Licence 2 Informatique et Mathématiques AP4.

Projet sur la complexité : Analyse des algorithmes de tri.

A- Le tri a bulle:

Le tri à bulles est l'une des méthodes classiques de réorganisation d'un tableau d'éléments avec comparaisons. Il est loin d'être le plus efficace parmi les tris par échanges, mais son principe reste parmi les plus simples à faire.

Principe de l'algorithme:

Considérons un tableau T d'entiers de taille N.
Le principe du tri à bulle consiste à parcourir T en comparant deux à deux les éléments, le premier avec le deuxième, puis le deuxième avec le troisième etc, et dès que deux éléments consécutifs ne sont pas dans l'ordre(croissant pour notre cas), on les échange. Après le premier parcours, le maximum remonte en haut du tableau. On recommence un autre parcours, en se limitant à l'avant dernière case du tableau, et ainsi de suite. Au $i^{ème}$ parcours on remonte le $i^{ème}$ plus grand élément du tableau à sa place jusqu'à n'avoir qu'un seul élément qui sera le minimum du tableau.

Voici ci_dessous un petit exemple d'un tableau de taille N=5 :

7 3 8 5 0			
3 7 8 5 0	3 7 5 0 8		
3 7 8 5 0	3 7 5 0 9	3 5 0 7 9	
3 7 5 8 0	3 5 7 0 9	3 5 0 7 9	3 0 5 7 9
3 7 5 0 **8**	3 5 0 **7 9**	3 0 **5 7 9**	**0 3 5 7 9**
itération 1	itération 2	itération 3	itération 4

Les Cases vertes représentent les éléments déjà triés et en rouge les échanges d'éléments.

Complexité :

Soit un tableau de taille N. Effectuer le tri à bulle sur consiste à faire un parcours avec deux boucles imbriquées.
Le tri bulle a donc une complexité de $\mathbf{O(N^2)}$ au pire des cas. Le pire des cas correspond au cas où le tableau est au départ trié par ordre décroissant, dans ce cas l'algorithme doit remonter chaque élément jusqu'à la $i^{ème}$ place à chaque étape i, en effectuant à chaque fois un

échange et s'il n'est pas optimisé, cela reviens à faire **(N-1)*(N-1)** comparaisons (et quelques permutations), soit une complexité de **$O(N^2)$.**

En ce qui concerne la complexité en temps, le temps augmente quadratiquement en fonction de la taille du tableau lorsque celui-ci est très grand(voir Figure 1).
Ainsi à partir d'une certaine valeur de N(par exemple si N >= 10^6), le programme peut prendre plusieurs minutes avant de pouvoir effectuer le tri.

La complexité en moyenne du tri à bulles est également en **$O(N^2)$**, ce qui le rend trop peu efficace par rapport à d'autres algorithmes de tri comme le tri rapide.
Le tri à bulle est en fait un des algorithmes de tri les plus lents, il est donc rarement utilisé en pratique.

B- Le tri Rapide(ou quick sort)

L'algorithme de tri rapide est un algorithme de type dichotomique et comme tous les algorithmes qui divisent et traitent le problème en sous-problèmes à l'aide d'une partition du tableau à trié.

Principe de l'algorithme:

Le tri rapide utilise le principe de diviser pour régner, c'est-à-dire que l'on va choisir un élément du tableau que l'on dénomme **pivot**(le premier élément pour notre cas), puis on partitionne le tableau en deux sous tableaux : l'un contenant les éléments inférieurs au pivot au début du tableau et l'autre les éléments supérieurs à la fin du tableau.
En suite, on met le pivot à sa place et on recommence de façon récursive sur chacune des parties du tableaux.

Complexité :

Comme le tri à bulle, le pire des cas de ce tri correspond à lorsque le tableau est trié de manière décroissante, le choix de la première valeur de la liste comme pivot produira deux sous-tableaux de taille 0 et (N-1). L'algorithme de tri rapide dans ce cas a une complexité en $O(N^2)$.

```
fonction TRI_RAPIDE (T , deb , fin)
1       si (deb < fin) alors
2               place_pivot ← PARTITION(T , deb, fin),          O(N)
3               TRI_RAPIDE (T , deb , place_pivot − 1),         T (0)
4               TRI_RAPIDE (T , place_pivot + 1, fin),          T (N-1)
```

Dans le pire des cas
$T(N) = T(0) + O(N) + T(N-1) =$ **$O(N^2)$.**

Dans la majorité des cas, le tri rapide sera donc préféré au tri fusion ou tri à bulle et sa complexité dans le meilleur des cas reste **O(N*log(N)**).

```
fonction TRI_RAPIDE (T , deb , fin)
1       si (deb < fin) alors
2               place_pivot ← PARTITION(T , deb, fin),          O(N)
3               TRI_RAPIDE (T , deb , place_pivot – 1),         T (N/2)
4               TRI_RAPIDE (T , place_pivot + 1, fin),          T (N/2)
```

Dans le meilleur cas
$T(N) = O(N) + 2 \times T(N/2) = O(N \times \log N)$.

C- Le tri par dénombrement :

Le tri par dénombrement est un tri qui s'effectue avec nombres entiers supérieurs à zéro. Ce tri est un tri limité car on ne peut utiliser ce tri pour trier des chaînes de caractères, ni des objets, on n'est dans l'obligation de trié que des entiers numériques positifs.
Raison pour laquelle le tri par dénombrement est basé sur une méthode qui permet d'utiliser les nombres qu'on est entrain de trier comme indice d'un tableau.

Principe :

Considérons en entrée un tableau **A** de taille N qui contient des valeurs positives ou nulles. Nous aurons également besoins de deux autres tableaux intermédiaires :
un tableau **B** de même taille que **A**, qui contiendra le tableau trié et un tableau C de taille max(max du tableau **A**), qui servira d'espace de stockage temporaire.

L'algorithme de tri par dénombrement se passe comme suit :

- Chercher l'élément maximum du tableau à trié(que l'on va appeler max).
- Créer un tableau C de taille max;
- Initialiser toutes les cases du tableau C à nul ;
- On incrémente chaque case de C à chaque fois que l'indice correspond à une valeur du tableau initial à trier.
- On détermine en parcourant le tableau C, le nombre d'éléments qui sont inférieurs ou égaux à l'indice, et ce en gérant un cumul constamment actualisé.
- On parcours en décrémentant B, on place chaque élément de A à sa bonne à sa place dans le tableau B.

Pour faire un tri stable, c'est-à-dire garder l'ordre de placement des valeurs identiques de A, on décrémente C[A[i]] à chaque fois qu'on place une valeur de A[i] dans B.

Alors que si les N éléments de A sont tous distincts, pour chaque indice de A la valeur C[A[i]] correspond à l'indice final de A[i] dans le tableau de sortie B.

Exemple de tri par dénombrement :
Soit A un tableau de taille 6.

indice	1	2	3	4	5	6
A	3	2	1	0	2	2

Étape 1 :

indice	0	1	2	3
C	0	0	0	0

Étape 2:

indice	0	1	2	3
C	1	1	3	1

Étape 3:

indice	0	1	2	3
C	1	2	5	6

Étape 4 :

indice	1	2	3	4	5	6
B	**0**				**2**	

indice	0	1	2	3
C	**0**	2	**4**	6

Étape 5:

indice	1	2	3	4	5	6
B	**0**			**2**	**2**	

indice	0	1	2	3
C	**0**	2	**3**	6

En fin, on obtient

indice	1	2	3	4	5	6
B	**0**	**1**	**2**	**2**	**2**	**3**

Une fois le tri terminer on fait une recopie des éléments de B dans A, ainsi A est trié.

Complexité :

L'algorithme de tri par dénombrement est un tri qui ne contient aucune comparaison entre les éléments au contraire, il utilise les valeurs comme indice d'un tableau. La complexité pour ce algorithme de tri est de **O(N)** car le premier parcours de boucle(du tableau C) pour prend un temps O(max), la deuxième boucle **O(N)**, la troisième boucle prend **O(N)**, ce qui fait une complexité global de **O(max +N).**

Par contre, l'algorithme de tri est à quelques subtilités sur son temps d'exécution :
si les valeurs à triées se rapprochent en valeurs par exemple (3, 2, 1, 0, 2, 2), le tri par dénombrement est très efficace.
En revanche, si les valeurs ne le sont pas dans le même ordre par exemple(3, 1259, 1, 98765, 58, 2) le tri par dénombrement n'est pas aussi efficace et peut être moins efficace que le tri rapide ou même parfois le tri à bulle.
Toutefois, malgré l'efficacité de cet algorithme, sa complexité en mémoire est très mauvaise car lors de son application nous créons à chaque fois deux tableaux intermédiaires que nous n'utilisons pas après le tri. C'est pourquoi quand on choisit d'utiliser ce algorithme de tri, il faut être prêt à consommer son espace mémoire.

D- Opération Élémentaires :

En terme d'opération élémentaires, comme la complexité le tri à bulle bat le record du plus grand nombre d'opérations élémentaires(comparaisons et affectations), car malgré l'optimisation effectuée sur ce tri le nombre d'opérations plus élevés par rapport aux deux autres tris (tri rapide et dénombrement).

E- Courbes :

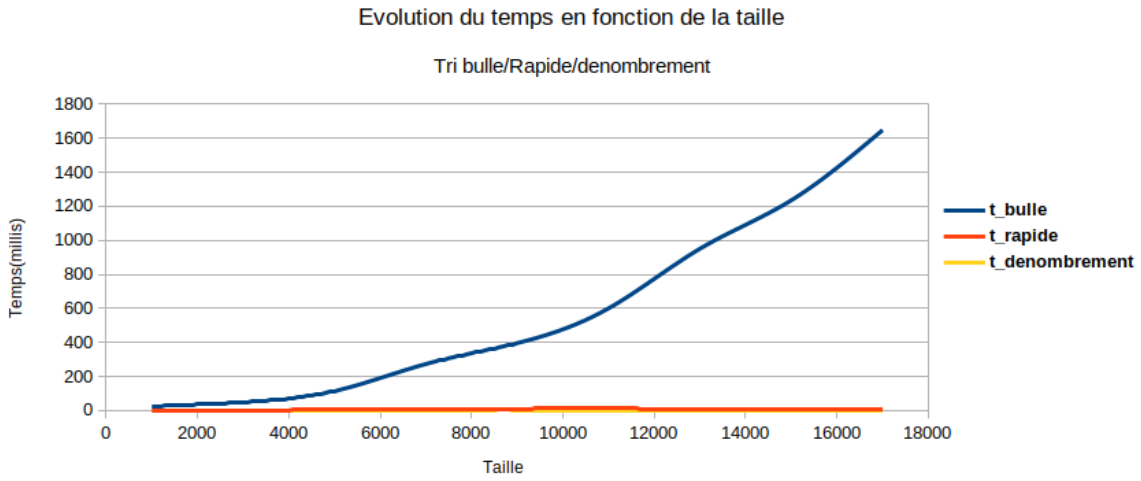

Figure 1: Evolution du temps en fonction de la taille du tri à bulle, rapide et dénombrement

Voici sur le graphe(Figure 1) une représentation de trois courbes qui présentent l'évolution du temps en fonction de la taille des trois tris. On voit clairement que le tri à bulle à ses débuts croit comme les deux autres tris(rapide et dénombrement) mais à partir d'une certaine taille (N=4000 sur notre graphe), il prend assez temps pour s'exécuter et sa courbe se détache des deux autres.

En ce qui concerne le tri rapide et dénombrement, on a l'impression que les deux tris évoluent de la même façon(ou ont la même courbe) car ils sont beaucoup trop rapide par rapport au tri à bulle. Ils sont presque linéaires à une certaine valeur de la taille.

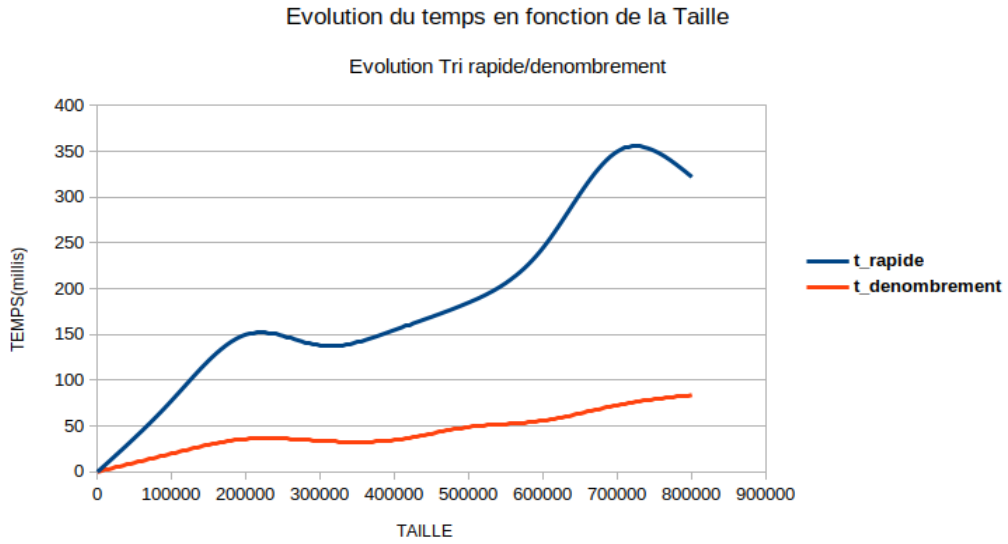

Figure 2: Tri rapide & Tri par dénombrement

Comparons sur le graphe ci-dessous le tri rapide et le tri dénombrement. Sur la **Figure1**, on ne voit pas clairement l'évolution du tri rapide par rapport à celui du dénombrement. mais sur la **Figure 2** on constate que le tri à met beaucoup plus de temps par rapport au tri par dénombrement. Nous pouvons en déduire que plus la taille est grande plus le tri rapide est plus lent.

Table des matières