Algorithmique & programmation

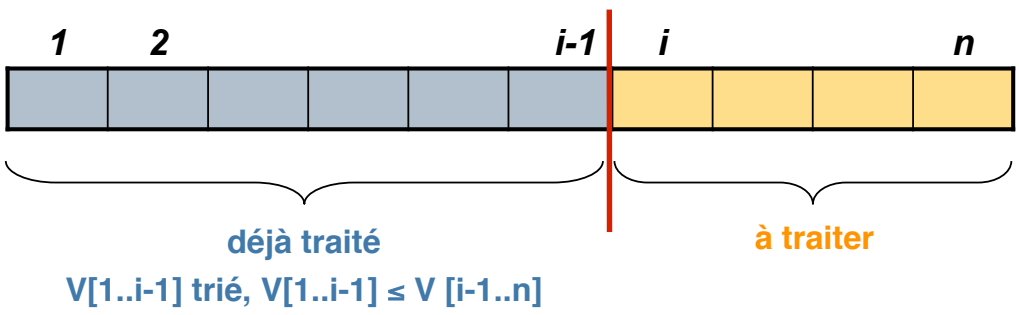
Chapitre 2 : Vecteurs

Algorithme de tri à bulles

Tri par la méthode des bulles

- Tri interne meilleur que **tripermut**

- Construction de l'hypothèse
  - On est en train de trier le vecteur
  - Donc on va partir de l'hypothèse que l'on a déjà trié une partie du vecteur



Tri par la méthode des bulles

- Idée de l'algorithme
  - Dans la partie du vecteur non traitée (V[i..n])
    - **Mettre le plus petit à sa place**
    - **Mettre en même temps de l'ordre dans V[i+1..n]**
  - Ainsi
    - V[1..i] sera trié et **V[i+1..n] sera mieux ordonné**
  - Poursuivre jusqu'à ce que le vecteur non traité devienne vide

Tri par la méthode des bulles

- Dans V[i..n], **comment** ?
  - Mettre le plus petit à sa place
  - Mettre en même temps de l'ordre dans V[i+1..n]
- Idée
  - Examiner deux éléments successifs de V[i..n]
    - Si V[j - 1] ≤ V[j]
      - alors V[j - 1] et V[j] ordonnés relativement
    - Si V[j - 1] > V[j]
      - alors V[j - 1] et V[j] non ordonnés relativement
      - il faut les permuter
  - Faire cela pour trous les éléments successifs de V[i..n]

Tri par la méthode des bulles

- C'est la remontée des bulles à partir de n
  - Les plus petits vont se rapprocher de i
  - Les plus grands vont se rapprocher de n
  - Le plus petit occupera la place i
- L'ordre général s'accroît
- Exemple
  - Avant

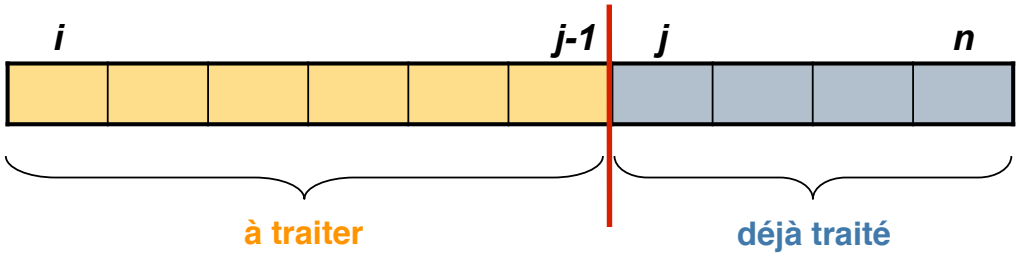
i							*n*
7	14	6	3	12	15	8	15

  - Après

i							*n*
3	7	14	6	8	12	15	15

Remontée des bulles

- Construction de l'hypothèse



- V[j] ≤ V[j..n]
- V [i..j-1] non traité

Remontée des bulles

- Raisonnement par récurrence

Hypothèse V[j] ≤ V[j..n], V[i..j-1] non traité

➤ j = i ⇨ *V[i] ≤ V[i..n]* ✳

➤ j > i

➤➤ V[j-1] ≤ V[j] ⇨ j := j - 1 ; ➡ ***H***

➤➤ V[j-1] > V[j] ⇨ permuter V[j-1] et V[j];
j := j - 1 ; ➡ ***H***

Itération tantque (j > i) faire …

Initialisation j := n ; ➡ ***H***

algorithme de remontée des bulles

```
j := n ;                                  {V[j] ≤ V[j..n]}
tantque j > i faire
    si V[j-1] > V[j] alors
        permut (V[j], V[j-1]) ;           {V[j-1] ≤ V[j..n]}
    finsi ;
    j := j - 1 ;                          {V[j] ≤ V[j..n]}
finfaire ;                                {(j = i, V[j] ≤ V[j..n]) ➔ V[i] ≤ V[i..n]}
```

ou encore :

```
pour j := n bas i + 1 faire
    si V[j-1] > V[j] alors
        permut (V[j], V[j-1]) ;
    finsi ;
finfaire;
```

Tri par la méthode des bulles (Version 1)

- Raisonnement par récurrence

Hypothèse V[1..i-1] ≤ V[i..n] , V[1..i-1] trié, V[i..n] non traité

➤ i = n ⇨ *V[1..n-1] trié, V[1..n-1] ≤ V[n]*
⇨ V[1..n] trié ✳

➤ i < n ⇨ permutations ("bulles") afin que le minimum de V[i..n] soit placé en V[i] ; puis i := i + 1 ; ➡ ***H***

Itération tantque (i < n) faire … *(pour …)*

Initialisation i := 1 ; ➡ ***H***

procédure **tribulle1**

```
procédure tribulle1 (dr V[1..n] : vecteur) ;
spécification {n > 0} ➔ {V[1..n] trié}
debproc
    pour i := 1 haut n - 1 faire
    {i croît de 1 à n - 1 pour faire le tri}
        pour j := n bas i + 1 faire
        {j décroît de n à i + 1 pour faire remonter la bulle}
            si V[j-1] > V[j] alors
                    permut (V[j-1], V[j]) ;
            finsi ;
        finfaire ;
    finfaire ;
finproc ;
```

procédure **tribulle1**

```
procedure tribulle1 (V[1..n] : in out vecteur) is
--spécification {n > 0} ➔ {V[1..n] trié}
begin
  for i in V'range loop
  --i croît de 1 à n - 1 pour le tri
    for j in reverse V'last .. i + 1 loop
    --j décroît de n à i + 1 pour remonter la bulle
      if V(j-1) > V(j) then
        permut (V[j-1], V[j]) ;
      end if ;
    end loop ;
  end loop ;
end tribulle1 ;
```


Coût de **tribulle1**

- Comparaisons
  - 1 comparaison à chaque itération (n - i)
    - pour i = 1 : n - 1 comparaisons,
    - pour i = 2 : n - 2 comparaisons,
    - …
    - pour i = n-1 : 1 comparaison.

  = n (n - 1) / 2 (n (n - 1) accès)
- Permutations
  - cas favorable (V trié) : 0 permuation
  - cas défavorable (une permutation à chaque comparaison) : n (n - 1) / 2 permutations (2n (n - 1) accès)

Coût de **tribulle1**

- Pour résumer

place occupée		(n + 1)t	
nombre de comparaisons		n(n - 1)/2	(≈ n^2 accès)
nombre d'affectations	cas favorable	0	
	cas défavorable	n(n - 1)/2	(≈ $2n^2$ accès)

plus grand que pour **tripermut** (n-1)

- Exemple
  - Si n = 1000 et t = 20 octets
  - Place occupée : 20 020 octets
  - nb de comparaisons : **500 000**
  - nb d'affectations : **250 000** en moyenne

Amélioration de **tribulle1**

- Remarque
  - Après avoir traité i - 1 (1 ≤ i < n) éléments, on a :
  - V[1..i-1] trié, V[1..i-1] ≤ V[i..n], V[i..n] non traité
- Or, si
  - V[i..n] est trié à la suite des permutations effectuées, il est inutile de continuer car :
  - V[1..i-1] trié, V[1..i-1] ≤ V[i], V[i..n] trié ➔ V[1..n] trié

Q : Comment savoir si V[i..n] est trié ?

R : Si on n'a pas fait de permutation pendant la remontée des bulles, V[i..n] est trié !

Amélioration de **tribulle1**

- Utilisation d'une variable booléenne
  - **onapermuté**
    - avant la remonté des bulles : initialisée à ***faux***
    - après la remonté des bulles :
      - ***vrai*** si au moins une permutation
      - ***faux*** si aucune permutation
- tantque i < n (pour i = 1 haut n - 1) devient :
  - tantque i < n et **onapermuté**

Amélioration de **tribulle1**

- tantque i < n et **onapermuté**
  - le test (i < n) est-il encore nécessaire ?
  - Dans le pire des cas
    - une permutation à chaque remonté
  - On atteindra toujours i = n
    - il n'y aura pas de permutation donc onapermuté restera faux
- tantque **onapermuté** est suffisante

procédure **tribulle2**

procédure tribulle2 (dr V[1..n] : vecteur) ;
spécification *{n > 0}* ➔ *{V[1..n] trié}*
    i : entier ; onapermuté : booléen ;
debproc
    i := 1 ;
    onapermuté := vrai ;
    tantque onapermuté **faire**
        onapermuté := faux ;
        pour j := n **bas** i + 1 **faire**
            si V[j-1] > V[j] **alors**
                permut (V[j-1], V[j]) ;
                onapermuté := vrai ;
            finsi ;
        finfaire ;
        i := i + 1 ;
    finfaire ; *{V[1..n] trié}*
finproc ;

procédure **tribulle2**

```
procedure tribulle2 (V : in out vecteur) is
--spécification {n > 0} ➔ {V[1..n] trié}
   i : integer ; onapermuté : boolean ;
begin
   i := V'First ;
   onapermuté := true ;
   while onapermuté loop
     onapermuté := false ;
     for j in reverse V'Last .. i + 1 loop
       if V(j-1) > V(j) then
         permut (V(j-1), V(j)) ;
         onapermuté := true ;
       end if ;
     end loop;
     i := i + 1 ;
   end loop ;              --V[1..n] trié
end tribulle2 ;
```

Coût de **tribulle2**

- **Toujours utiliser l'algorithme optimisé** si on veut faire un tri à bulles
- Performances moyennes inférieures à celles du tri par permutation
- Un tri classique simple à écrire