Correction TD 5

Pascal Vanier

April 24, 2013

Calcul du nombre d'inversions :

(1) La méthode brutale, consiste, pour chaque position i à vérifier si il combien il y a de positions j plus loin qui contiennent un entier plus petit :

```
Inversions_bete(tableau s)
   nb_inversions := 0
   pour i=0 jusqu'à taille(s)-1 faire
     pour j=i+1 jusqu'à taille(s) faire
       si s[j] < s[i], alors
          nb_inversions := nb_inversions+1
   retourner nb_inversions
```

Cette méthode fonctionne en temps $\mathcal{O}(n^2)$.

(2) On peut faire mieux en utilisant une approche du type diviser pour régner : on découpe la suite en deux sous-suites de manière récursive, que l'on fusionne en triant et en comptant le nombre d'inversions. Lors de la fusion il faudra bien faire attention à conserver à la fin une suite triée et à compter le nombre d'inversions. Imaginons que nous en soyons à l'étape de fusion pour les deux listes suivantes :

3	7	10	14	18	19

2	11	16	17	23	25

Lors de leur fusion, il faut compter le nombre d'inversions entre ces deux suites et l'additionner au d'inversions qu'il y avait déjà dans la sous-suite avant qu'elle ne soit elle-même triée. Pour faire cela, on peut remarquer qu'à chaque fois que l'on insère un élément de la liste de droite dans la liste finale avant un élément de la liste de gauche, cela signifie qu'il y avait une inversion. Ainsi, pour chaque élément de la liste de droite que l'on insère, il y a autant d'inversions avec lui qu'il y a d'éléments restants dans la liste de gauche. Pour l'exemple précédent, cela signifie par exemple que lorsque l'on insère 2 avant 3 dans la liste finale on sait qu'il y a 6 inversions (longueur de la liste de gauche).

Ayant remarqué cela, on peut maintenant écrire la fonction `Fusionner-et-compter(tableau s)` :

```
Fusionner-et-compter(tableaux A,B):
  inversions := 0
  a_ajouter := 0
  R := []
  tant que A!=[] et B!=[] faire
  --si A!=[] alors
  ----si B!=[] alors
  ------si tête(A) < tête(B) alors
  --------R := tete(A)::R
  --------inversions := inversions + a_ajouter
```

```
--------A := queue(A)
------si tête(B) < tête(A) alors
--------R := tête(B)::R
--------a_ajouter := a_ajouter+1
--------B := queue(B)
--si B!=[] alors
----R := tête(B)::R
----B := queue(B)
retourner (R,inversions)
```

Ici la variable `a_ajouter` contient le nombre d'éléments de la liste de droite qui ont déjà été insérés dans la liste résultante et donc le nombre d'inversions à ajouter au compte à chaque fois que l'on insère un élément de la liste de gauche par la suite.

Calculons maintenant la complexité de cet algorithme : $T(n) = T(n/2) + T(n/2) + n$ ce qui nous donne une complexité en $\mathcal{O}(n \log n)$.