. Si au contraire
   ficA et ficB recueillent le meme nombre de monotonies, la derniere
   monotonie ecrite dans ficB peut etre de longueur inferieure a lg. */

void eclatement(const char * nomFicX, int lg, \
                const char * nomFicA, const char * nomFicB)
{
  FILE * ficX, *ficA, *ficB ;
  int mettreDansFicA = 1;
  int i = 0;
  double elem_lu;

  ficX = fopen(nomFicX, "rb");
  if (ficX == NULL)
    {
      printf("Impossible de lire le fichier %s\n", nomFicX);
      printf("Il s'agit peut-etre d'un probleme de droits d'acces, \n");
      printf("ou peut-etre que le fichier n'existe pas. \n");
      exit(EXIT_FAILURE);
    }

  ficA = std::fopen(nomFicA, "wb");
  if (ficA == NULL)
    {
      printf("Impossible de creer le fichier %s en ecriture\n", nomFicA);
      printf("Il s'agit peut-etre d'un probleme de droits d'acces.\n");
      exit(EXIT_FAILURE);
    }


  ficB = std::fopen(nomFicB, "wb");
  if (ficB == NULL)
```

```
    {
      printf("Impossible de creer le fichier %s en ecriture\n", nomFicB);
      printf("Il s'agit peut-etre d'un probleme de droits d'acces.\n");
      exit(EXIT_FAILURE);
    }



  /* On boucle en fonction de la valeur de retour de fread et non en fonction
     de feof, cf cours magistral */
  while(fread(&elem_lu, sizeof(double), 1, ficX) == 1)
    {
      if (mettreDansFicA == 1)  fwrite(&elem_lu, sizeof(double), 1, ficA);
      else fwrite(&elem_lu, sizeof(double), 1, ficB);

      i++;
      if (i == lg)
       {
         if (mettreDansFicA == 1) mettreDansFicA = 0;
         else mettreDansFicA = 1;
         i = 0;
       }
    }

  fclose(ficX);
  fclose(ficA);
  fclose(ficB);
}
```