TD7 : Tri fusion (1/2)

Exercice 1 : Fusion de deux monotonies sur fichiers

On dispose de deux fichiers contenant chacun une séquence triée de réels. En d'autres termes, chacun des deux fichiers contient une monotonie et une seule – il s'agit donc d'un problème plus simple que celui du tri fusion sur fichiers (cf. exercice 2). On veut écrire une procédure qui fusionne ces deux monotonies et écrit la monotonie résultante dans un troisième fichier. Cette procédure prend comme *seuls* paramètres les noms des deux fichiers d'entrée (**nomfic1** et **nomfic2**) et le nom du fichier de sortie (**nomficSortie**).

Préconditions :

- nomfic1 et nomfic2 sont des fichiers texte qui contiennent chacun une séquence triée (monotonie) de nombres au format IEEE 754 double précision. Si l'un de ces fichiers ne peut pas être ouvert (par exemple parce qu'il n'existe pas), alors le programme se termine avec un code d'échec.
- Les deux séquences peuvent être de longueur différente, mais elles sont au moins de longueur 1 : autrement dit, chaque fichier contient au moins un élément.

Postconditions :

- Un nouveau fichier nommé nomficSortie est créé (s'il existait déjà, son contenu est écrasé). Ce fichier contient une monotonie correspondant à la fusion des deux monotonies contenues dans nomfic1 et nomfic2. Si ce fichier ne peut pas être créé (par exemple à cause d'un problème de droits insuffisants dans le répertoire courant), le programme se termine avec un code d'échec.
- Les contenus des fichiers nomfic1 et nomfic2 sont inchangés.

Donnez le code C++ de cette procédure. Vous utiliserez les variables locales suivantes (à vous d'en préciser le type) : **val1, val2, fic1, fic2, ficSortie, succeslect1, succeslect2**. Vous pouvez en ajouter d'autres si nécessaire.

Exercice 2 : Complexité du tri fusion sur fichier

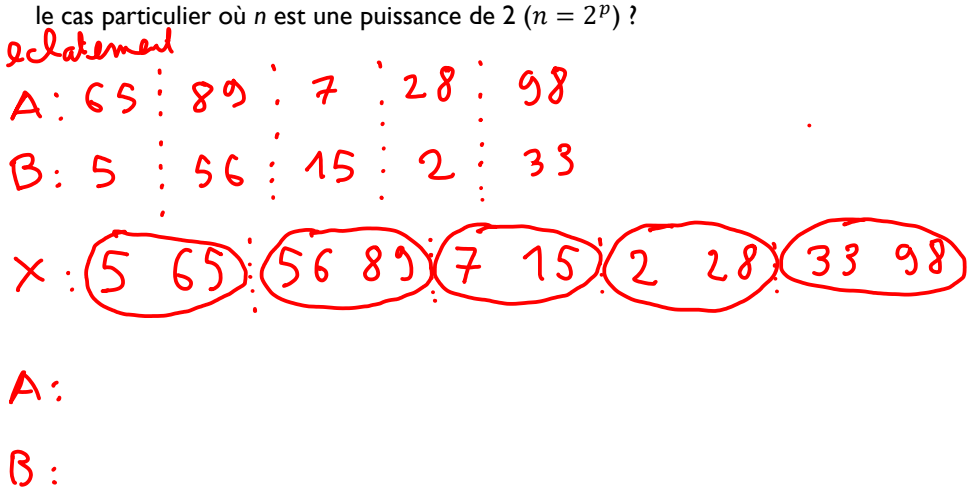
On considère le fichier non trié contenant la séquence d'éléments suivante :

65	5	89	56	7	15	28	2	98	33

a. 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.

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 ?

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 ($n = 2^p$) ?



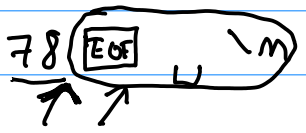
```
ifstream fic1(nomfic1.c_str());
    if(fic1.is_open()) ....
ofstream ficsortie(nomficsortie.c_str());

fic1 >> x;
ficsortie << y;

if(!fic1.eof()) ---
                    fic1 >> x ...

fic1 >> x;
if(!fic1.eof()) alors OK
                sinon c'est qu'on était déjà à la fin
```

3 56 78 [EOF] \n



X : (5 65) (56 89) (7 15) (2 28) (33 98) $\lg = 2$

éclate

A : 5 65 : 7 15 : 33 98

B : 56 89 : 2 28 :

fusion

X : 5 56 65 89 : 2 7 15 28 : 33 98 $\lg = 4$

A : 5 56 65 89 : 33 98

B : 2 7 15 28 :

X : 2 5 7 15 28 56 65 89 : 33 98 $\lg = 8$

A : 2 5 7 15 28 56 65 89

B : 33 98

X : 2 5 7 15 28 33 56 65 89 98 $\lg = 16$

TD7 : Tri fusion (1/2)

Exercice 1 : Fusion de deux monotonies sur fichiers

On dispose de deux fichiers contenant chacun une séquence triée de réels. En d'autres termes, chacun des deux fichiers contient une monotonie et une seule – il s'agit donc d'un problème plus simple que celui du tri fusion sur fichiers (cf. exercice 2). On veut écrire une procédure qui fusionne ces deux monotonies et écrit la monotonie résultante dans un troisième fichier. Cette procédure prend comme *seuls* paramètres les noms des deux fichiers d'entrée (**nomfic1** et **nomfic2**) et le nom du fichier de sortie (**nomficSortie**).

Préconditions :

- nomfic1 et nomfic2 sont des fichiers texte qui contiennent chacun une séquence triée (monotonie) de nombres au format IEEE 754 double précision. Si l'un de ces fichiers ne peut pas être ouvert (par exemple parce qu'il n'existe pas), alors le programme se termine avec un code d'échec.
- Les deux séquences peuvent être de longueur différente, mais elles sont au moins de longueur 1 : autrement dit, chaque fichier contient au moins un élément.

Postconditions :

- Un nouveau fichier nommé nomficSortie est créé (s'il existait déjà, son contenu est écrasé). Ce fichier contient une monotonie correspondant à la fusion des deux monotonies contenues dans nomfic1 et nomfic2. Si ce fichier ne peut pas être créé (par exemple à cause d'un problème de droits insuffisants dans le répertoire courant), le programme se termine avec un code d'échec.
- Les contenus des fichiers nomfic1 et nomfic2 sont inchangés.

Donnez le code C++ de cette procédure. Vous utiliserez les variables locales suivantes (à vous d'en préciser le type) : **val1, val2, fic1, fic2, ficSortie, succeslect1, succeslect2**. Vous pouvez en ajouter d'autres si nécessaire.

```
/* Préconditions :
    - nomfic1 et nomfic2 sont des fichiers qui contiennent chacun une sequence
      triée de nombres au format double (monotonie).
    - les deux séquences peuvent être de longueur différente, mais elles sont
      au moins de longueur 1.
   Postcondition :
    - un nouveau fichier nomme nomficSortie est créé (s'il existait déjà, son
      contenu est écrasé). Ce fichier contient la fusion des deux monotonies
*/
void fusionner(const string& nomfic1,const string& nomfic2,const string& nomficSortie){
  double val1, val2;
  bool succeslect1 = true, succeslect2 = true;

  ifstream fic1 (nomfic1.c_str());
  if (!fic1.is_open()) {
      cout << "Impossible d'ouvrir en lecture le fichier : " << nomfic1;
      exit(EXIT_FAILURE);
  }

  ifstream fic2 (nomfic2.c_str());
  if (!fic2.is_open()) {
      cout << "Impossible d'ouvrir en lecture le fichier : " << nomfic2;
      exit(EXIT_FAILURE);
  }

  ofstream ficSortie (nomficSortie.c_str());
  if (!ficSortie.is_open()) {
      cout << "Impossible d'ouvrir en écriture le fichier : " << nomficSortie;
      exit(EXIT_FAILURE);
  }

  fic1 >> val1; // il y a au moins un élément dans le fichier
```

```
  fic2 >> val2; // il y a au moins un élément dans le fichier

  while (succeslect1 && succeslect2) {
    if (val1 < val2) {
      ficSortie << val1 << " ";
      fic1 >> val1;
      if (fic1.eof()) succeslect1 = false;
    }
    else {
      ficSortie << val2 << " ";
      fic2 >> val2;
      if (fic2.eof()) succeslect2 = false;
    }
  }

  if (!succeslect1) {
    /* On a épuisé fic1, il reste à recopier la fin de fic2 */
    do {
      ficSortie << val2 << " ";
      fic2 >> val2;
    }
    while (!fic2.eof());
  }
  else {
    /* On a épuisé fic2, il reste à recopier la fin de fic1 */
    do {
      ficSortie << val1 << " ";
      fic1 >> val1;
    }
    while (!fic1.eof());
  }

  fic1.close();
  fic2.close();
  ficSortie.close();
}
```

Exercice 2 : Complexité du tri fusion sur fichier

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

65	5	89	56	7	15	28	2	98	33

a. 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, elles sont là 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, l'autre 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 1 de longueur 2.
Fichier X : (5 56 65 89) (2 7 15 28) (33 98)

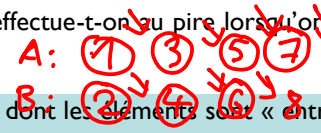
Eclatement sur 2 fichiers (on écrase le contenu précédent des fichiers A et B) :
Fichier A : (5 56 65 89) (33 98)
Fichier B : (2 7 15 28)

Fusion dans le fichier X (on écrase le contenu précédent du fichier X) : on obtient 1 monotonie de longueur 8 et 1 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 monotonie 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 » (équilibrés), 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 pire cas 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 pire cas, 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 ($n = 2^p$) ?

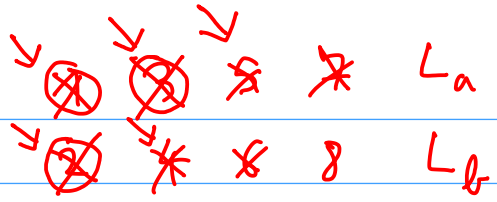
L'intérêt de choisir n puissance de 2 est qu'on aura à chaque étape un nombre pair de monotonies à fusionner, et que les monotonies d'une étape donnée auront toutes la même longueur, même la dernière. Dans le cas plus général où n n'est pas une puissance de 2 (cf. question a pour un exemple), FicA peut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

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

Itération 2 : 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.

Itération 3 : On a n/4 monotonies de longueur 4 à fusionner 2 à 2, soit (n/8)*(4+4-1)=7n/8 comparaisons à réaliser.

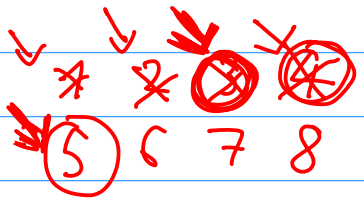
Itération k : A chaque itération, on multiplie par deux la longueur des monotonies et on divise par deux le nombre de monotonies. Avant la k-ieme itération, on a $n/2^{k-1}$ monotonies de longueur 2^{k-1} à fusionner 2 à 2, soit $n/2^k$



// $L_a + L_b - 1$ comparaisons

comp ↓

sortie 1 2 3 4 5 6 7 8



1 2 3 4 5 6 7 8 // L_a comparaisons

L_a

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 itération, on a $n/2^k$ monotonies de longueur 2^k.
Rappel : $2^{k-1} + 2^{k-1} = 2 \times 2^{k-1} = 2^{1+k-1} = 2^k$

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 itérations sur le fichier.

Le nombre total de comparaisons est donc :
$C = \sum_{k=1}^{p} \frac{n(2^k-1)}{2^k} = n\sum_{k=1}^{p} \frac{(2^k-1)}{2^k} = n\left(\sum_{k=1}^{p} 1 - \sum_{k=1}^{p} \frac{1}{2^k}\right) = n\left(p - \sum_{k=1}^{p} \frac{1}{2^k}\right)$
Soit $S = \sum_{k=1}^{p} \frac{1}{2^k}$. On reconnaît la somme d'une suite géométrique $\sum_{i=0}^{n} b^i = \frac{1-b^{n+1}}{1-b}$ de raison b=1/2.
On a donc $S = \frac{1-\frac{1}{2^{p+1}}}{1-\frac{1}{2}} - 1 = 2 - \frac{2}{2^{p+1}} - 1 = 1 - \frac{1}{2^p}$

Donc $C = n\left(p - 1 + \frac{1}{2^p}\right) = n\left(\log n - 1 + \frac{1}{n}\right) = n\log n - n + 1$
Au final, on obtient donc $O(n \log n)$ 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 $O(n \log n)$. Pour en savoir plus, voir le chapitre 4 de Cormen et al., Introduction à l'algorithmique, 2ème 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
Postconditions : 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 que 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

Remarque : 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 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 nbMonoDansX 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 donnée-résultat : nbMonoDansX

Procédure tri_par_fusion (nomFic : chaîne de caractères)
Précondition : le fichier appelé nomFic contient des éléments
Postcondition : le fichier appelé nomFic contient les mêmes éléments, mais triés.

$n = 2^p$

étape	longueur monotonie	nombre de monotonies dans A (ou B)	$L_A + L_B - 1$	# comparaisons
0	1	$n/2$	$1+1-1=1$	$n/2 \times 1$
1	2	$n/4$	$2+2-1=3$	$n/4 \times 3$
2	4	$n/8$	$2\times4-1=7$	$n/8 \times 7$
⋮				
i	2^i	$n/2^{i+1}$	$2\times2^i-1 = 2^{i+1}-1$	$\frac{n}{2^{i+1}} \times (2^{i+1}-1)$
⋮				
$p-1$	2^{p-1}	1	$2\times2^{p-1}-1 = 2^p-1$	$\frac{n}{2^p} \times (2^p-1) = 2^p-1$

$$\sum_{i=0}^{p-1} \frac{n}{2^{i+1}} \times (2^{i+1}-1) = \sum_{i=0}^{p-1} n - n\sum_{i=0}^{p-1} \frac{1}{2^{i+1}}$$

somme des h premiers termes d'une suite géométrique de premier terme a et de raison r:

$$S_h = a \times \frac{1-r^h}{1-r}$$

j'avais oublié le premier terme lors de la séance

$$= np - n \times \frac{1}{2} \times \frac{1-\left(\frac{1}{2}\right)^p}{1-\frac{1}{2}}$$

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

$$= np - n\left(1-\frac{1}{n}\right) \quad \text{car } n = 2^p$$

$$= np - n + 1$$

$$= n\log_2 n - n + 1 \quad \text{car } p = \log_2 n$$

dans notre cas, $a = 1$ fait:

$$\sum_{i=0}^{p-1} \frac{1}{2^{i+1}} = \frac{1}{2} \times \frac{1-\left(\frac{1}{2}\right)^p}{1-\frac{1}{2}} = 1 - \frac{1}{2^p}$$