Algorithmes de tri

Introduction : Les méthodes de tri sont très utilisées en pratique, en particulier en informatique de gestion où on trouve beaucoup d'applications qui consistent à trier des données. Elles sont également utilisées dans d'autres problèmes tels que les algorithmes sur les graphes. Plusieurs algorithmes de tri existent, d'où l'analyse et la comparaison de ceux-ci est très intéressante.
On suppose dans ce qui suit qu'on veut trier un tableau A d'entiers dans l'ordre croissant.

I- TRIS LENTS (Itératifs) :

I.1- Tri à bulle :

Principe :
1- On parcourt le tableau séquentiellement et on compare chaque élément i (i=1, ..., (n-1)) avec son suivant ; si l'ordre n'est pas bon, on permute ces deux éléments. Ainsi après à la première étape, l'élément maximum se retrouve en A[n] ;
2- On recommence l'étape (2) jusqu'à ce que l'on n'ait plus aucune permutation à effectuer (tableau trié).

Rmq : Le nom de tri à bulle vient donc de fait qu'à la fin de chaque itération, le plus grand élément se déplace comme une bulle de la gauche vers la droite.

```
Procédure TriBulle (var A : Tableau [1..n] d'entiers) ;
Variables trié : Booléen, i, tmp : entier ;
Début
        trié ←faux
        Tant que non trié
                trié←vrai ;
                Pour i ←1 à n-1 faire
                        Si A[i] > A[i+1] Alors
                                trié ←faux ;
                                tmp ←A[i] ; A[i] ←A[i+1] ; A[i+1] ←tmp ;
                        FinSi ;
                FinPour ;
        FinTQ ;
Fin ;
```

Complexité :
Meilleur cas (le tableau est déjà trié) :
- Nombre de comparaisons : n-1 ;
- Nombre de permutations : 0 ;
⇒ Complexité : O(n).

Pire cas (le tableau est trié dans l'ordre inverse) :
- Nombre de parcours : n (boucle pour) ;
- Nombre de comparaisons : n*(n-1) (chaque parcours n-1 comparaisons) ;
- Nombre de permutations : n*(n-1)/2 ; (1^{er} parcours : n-1 permutations, $2^{ème}$: n-2, ..., $n^{ème}$: 0) ;
⇒ Complexité : $O(n^2)$.

I.2- Tri par sélection :

Principe :
On compare chaque élément i (i=1, ..., (n-1)) à tous ses suivants ; à chaque fois que l'ordre n'est pas bon, on récupère l'indice de ce suivant j. A la fin de la comparaison, on compare i à j ; si i<>j alors on permute (A[i], A[i]). Ainsi à chaque étape k, on range le plus petit élément dans A[k].

Rmq : Le nom de tri par sélection vient donc de fait qu'à chaque étape k, on sélectionne et on range le plus petit élément dans A[k].

```
Procédure TriSélection (var A : Tableau [1..n] d'entiers) ;
Variables i, j, posMin, tmp : entier ;
Début
        Pour i ←1 à n-1 faire
                posMin ←i ;
                Pour j ←i + 1 à n faire
                        si A[j] < A[posMin] alors
                                posmin ←j ;
                        FfinSi ;
                finPour ;
                si posMin ≠ i Alors
                        temp ←A[posmMin] ; A[posMin] ←A[i] ; A[i] ←tmp ;
                FinSi ;
        finPour ;
Fin ;
```

Complexité :

Meilleur cas (le tableau est déjà trié) :

- Nombre de comparaisons : $(n-1)+(n-2)+...+1 = n*(n-1)/2$; ($1^{er}$ élét A[1] = n-1 compar, ..., A[n-1] = 1 compar)
- Nombre de permutations : 0 ;

⇒ Complexité : $O(n^2)$.

Pire cas (le tableau de la forme m, 1, 2, ..., m-1 : 5, 1, 2, 3, 4) :

- Nombre de comparaisons : $(n-1)+(n-2)+...+1 = n*(n-1)/2$; (même que MC)
- Nombre de permutations : n-1 (A chaque étape on permute) ;

⇒ Complexité : $O(n^2)$.

I.3- Tri par insertion :

Principe : On compare chaque élément i (i=2, ..., n) avec les éléments précédents ((i-1), ..., 1), déjà triés, jusqu'à trouver la place de l'élément i que l'on considère.

```
Procédure triInsertion (var A : Tableau [1..n] d'entiers) ;
Variables i, j, tmp : entier ;
Début
        pour i ←2 à n faire
                j ←i
                tantque (j > 1) et (A[j] < A[j-1])
                        tmp ←A[j] ; A[j] ←A[j-1] ;A[j-1] ←tmp
                        j ←j-1
                finTantque ;
        finPour ;
 fin ;
```

Complexité :

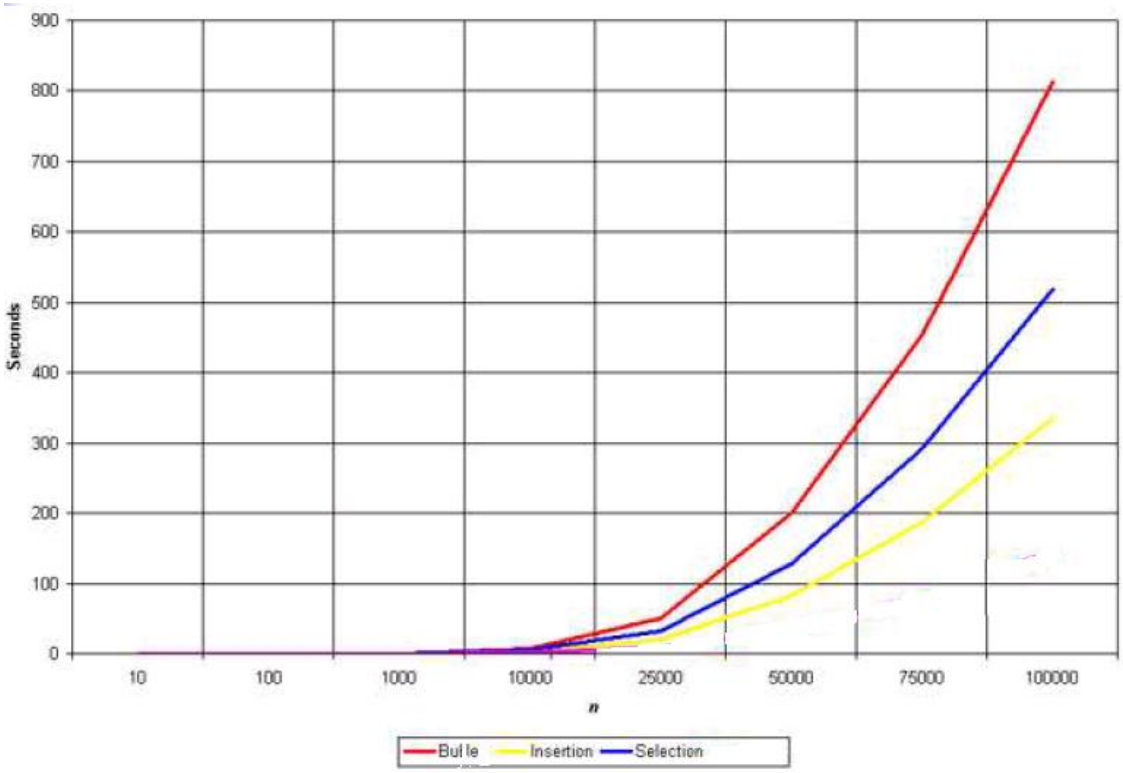
Meilleur cas (le tableau est déjà trié) :

- Nombre de comparaisons : n-1 ; (une pour chaque élément)
- Nombre de permutations : 0 ;

⇒ Complexité : $O(n^2)$.

Pire cas (le tableau trié dans l'ordre inverse) :
- Nombre de comparaisons : (n-1)+(n-2)+...+1 = n*(n-1)/2 ; (1er A[2] =1 compar, ..., A[n]=n-1 compar)
- Nombre de permutations : 1+ ... + n-1 = n*(n-1)/2; (chaque compar permutation)
⇒ Complexité : O(n²).

Comparaison des algorithmes de tris lents :



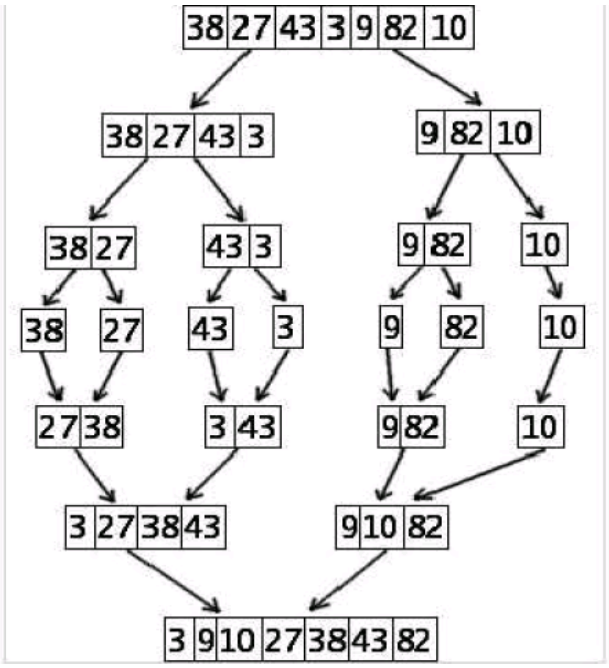
II- TRIS RAPIDES (Récursifs) : Il s'agit de tris suivant le paradigme diviser pour régner.
II.1- Tri par fusion : Il s'agit à nouveau d'un tri suivant le paradigme « diviser pour régner ».
Principe :
- On divise en deux moitiés le tableau (la liste) à trier ;
- On trie chacune d'entre elles ;
- On fusionne les deux moitiés obtenues pour reconstituer le tableau triée.

```
Procédure triFusion(var A : Tableau [1..n] d'entiers, début, fin : Eentier) ;
Variables milieu : entier
début
        si (début < fin) Alors      // au moins deux cases
                milieu ←(début + fin)/2 ;
                triFusion(A, début, milieu) ;
                triFusion(A, milieu+1, fin) ;
                fusionner(A,debut,milieu,fin) ;
        finSi ;
fin ;
```

```
Procédure Fusionner(A[n] : Tableau, début, milieu, fin : entier) ;
Variables i, i1, i2 : entier, tmp[1..n] : Tableau ;
début
        i ←0 ;
        i1 ←début ;
        i2 ←milieu + 1 ;
```

```
        Tantque (i1 <= milieu) et (i2 <= fin) faire
                si A[i1] < A[i2] Alors
                        tmp[i] ←A[i1] ;
                        i1 ←i1 +1 ;
                Sinon
                        tmp[i] ←A[i2] ;
                        i2 ←i2 + 1
                finSi ;
                i ←i + 1 ;
        FinTantque ;
        Tantque i1 <= milieu faire
                tmp[i] ←A[i1] ;
                i ←i + 1 ; i1 ←i1 + 1 ;
        finTantque ;
        Tantque i2 <= fin faire
                Tmp[i] ←A[i2] ;
                i ←i + 1 ; i2 ←i2 + 1 ;
        finTantque ;
        A ← Tmp ;
fin ;
```

Exemple :



Complexité :

La fusion du tableau A nécessité le parcourir des deux sous tableaux (à gauche et à droite du milieu). On itère ensuite le processus de manière récursive sur les deux sous tableaux (à gauche et à droite du milieu), et ainsi de suite jusqu'à avoir des tableaux de taille 1 (log (n) fois).

c(n)=n+2*c(n/2)

=> complexité = log(n)*O(n)=O(n*log(n)).

II.2- Tri rapide :

Principe

1- Choisir un élément du tableau appelé pivot (1er élément : A[1]) et ordonner les élément du tableau de telle manière que :
* les éléments de la partie de gauche soient tous inférieurs ou égaux à ce pivot ;
* et ceux de la partie de droite soient tous supérieurs à ce pivot ;
2- Itérer d'une manière récursive ce procédé sur les deux parties (gauche et droit) ainsi crées jusqu'à obtenir des tableaux avec 1 seul case.

```
Procédure triRapide(var A : Tableau [1..n] d'entiers, début, fin : entier) ;
Variables indexPivot : entier ;
début
        si (début < fin) Alors   // au moins deux cases
                partitionner(A, début, fin, indexPivot) ;
                triRapide(A, début, indexPivot -1) ;
                triRapide(A, indexPivot +1, fin) ;
        finSi ;
fin ;
```

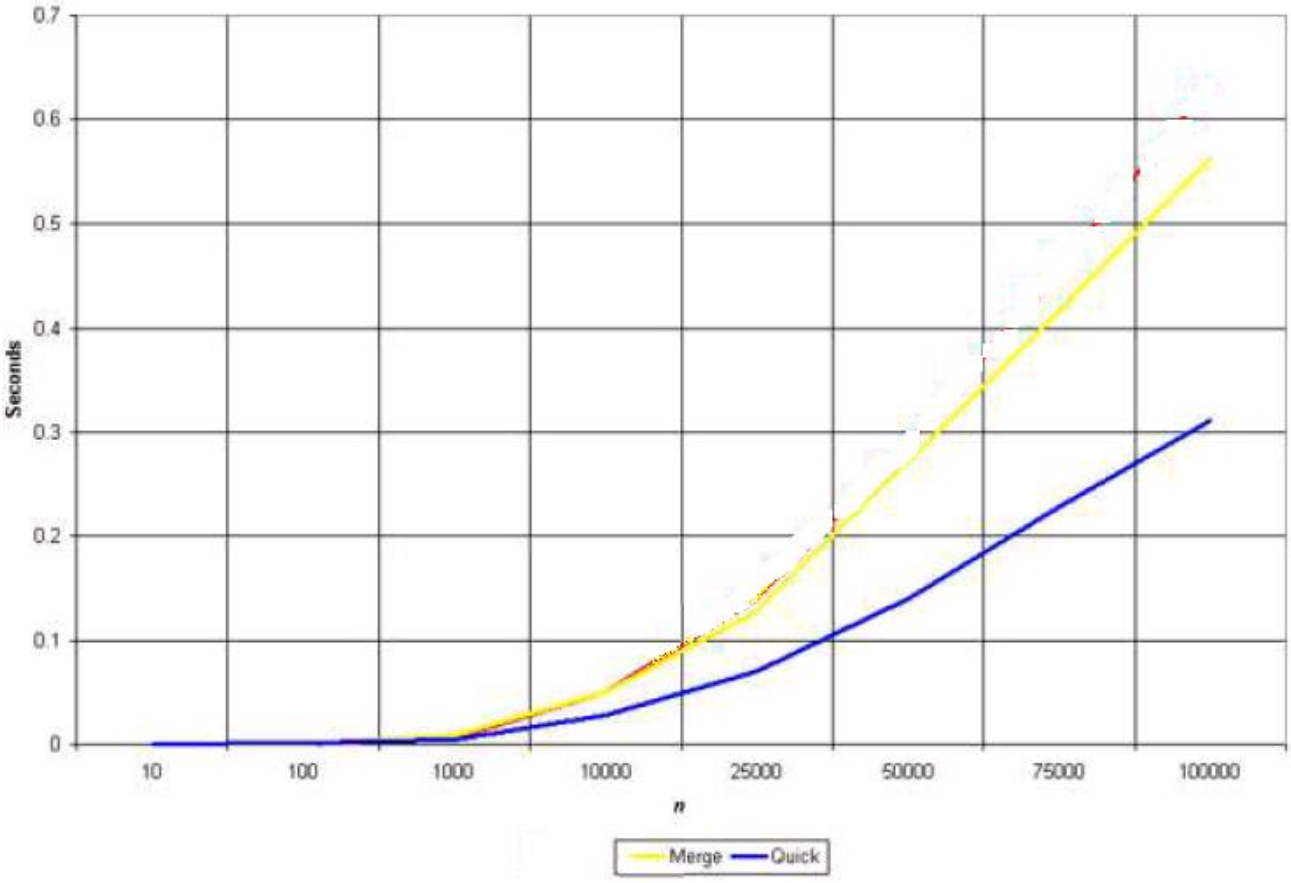
```
procédure Partitionner(var A :Tableau , début, fin, var indexPivot: entier) ;
Variables i, pivot, cpt : entier ;
début
        cpt← début ;  pivot ← t[début] ;
        pour i ←début+1 à fin faire
                        si A[i] < pivot alors
                                cpt ← cpt+1
                        permuter(A[i],t[cpt]);
                finSi ;
        finPour ;
        permuter(A,cpt,début) ;
        indexPivot ← cpt ;

fin ;
```

Complexité :

Le tri par rapport au pivot nécessite de parcourir le tableau. On itère ensuite le processus de manière récursive sur les deux sous tableaux (à gauche et à droite du pivot), et ainsi de suite jusqu'à avoir des tableaux de taille 1 (log (n) fois).
c(n)=n+2*c(n/2)
=> complexité = log(n)*O(n)=O(n*log(n)).

Exp : Soit à trier le tableau suivant : A=[27, 63, 1, 71, 64, 58, 94, 9]
9, 1, 18, 27, 64, 63, 94, 71
1, 9, 18, 27, 64, 63, 94, 71
1, 9, 18, 27, 64, 63, 94, 71
1, 9, 18, 27, 64, 63, 94, 71
1, 9, 18, 27, 63, 64, 94, 71
1, 9, 18, 27, 63, 64, 94, 71
1, 9, 18, 27, 63, 64, 71, 94
1, 9, 18, 27, 63, 64, 71, 94

Comparaison des algorithmes de tris rapides :



Exercice :

Ecrire une fonction récursive et fonction itérative qui retournent la position j dans un tableau A telle que A[j] est le début de la plus longue suite consécutive de zéros et calculer leurs complexités.

UNIVERSITE ABDERRAHMANE MIRA DE BEJAIA
Faculté des Sciences Exactes
Département d'Informatique

Module : ASD
Niveau : 2e Année Licence
Année : 2019/2020

TD N° 2 : Algorithmes de tri

Exercice N° 1 :

Soit l'algorithme de « Tri par Fusion » suivant :

```
Procédure triFusion(var A : Tableau [1..n] d'entiers,
début, fin : Eentier) ;
Variables milieu : entier
début
        si (début < fin) Alors      // au moins deux cases
                milieu ←(début + fin)/2 ;
                triFusion(A, début, milieu) ;
                triFusion(A, milieu+1, fin) ;
                fusionner(A,debut,milieu,fin) ;
        finSi ;
fin ;

Procédure Fusionner(A[n] : Tableau, début, milieu, fin :
entier) ;
Variables i, i1, i2 : entier, tmp[1..n] : Tableau ;
début
        i ←0 ;
        i1 ←début ;
        i2 ←milieu + 1 ;
        Tantque (i1 <= milieu) et (i2 <= fin) faire
                si A[i1] < A[i2] Alors
                        tmp[i] ←A[i1] ;
                        i1 ←i1 +1 ;
                Sinon
                        tmp[i] ←A[i2] ;
                        i2 ←i2 + 1
                finSi ;
                i ←i + 1 ;
        FinTantque ;
        Tantque i1 <= milieu faire
                tmp[i] ←A[i1] ;
                i ←i + 1 ; i1 ←i1 + 1 ;
        finTantque ;
        Tantque i2 <= fin faire
                Tmp[i] ←A[i2] ;
                i ←i + 1 ; i2 ←i2 + 1 ;
        finTantque ;
        A ← Tmp ;
fin ;
```

Q1- Dérouler cet algorithme sur le tableau A suivant :
A=[38, 27, 43, 3, 9, 82, 10] ;
Q2- Calculer sa complexité.

Exercice N° 2 :

Soit l'algorithme de « Tri Rapide » suivant :

```
Procédure triRapide(var A : Tableau [1..n] d'entiers,
début, fin : entier) ;
Variables indexPivot : entier ;
début
        si (début < fin) Alors   // au moins deux cases
                partitionner(A, début, fin, indexPivot) ;
                triRapide(A, début, indexPivot -1) ;
                triRapide(A, indexPivot +1, fin) ;
        finSi ;
fin ;

procédure Partitionner(var A :Tableau , début, fin, var
indexPivot: entier) ;
Variables i, pivot, cpt : entier ;
début
        cpt← début ;  pivot ← t[début] ;
        pour i ←début+1 à fin faire
                        si A[i] < pivot alors
                                cpt ← cpt+1
                        permuter(A[i],t[cpt]);
                finSi ;
        finPour ;
        permuter(A,cpt,début) ;
        indexPivot ← cpt ;

fin ;
```

Q1- Dérouler cet algorithme sur le tableau A suivant :
A=[27, 63, 1, 71, 64, 58, 94, 9] ;
Q2- Calculer sa complexité ;
Q3- Comparer cet algorithme à l'algorithme de « Tris par fusion », justifiez votre réponse.