Introduction à l'algorithmique : Compléments

Serge Haddad

LMF, ENS Paris-Saclay & CNRS

M2 FESUP

Plan

La $k$ième valeur d'un tableau

Soit T un tableau de taille n.

Une valeur v est la $k$ième valeur de T si :

- $|\{i \mid T[i] < v\}| < k$
- et $|\{i \mid T[i] > v\}| \leq n - k$.

Un algorithme en $O(n^2)$

```
For i from 1 to n do
  v ← T[i] ; ℓ ← 0 ; u ← 0 ;
  For j from 1 to n do
    If T[j] < v then ℓ ← ℓ + 1 ;
    Else If T[j] > v then u ← u + 1 ;
  If ℓ < k and u ≤ n − k then
    return(v) ;
```

Un algorithme en $O(n \log(n))$

```
Sort(T, n) ;
return(T[k]) ;
```

Un algorithme récursif

KSel(T, n, k)

If $n < 30$ **then Sort**(T, n) ; **return**$(T[k])$; (petites instances)

$mid \leftarrow T[1]$; $n_1 \leftarrow 0$; $n_2 \leftarrow 0$; $n_3 \leftarrow 0$;

For i **from** 1 **to** n **do** (partition de T)

  If $T[i] < mid$ **then** $n_1 \leftarrow n_1 + 1$; $T_1[n_1] \leftarrow T[i]$;

  Else If $T[i] = mid$ **then** $n_2 \leftarrow n_2 + 1$;

  Else $n_3 \leftarrow n_3 + 1$; $T_3[n_3] \leftarrow T[i]$;

If $n_1 \geq k$ **then return**$(KSel(T_1, n_1, k))$; (recherche récursive)

If $n_1 + n_2 \geq k$ **then return** (mid) ;

return$(KSel(T_3, n_3, k - (n_1 + n_2)))$;

Correction

Observation. La correction est vérifiée quelque soit le choix de mid.

Dans T, il y a n_1 valeurs inférieures à mid copiées dans T_1.

Dans T, il y a n_2 valeurs égales à mid.

Dans T, il y a n_3 valeurs supérieures à mid copiées dans T_3.

Cas $k \leq n_1$. La $k$ième valeur de T est la $k$ième valeur de T_1.

Cas $n_1 < k \leq n_1 + n_2$. La $k$ième valeur de T est mid.

Cas $n_1 + n_2 < k$. La $k$ième valeur de T est la $k - (n_1 + n_2)$ième valeur de T_3.

Un meilleur algorithme récursif

KSel(T, n, k)

If $n < 30$ **then Sort**(T, n) ; **return**$(T[k])$; (petites instances)

$m \leftarrow \lfloor \frac{n}{5} \rfloor$; (choix d'un médian)

For i **from** 1 **to** m **do**

  For j **from** 1 **to** 5 **do** $S[j] \leftarrow T[5 * (i-1) + j]$

  Sort$(S, 5)$; $R[i] \leftarrow S[3]$;

$mid \leftarrow \mathbf{KSel}(R, m, \lceil \frac{m}{2} \rceil)$; $n_1 \leftarrow 0$; $n_2 \leftarrow 0$; $n_3 \leftarrow 0$;

For i **from** 1 **to** n **do** (partition de T)

  If $T[i] < mid$ **then** $n_1 \leftarrow n_1 + 1$; $T_1[n_1] \leftarrow T[i]$;

  Else If $T[i] = mid$ **then** $n_2 \leftarrow n_2 + 1$;

  Else $n_3 \leftarrow n_3 + 1$; $T_3[n_3] \leftarrow T[i]$;

If $n_1 \geq k$ **then return**$(KSel(T_1, n_1, k))$; (recherche récursive)

If $n_1 + n_2 \geq k$ **then return** (mid) ;

return$(KSel(T_3, n_3, k - (n_1 + n_2)))$;

Complexité

Soit a le pire temps d'exécution sur une instance de taille inférieure à 30.

Soit bn une borne du temps d'exécution hors appels récursifs de $\mathbf{KSel}$ sur une instance de taille $n \geq 30$.

On a :

$$\forall n < 30 \; T(n) \leq a \;\text{ et }\; \forall n \geq 30 \; T(n) \leq T(\lfloor \frac{n}{5} \rfloor) + \max(T(n_1), T(n_3)) + bn$$

Or

$$n - \max(n_1, n_3) \geq 3(\lceil \frac{m}{2} \rceil) \geq \frac{3}{2}m \geq \frac{3}{2}(\frac{n}{5} - 1) = \frac{3n}{10} - \frac{3}{2}$$

D'où : $\max(n_1, n_3) \leq \frac{7n}{10} + \frac{3}{2} \leq \frac{3n}{4}$ dès que $n \geq 30$.

Puisque $\frac{1}{5} + \frac{3}{4} = \frac{19}{20} < 1$,

par application des équations de récurrence, $\mathbf{KSel}$ opère en $\Theta(n)$ ...

mais la constante multiplicative est élevée (≥ 20).

Plan

Recherche de la $k$ième valeur d'un tableau

2 Recherche d'une valeur majoritaire

Une fusion en place

Le problème de la valeur majoritaire

Soit T un tableau de taille n.

Une valeur v est *majoritaire* si $|\{i \mid T[i] = v\}| > \frac{n}{2}$.

Le problème de la valeur majoritaire consiste à décider

- s'il existe une valeur majoritaire
- et dans l'affirmative à la renvoyer.

Pour la borne inférieure de complexité,
on fait l'hypothèse qu'on ne peut tester que l'égalité de valeurs.

Un algorithme en $O(n^2)$

For i **from** 1 **to** n **do**
  $v \leftarrow T[i]$; $occ \leftarrow 0$;
  For j **from** 1 **to** n **do**
    If $T[j] = v$ **then**
      $occ \leftarrow occ + 1$; **If** $occ > \frac{n}{2}$ **then return**$(\mathbf{true}, v)$;
return$(\mathbf{false}, v)$;

Un algorithme en $O(n \log(n))$

Lorsque l'espace des valeurs est ordonné.

Sort(T, n) ;
$v \leftarrow T[1]$; $i \leftarrow 2$; $occ \leftarrow 1$;
While $occ \leq \frac{n}{2}$ **and** $i \leq n$ **do**
  If $T[i] = v$ **then**
    $occ \leftarrow occ + 1$;
  Else
    $occ \leftarrow 1$; $v \leftarrow T[i]$;
  $i \leftarrow i + 1$;
return$(occ > \frac{n}{2}, v)$;

Un algorithme en $O(n)$

Observation.

Lorsque l'espace des valeurs est ordonné,

s'il existe une valeur majoritaire alors

cette valeur est la $\lceil \frac{n}{2} \rceil$ième valeur de T.

$v \leftarrow \mathbf{KSel}(T, n, \lceil \frac{n}{2} \rceil)$;
$occ \leftarrow 0$;
For i **from** 1 **to** n **do**
  If $T[i] = v$ **then**
    $occ \leftarrow occ + 1$;
    If $occ > \frac{n}{2}$ **then return**$(\mathbf{true}, v)$;
return$(\mathbf{false}, v)$;

Un algorithme optimal

shelf et *box* sont deux piles. Le « sommet » d'une pile vide est $\perp$.

$shelf \leftarrow \perp$; $box \leftarrow \perp$; $first \leftarrow \mathbf{true}$; (Première étape)
For i **from** 1 **to** n **do**
  If $first$ **or** $T[i] \neq \mathbf{Top}(shelf)$ **then**
    $\mathbf{Push}(shelf, T[i])$; **If** $\mathbf{Top}(box) \neq \perp$ **then** $\mathbf{Push}(shelf, \mathbf{Pop}(box))$;
  Else $\mathbf{Push}(box, T[i])$;
  $first \leftarrow \mathbf{false}$
$v \leftarrow \mathbf{Top}(shelf)$; $first \leftarrow \mathbf{true}$
While $\mathbf{Top}(shelf) \neq \perp$ **do** (Deuxième étape)
  $v' \leftarrow \mathbf{Pop}(shelf)$;
  If $first$ **or** $v' = v$ **then**
    If $\mathbf{Pop}(shelf) = \perp$ **then** $\mathbf{Push}(box, v)$
  Else If $\mathbf{Pop}(box) = \perp$ **then return**$(\mathbf{false}, v)$;
  $first \leftarrow \mathbf{false}$
return$(\mathbf{Top}(box) \neq \perp, v)$;

Déroulement de l'algorithme

$$T = [1, 1, 2, 2, 3, 3, 2, 3, 1, 1, 1]$$

On représente les états par $(\mathit{shelf}, \mathit{box})$.

Première étape.

$$([1], []) \to (([1], [1])) \to ([1, 2, 1], []) \to^* ([1, 2, 1, 2, 3], []) \to ([1, 2, 1, 2, 3], [3])$$

$$\to ([1, 2, 1, 2, 3, 2, 3], []) \to ([1, 2, 1, 2, 3, 2, 3], [3]) \to ([1, 2, 1, 2, 3, 2, 3, 1, 3], [])$$

$$\to ([1, 2, 1, 2, 3, 2, 3, 1, 3, 1], []) \to ([1, 2, 1, 2, 3, 2, 3, 1, 3, {\color{red}1}], [1])$$

Deuxième étape $(v = 1)$.

$$\to ([1, 2, 1, 2, 3, 2, 3, 1], [1]) \to ([1, 2, 1, 2, 3, 2], [1]) \to ([1, 2, 1, 2, 3], [])$$

Il n'y a pas de valeur majoritaire.

Complexité de l'algorithme

A la fin de la première étape,

- il y a eu $n - 1$ comparaisons de valeurs
- et les deux piles contiennent (ensemble) les n valeurs de T.

Durant la deuxième étape,

- lors de la première itération il n'y a pas de comparaison et deux éléments sont dépilés (si $n > 1$).
- lors de la dernière itération en cas d'échec ou si n est impair il y a une comparaison et un seul élément est dépilé,
- lors d'une autre itération il y a une comparaison et deux éléments sont dépilés.

D'où un nombre de comparaisons égal à $\lceil \frac{n}{2} \rceil - 1$.

Soit un nombre total égal à $\boxed{n + \lceil \frac{n}{2} \rceil - 2}$.

Correction de l'algorithme (1)

Invariant de boucle de la première étape.

> $shelf$ ne contient pas deux valeurs identiques contiguës
> et toutes les valeurs de box sont égales au sommet de $shelf$.

Preuve. (vrai initialement)

- Si la valeur courante est identique au sommet $shelf$
  elle est empilée dans box
- Si la valeur courante est différente du sommet $shelf$ elle est empilée dans $shelf$
  et (éventuellement) le sommet de box est dépilé puis empilé dans $shelf$.

Dans les deux cas, l'invariant reste vérifié.

A la fin de la première étape

Il y a m valeurs dans $shelf$
et $n - m$ valeurs dans box égales à v le sommet de $shelf$.

Toute valeur différente de v apparait au plus $\lfloor \frac{m}{2} \rfloor$ et ne peut être majoritaire.

Correction de l'algorithme (2)

Chaque paire de valeurs dépilées contient v et une valeur différente de v.

Par conséquent, v est majoritaire ssi v est majoritaire parmi les valeurs restantes.

- Cas d'échec « prématuré »
  box est vide.
  $shelf$ contient α valeurs sans valeurs identiques contiguës
  dont la dernière valeur est différente de v.
  Par conséquent il y a au plus $\lfloor \frac{\alpha}{2} \rfloor$ occurrences de v.

- Cas d'échec final
  box et $shelf$ sont vides.
  Donc il y a exactement $\frac{n}{2}$ occurrences de v.

- Cas de succès
  $shelf$ est vide et box contient des occurrences de v.
  Par conséquent, v est majoritaire.

Un jeu entre algorithme et environnement

Soit $\mathcal{A}$ un algorithme de recherche de valeur majoritaire.

On introduit un environnement $\mathcal{E}$ qui répond aux comparaisons de T
au fur et à mesure de l'exécution de $\mathcal{A}$ pour maximiser le temps d'exécution de $\mathcal{A}$.

La stratégie de $\mathcal{E}$ repose sur une structure appelée amphithéâtre
dans lequel les indices du tableau sont répartis entre les gradins et l'arène
qui contient des troupeaux et des binômes.

- Lorsque i est dans les gradins alors $\forall j \neq i\ T[i] \neq T[j]$;
- Un binôme $\{i, j\}$ vérifie $T[i] \neq T[j]$;
- Lorsque i et j appartiennent à un même troupeau alors $T[i] = T[j]$.

Initialement, tous les indices sont dans l'arène
et constituent des troupeaux singletons.

g est le nombre d'indices des gradins, B est le nombre de binômes,
Tr est le nombre de troupeaux et t est le nombre d'indices des troupeaux.
$d = B + t$ et $m = \lfloor \frac{n}{2} \rfloor + 1$, le seuil de majorité.

La stratégie de l'environnement

On démontrera par induction qu'avec la stratégie de $\mathcal{E}$,
d ne croît jamais, $d \geq m$
et $d > m$ implique que tous les troupeaux sont des singletons.

En réponse à un test $T[i] \stackrel{?}{=} T[j]$, la stratégie de $\mathcal{E}$ est la suivante :

- Si i ou j est dans les gradins alors $T[i] \neq T[j]$;
- Sinon si $\{i, j\}$ est un binôme alors $T[i] \neq T[j]$;
- Sinon si i et j sont dans un même troupeau alors $T[i] = T[j]$;
- Sinon si i ou j appartiennent à un binôme alors $T[i] \neq T[j]$ et :
  - S'il existe un binôme $\{i, j'\}$ avec $j \neq j'$,
    i va dans les gradins et j' forme un troupeau singleton ;
  - Sinon il existe un binôme $\{i', j\}$ avec $i \neq i'$ alors
    j va dans les gradins et i' forme un troupeau singleton ;
- Sinon si i et j sont dans des troupeaux différents
  - Si $d > m$ alors $T[i] \neq T[j]$ et $\{i, j\}$ forme un binôme ;
  - Sinon $T[i] = T[j]$ et les troupeaux fusionnent.

Propriétés de la stratégie (1)

Evolution de d.

Lorsque un binôme est défait un troupeau singleton se forme donc d est inchangé.

Lorsque $d > m$ et qu'un binôme se forme d décroît d'une unité.

Tant que $d > m$, il n'y a pas de fusion de troupeaux,
donc tous les troupeaux sont des singletons.

Lorsque $d = m$ et que deux troupeaux fusionnent d est inchangé.

Possibilité d'une valeur majoritaire.

Puisque $d \geq m$,

- on choisit une valeur commune v pour tous les troupeaux ;
- en choisit v pour l'un des éléments du binôme ;
- on choisit des valeurs uniques pour les éléments restants.

v est une valeur majoritaire pour T
et les résultats des comparaisons effectuées sont cohérents avec T.

Propriétés de la stratégie (2)

Possibilité de l'absence de valeur majoritaire.

Tant que l'arène n'est pas constituée d'un troupeau de taille m

- on choisit une valeur spécifique v pour chaque troupeau ;
- on choisit des valeurs uniques pour les éléments restants.

Aucune valeur n'est majoritaire pour T

et les résultats des comparaisons effectuées sont cohérents avec T.

Il y a au moins $2g + B$ comparaisons négatives.

C'est initialement vrai : $g = B = 0$. Cette quantité est modifiée ainsi.

- Lors d'une comparaison négative débouchant sur la constitution d'un binôme, g est inchangé et B est incrémenté ;
- Lors d'une comparaison négative défaisant un binôme, g est incrémenté et B est décrémenté.

Propriétés de la stratégie (3)

Il y a au moins $t - Tr$ comparaisons positives.

C'est initialement vrai : $t - Tr = 0$. Cette quantité est modifiée ainsi.

- Lors d'une comparaison négative débouchant sur la constitution d'un binôme, t et Tr sont décrémentés de 2 ;
- Lors d'une comparaison négative défaisant un binôme, t et Tr sont incrémentés ;
- Lors d'une comparaison positive fusionnant deux troupeaux, t est inchangé et Tr est décrémenté ;

Lorsque l'arène est constituée d'un troupeau de taille m, il y au moins $\lfloor \frac{n}{2} \rfloor + 2(n - \lfloor \frac{n}{2} \rfloor - 1) = n + (n - \lfloor \frac{n}{2} \rfloor) - 2 = n + \lceil \frac{n}{2} \rceil - 2$ comparaisons.

$\boxed{\text{L'algorithme est donc optimal.}}$

Une généralisation

Soit T un tableau de taille n.

Une valeur v est k-*majoritaire* si $|\{i \mid T[i] = v\}| > \frac{n}{k}$.

Le problème de la valeur k-majoritaire consiste à renvoyer l'ensemble des valeurs k-majoritaires.

Une valeur est 2-majoritaire si et seulement si elle est majoritaire.

On fait l'hypothèse que :

- l'espace des valeurs est ordonné ;
- on ne peut que comparer les valeurs.

Observation. Il y a au plus $k - 1$ valeurs k-majoritaires.

Calcul des valeurs k-majoritaires

Un premier algorithme en $O(n \log(n))$.

Sort(T, n) ;
$E \leftarrow \emptyset$; $v \leftarrow T[1]$; $occ \leftarrow 1$;
For i **from** 2 **to** n **do**
  If $T[i] = v$ **then**
    $occ \leftarrow occ + 1$;
  Else
    If $occ > \frac{n}{k}$ **then** $\mathbf{Add}(E, v)$
    $occ \leftarrow 1$; $v \leftarrow T[i]$;
If $occ > \frac{n}{k}$ **then** $\mathbf{Add}(E, v)$

Principe d'un algorithme plus efficace.

- L'algorithme sélectionne au plus $k - 1$ valeurs susceptibles d'être k-majoritaires ;
- Il vérifie ensuite cette hypothèse simultanément pour ces valeurs.

AVL d'occurrences

Dans un AVL étendu $\mathcal{A}$,

- $\mathcal{A}.size$ contient le nombre de sommets ;
- chaque sommet de $\mathcal{A}$ contient une paire (clé, nombre d'occurrences) ;
- et les primitives prennent en compte cet aspect (sauf $\mathbf{Seek}(\mathcal{A}, key)$).

$\mathbf{Insert}(\mathcal{A}, key)$ en $O(\log(n))$ où n est le nombre de paires de $\mathcal{A}$

- insère la paire $(key, 1)$ si key n'est pas présente et met à jour $\mathcal{A}.size$
- incrémente le nombre d'occurrences sinon.

$\mathbf{Delete}(\mathcal{A}, key)$ en $O(\log(n))$

- décrémente le nombre d'occurrences si key est présente
- et supprime la paire si ce nombre devient nul et met à jour $\mathcal{A}.size$.

$\mathbf{Reduce}(\mathcal{A})$ en $O(n\log(n))$

- décrémente le nombre d'occurrences de toutes les clés,
- supprime les paires dont le nombre devient nul et met à jour $\mathcal{A}.size$.

$\mathbf{Reset}(\mathcal{A})$ en $O(n)$ remet à zéro les occurrences des clés de $\mathcal{A}$ sans les supprimer.

Recherche de valeurs k-majoritaires

For i **from** 1 **to** n **do**
  $\mathbf{Insert}(\mathcal{A}, T[i])$;
  If $\mathcal{A}.size = k$ **then** $\mathbf{Reduce}(\mathcal{A})$;
$\mathbf{Reset}(\mathcal{A})$;
For i **from** 1 **to** n **do**
  If $\mathbf{Seek}(\mathcal{A}, T[i])$ **then** $\mathbf{Insert}(\mathcal{A}, T[i])$;
$E \leftarrow \emptyset$; $\mathbf{Select}(\mathcal{A}, k)$
return(E) ;

$\mathbf{Select}(\mathcal{A}, k)$

If $\mathcal{A} \neq \texttt{NULL}$ **then**
  If $\mathcal{A}.occ \geq k$ **then** $\mathbf{Add}(E, \mathcal{A}.key)$
  $\mathbf{Select}(\mathcal{A}.fg, k)$; $\mathbf{Select}(\mathcal{A}.fd, k)$

Complexité et correction

Complexité.

- $\mathcal{A}.size$ ne dépasse jamais k.
- $\mathbf{Reduce}$ est appelée au plus $\lfloor \frac{n}{k} \rfloor$ fois
  d'où une complexité cumulée en $O(\lfloor \frac{n}{k} \rfloor k \log(k)) = O(n \log(k))$.

La complexité de cet algorithme est en $O(n \log(k))$ car :

- Les deux boucles $\mathbf{For}$ (sans les appels à $\mathbf{Reduce}$)
  ont une complexité en $O(n \log(k))$;
- $\mathbf{Reset}$ et $\mathbf{Select}$ ont une complexité en $O(k)$.

Correction.

Puisque $\mathbf{Reduce}$ est appelée au plus $\lfloor \frac{n}{k} \rfloor$ fois,
au plus $\lfloor \frac{n}{k} \rfloor$ occurrences d'une valeur sont supprimées.

Une valeur k-majoritaire est donc présente dans $\mathcal{A}$ à l'issue de la première étape.

Plan

Recherche de la $k$ième valeur d'un tableau

Recherche d'une valeur majoritaire

3. Une fusion en place

Rotation efficace en place

$\textbf{Rotate}(T, i, j, d)$ effectue une rotation d'amplitude d sur le sous-tableau $T[i, j]$.

Illustration. $\textbf{Rotate}(T, 1, 7, 4)$: $[1, 2, 3, 4, 5, 6, 7] \to [4, 5, 6, 7, 1, 2, 3]$

Une procédure en $\Theta(j - i + 1)$.

$\textbf{Rotate}(T, i, j, d)$

$d \leftarrow d \mod (j - i + 1)$

For k **from** 1 **to** $\lfloor \frac{j-i+1}{2} \rfloor$ **do** $\textbf{Swap}(T, i + k - 1, j - k + 1)$;

For k **from** 1 **to** $\lfloor \frac{j-i-d+1}{2} \rfloor$ **do** $\textbf{Swap}(T, j - d + k, j - k + 1)$;

For k **from** 1 **to** $\lfloor \frac{d}{2} \rfloor$ **do** $\textbf{Swap}(T, i + k - 1, i + d - k)$;

Illustration. $\textbf{Rotate}(T, 1, 7, 4)$

$$[1, 2, 3, 4, 5, 6, 7] \to [7, 6, 5, 4, 3, 2, 1] \to [7, 6, 5, 4, 1, 2, 3] \to [4, 5, 6, 7, 1, 2, 3]$$

Fusion : un cas particulier

La fusion a pour entrée $T[1, m]$ et $T[m+1, n]$ (triés). On note $s = \lfloor \sqrt{n} \rfloor$.

Hypothèse : $m \leq s$.

LeftShortMerge(T, m, n) $(m \leq s)$
  $i \leftarrow 1$; $j \leftarrow m+1$; $\ell \leftarrow m$
  While $\ell > 0$ **and** $j \leq n$ **do**
    While $j \leq n$ **and** $T[j] < T[i]$ **do** $j \leftarrow j+1$;
    Rotate$(T, i, j-1, -\ell)$;
    $\ell \leftarrow \ell - 1$; $i \leftarrow j - \ell$

Complexité.

Les m éléments du petit tableau sont testés et déplacés au plus m fois

Les $n-m$ éléments du grand tableau
sont déplacés par **Rotate** au plus une fois.

Rotate opère en temps linéaire. D'où une complexité en $O(n)$.

Illustration

- $i = 1$, $j = 3$, $\ell = 2$

$$[BD][AACEGH]$$

- Après la mise à jour de j, $j = 5$, d'où une rotation de 2.

$$[AAB][D][CEGH]$$

- $i = 4$, $j = 5$, $\ell = 1$

- Après la mise à jour de j, $j = 6$, d'où une rotation de 1.

$$[AABCD][EGH]$$

- $\ell = 0$, la fusion est achevée.

Fusion : un autre cas particulier

Hypothèse : $n - m \leq s$.

$\mathbf{RightShortMerge}(T, m, n)$ $(n - m \leq s)$
 $i \leftarrow m$; $j \leftarrow n$; $\ell \leftarrow n - m$
 While $\ell > 0$ **and** $i \geq 1$ **do**
  While $i \geq 1$ **and** $T[j] < T[i]$ **do** $i \leftarrow i - 1$;
  $\mathbf{Rotate}(T, i+1, j, \ell)$;
  $\ell \leftarrow \ell - 1$; $j \leftarrow i + \ell$

- $i = 6$, $j = 8$, $\ell = 2$

$$[AACEGH][BD]$$

- Après la mise à jour de i, $i = 3$, et une rotation de 2.

$$[AAC][B][DEGH]$$

- $i = 3$, $j = 4$, $\ell = 1$
- Après la mise à jour de i, $i = 2$, et une rotation de 1.

$$[AA][BCDEGH]$$

Fusion : la première étape

Hypothèse $m > s$ **et** $n - m > s$**.**

Constitution d'un « buffer » des s plus grandes valeurs entre les deux séquences.

$i_1 \leftarrow m$; $i_2 \leftarrow n$; $s_2 \leftarrow 0$;
For i **from** 1 **to** s **do**
 If $T[i_1] > T[i_2]$ **then** $i_1 \leftarrow i_1 - 1$; **else** $i_2 \leftarrow i_2 - 1$; $s_2 \leftarrow s_2 + 1$;
For i **from** 0 **to** $s_2 - 1$ **do** **Swap**$(T, i_1 - i, n - i)$

Illustration. $n = 16$, $s = 4$

$$[AAGMNOPQRS][BCDEGZ] \to [AAGMNO][ZQRS][BCDEG][P]$$

Soit $b = \lfloor \frac{i_2 - m}{s} \rfloor$. Décomposition de la deuxième séquence $T[m+1, i_2]$
en $T[m+1, m+bs]T[m+bs+1, i_2]$,
une séquence de b s-*blocs* et une séquence de taille inférieure à s.

Illustration.

$$[AAGMNO][ZQRS][BCDEG][P] = [AAGMNO][ZQRS][BCDE][G][P]$$

$T[m+bs+1, i_2]$ et $T[i_2+1, n]$ seront fusionnées en $O(n)$ par **RightShortMerge** à la fin de l'algorithme et seront omis dans la suite.

Fusion : la deuxième étape (1)

A la fin de la première étape, le tableau se présente ainsi :

- $T[1, m]$, la première séquence à fusionner ;
- $T[m+1, m+s]$, le buffer composé des s plus grandes valeurs ;
- $T[m+s+1, n]$, la deuxième séquence à fusionner composée de s-blocs. (i.e., $n-m$ multiple de s)

On note $t = m\%s$.

- On échange $T[1, t]$ et $T[m+s-t+1, m+s]$;
- puis on fusionne $T[m+s-t+1, m+s]$ et $T[m+s+1, m+2s]$ en $O(n)$ par **RightShortMerge** (ou **LeftShortMerge**).

Illustration. $s = 4$, $m = 10$, $t = 2$

$$[AGGMNOPPPP][ZQRS][BCDE][FFFF][\ldots]$$
$$\rightarrow [RS][GMNOPPPP][ZQ][AG][BCDE][FFFF][\ldots]$$
$$\rightarrow [RS][GMNOPPPP][ZQ][ABCDEG][FFFF][\ldots]$$

Fusion : la deuxième étape (2)

$T[m+s-t+1, m+s]$ contient alors les t plus petites valeurs.

- On échange $T[m+s-t+1, m+s]$ avec $T[1,t]$;
- puis on échange le buffer (reformé) $T[m+1, m+s]$ avec $T[t+1, t+s]$.

Illustration.

$$\begin{aligned}
&\to [RS][GMNOPPPP][ZQ][ABCDEG][FFFF][\ldots]\\
&\to [AB][\underline{G}MNO][PPPP][ZQRS][CDE\underline{G}][FFFF][\ldots]\\
&\to [AB][ZQRS][PPPP][GMNO][CDEG][FFFF][\ldots]\\
&= [AB][ZQRS][PPPP][GMNO][CDEG][FFFF][\ldots]
\end{aligned}$$

Observations.

$T[1,t]$ contient donc les t plus petites valeurs qu'on omet dans la suite.

$T[m+s+1, m+2s]$ peut être vu comme le premier bloc (déplacé) de la séquence auquel appartient $T[m+2s]$.

Illustration. $[CDEG]$ est le premier bloc de la première séquence.

Fusion : la troisième étape

A la fin de la deuxième étape, le tableau se présente ainsi :

- n est un multiple de s, $n = bs$ avec $s - 2 \leq b \leq s + 1$;
- $T[1, s]$, le buffer composé des s plus grandes valeurs ;
- une séquence de $b - 1$ blocs triés $T[is + 1, (i + 1)s]$ pour $1 \leq i < b$ issus de deux séquences triées.

Illustration. $n = 25$, $b = s = 5$

$$T = [H_2 H_3 I_1 I_2 J_1][B_1 B_2 B_3 D_1 D_2][E_1 E_2 F_1 G_1 H_1][A_1 A_2 A_3 B_4 B_5][C_1 C_2 E_3 G_2 G_3]$$

Tri des blocs

Tri (par sélection) des blocs selon l'ordre de leur plus grand élément.

Ce tri opère en $O(b(b+s)) = O(n)$.

```
For i from 2 to b − 1 do
  k ← i ;
  For j from  i + 1 to b do
    If T[j ∗ s] < T[k ∗ s] then k ← j ;
  For j from 0 to s − 1 do Swap(i ∗ s − j, k ∗ s − j) ;
```

Illustration.

$$T = [H_2 H_3 I_1 I_2 J_1][B_1 B_2 B_3 D_1 D_2][E_1 E_2 F_1 G_1 H_1][A_1 A_2 A_3 B_4 B_5][C_1 C_2 E_3 G_2 G_3]$$

$$T' = [H_2 H_3 I_1 I_2 J_1][A_1 A_2 A_3 B_4 \overline{B}_5][B_1 B_2 B_3 D_1 \overline{D}_2][C_1 C_2 E_3 G_2 \overline{G}_3][E_1 E_2 F_1 G_1 \overline{H}_1]$$

Fusion : la quatrième étape

L'algorithme de la quatrième étape maintient

- i_1 début d'une séquence à fusionner qui suit le buffer $T[i_1, i_2 - 1]$
- i_2 début du bloc qui suit $T[i_2, i_2 + b - 1]$
  à fusionner avec la séquence si nécessaire ;

Après une fusion,

- la séquence à fusionner contient (initialement)
  les éléments du bloc précédent supérieurs ou égaux à la précédente séquence
- qui ne sont donc pas déplacés par la fusion.

Lorsque la séquence n'est plus suivie par un bloc

- l'algorithme échange la séquence avec le buffer par une rotation.
- et trie en place le buffer.

Illustration

Buffer rouge, séquence bleue et bloc gris

$$[H_2 H_3 I_1 I_2 J_1][A_1 A_2 A_3 B_4 B_5][B_1 B_2 B_3 D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Extension de la séquence

$$[H_2 H_3 I_1 I_2 J_1][A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Après la première fusion

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2 D_1 D_2][I_1 H_2 H_3 I_2 J_1][E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Après la deuxième fusion

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2 D_1 D_2 E_3 E_1 E_2 F_1 G_2 G_3][I_1 H_2 H_3 I_2 J_1][G_1 H_1]$$

Après l'échange de la séquence et du buffer

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2 D_1 D_2 E_3 E_1 E_2 F_1 G_2 G_3 G_1 H_1][I_1 H_2 H_3 I_2 J_1]$$

Après le tri du buffer

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2 D_1 D_2 E_3 E_1 E_2 F_1 G_2 G_3 G_1 H_1 H_2 H_3 I_2 I_1 J_1]$$

Algorithme de la quatrième étape

$i_1 \leftarrow s+1$; $i_2 \leftarrow 2*s+1$;
While $i_2 \leq n$ **do**
  (Extension de la séquence)
  While $i_2 \leq n$ **and** $T[i_2-1] \leq T[i_2]$ **do** $i_2 \leftarrow i_2+s$;
  If $i_2 \leq n$ **then**
    (Fusion de la séquence et du bloc)
    $\mathbf{Buffermerge}(T, i_1, i_2)$; (i_1 et i_2 sont modifiés par $\mathbf{Buffermerge}$)
(Echange de la séquence et du buffer)
$\mathbf{Rotate}(T, i_1-s, n, n-i_1+1)$;
(Tri en place du buffer)
$\mathbf{Heapsort}(T, n-s+1, n)$;

Buffermerge

Durant le **Buffermerge**,

- le buffer se déplace et se scinde
- mais se regroupe derrière la nouvelle séquence à la fin du merge

Buffermerge(T, i_1, i_2)

$newi_2 \leftarrow i_2 + s$; $i \leftarrow i_1 - s$; $n_1 \leftarrow i_2 - i_1$;

While $n_1 > 0$ **do**

  If $T[i_1] \leq T[i_2]$ **then**

    Swap(T, i_1, i) ; $i \leftarrow i + 1$; $i_1 \leftarrow i_1 + 1$; $n_1 \leftarrow n_1 - 1$;

  Else

    Swap(T, i_2, i) ; $i \leftarrow i + 1$; $i_2 \leftarrow i_2 + 1$;

$i_1 \leftarrow i_2$; $i_2 \leftarrow newi_2$;

La complexité de **Buffermerge** est en $O(i_2 - i_1)$

Illustration

$$[H_2 H_3 I_1 I_2 J_1][A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Insertion d'éléments de la séquence

$$[A_1][H_3 I_1 I_2 J_1 H_2][A_2 A_3 B_4 B_5 B_1 B_2 B_3 D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

...

$$[A_1 A_2 A_3 B_4 B_5][H_2 H_3 I_1 I_2 J_1][B_1 B_2 B_3 D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

...

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3][I_2 J_1 H_2 H_3 I_1][D_1 D_2][C_1 C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Insertion d'éléments du bloc

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1][J_1 H_2 H_3 I_1][D_1 D_2][I_2][C_2 E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2][H_2 H_3 I_1][D_1 D_2][I_2 J_1][E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Insertion d'éléments de la séquence

$$[A_1 A_2 A_3 B_4 B_5 B_1 B_2 B_3 C_1 C_2 D_1 D_2][I_1 H_2 H_3 I_2 J_1][E_3 G_2 G_3][E_1 E_2 F_1 G_1 H_1]$$

Complexité de la quatrième étape

Observation.

Un élément d'une séquence est déplacé au plus une fois

lors des **Buffermerge** successifs.

D'où une complexité en $O(n)$ pour la phase des fusions.

Le tri en place du buffer se fait en $O(s \log(s)) \subseteq O(n)$.

Correction de l'algorithme (1)

On note A et B les séquences à fusionner. Pour $X \in \{A, B\}$,

on note $\overline{X}$ l'autre séquence et $M_{X,i}$ le plus grand élément du $i$ème bloc de X.

Observation. La première (éventuelle) fusion a lieu lorsque $z < M_{X,i}$
où z est le plus petit élément d'un $j$ème bloc de $\overline{X}$.
Le tri des blocs implique $M_{X,i} \leq M_{\overline{X},j}$.

Après la fusion, $T[i_1, i_2 - 1]$
contient les éléments x du $j$ème bloc de $\overline{X}$ tels que $M_{X,i} \leq x \leq M_{\overline{X},j}$
et $T[1, i_1 - s - 1]T[i_1, i_2 - 1]$ est la séquence triée des blocs examinés.

Invariant de boucle.
Soit i le dernier bloc examiné de A et j le dernier bloc examiné de B.

- $T[i_1 - s - 1]$ contient le buffer ;
- $T[1, i_1 - s - 1]T[i_1, i_2 - 1]$ est la séquence triée des blocs examinés ;
- $T[i_1 - s - 1] \leq \min(M_{A,i}, M_{B,j})$.

Correction de l'algorithme (2)

Preuve de l'invariant.

Déjà démontré après la première fusion.

Considérons l'examen du prochain bloc.

• S'il n'y a pas de fusion, la concaténation reste triée

et les conditions sur $T[i_1 - s - 1]$ et le buffer restent valides.

• S'il y a une fusion, puisque $T[i_1 - s - 1] = \min(M_{A,i}, M_{B,j})$,

la fusion de l'algorithme produit la séquence triée

et le buffer se retrouve en $T[i_1 - s, i_1 - 1]$.

Supposons que le $j + 1$ème bloc de B provoque la fusion

alors $T[i_2 - 1] = M_{A,i}$ pour le dernier $i$ème bloc de A examiné.

Après la fusion les éléments du $j + 1$ème bloc de B supérieurs ou égaux à $M_{A,i}$

se trouvent à des indices supérieurs ou égaux à i_1,

On a donc $T[i_1 - s - 1] = M_{A,i} \leq M_{B,j+1}$.