

SAE Algorithmie

Tâche 1 : Vérifier

Parmi les trois algorithmes qui nous ont été donnés, nous avons reconnu les trois algorithmes de tri suivants : tri par insertion, tri cocktail, tri rapide. Nous avons ensuite vérifié et corrigé les petites erreurs présentes dans les programmes (voir les programmes).

Tâche 2 : Décrire

Le tri par Insertion :

6 5 3 1 8 7 2 4

Le tri par insertion est un algorithme de tri stable et en place. Il est considéré comme étant l'algorithme de tri le plus rapide lorsqu'il s'agit de trier de petites listes, ou des listes presque triées mais il est beaucoup plus lent lorsqu'il faut trier de grandes listes et il se retrouve loin derrière des algorithmes comme le tri rapide ou le tri fusion. Le tri par insertion peut être utilisé en complément d'autres tris (il peut servir de module d'amélioration pour un tri rapide par exemple).

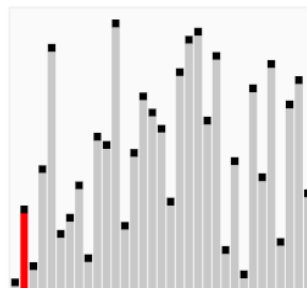
Le tri par insertion va parcourir une liste ou un tableau et comparer chaque élément avec l'élément qui le suit. Si l'élément qui le suit est plus petit alors il permutera les deux valeurs et il recommencera à parcourir le tableau en revenant au début.

De cette manière, le tri par insertion va trier la partie gauche du tableau et au fur et à mesure qu'il avance, il va insérer chaque chiffre à l'intérieur de la partie triée du tableau.

Cet algorithme de tri a donc une rapidité qui dépend de la taille du tableau avec une **complexité temporelle de $O(n^2)$ dans le pire cas (et en moyenne)** et une **complexité temporelle de $O(n)$ dans le meilleur cas**. Sa **complexité spatiale est $O(1)$** .

Le pire des cas pour cet algorithme serait d'avoir à trier un tableau trié à l'envers.

Le tri cocktail :



Le Tri Cocktail est une version améliorée du Tri à Bulles. C'est un algorithme de tri stable et en place.

Le tri à bulles consiste à parcourir le tableau de gauche à droite en comparant deux valeurs consécutives plusieurs fois tant que le tableau n'est pas trié.

Le tri cocktail fonctionne de la même manière mais il parcourt aussi le tableau de droite à gauche, ce qui fait gagner un aller retour au tri.

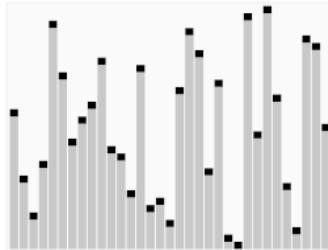
L'algorithme va donc déplacer l'élément le plus grand à son emplacement définitif quand il parcourra le tableau de gauche à droite, et déplacera l'élément le plus petit à son emplacement définitif quand il parcourra le tableau de droite à gauche.

Complexité : $O(n^2)$ dans le pire des cas ou dans les cas moyens.

$O(n)$ quand la liste est presque ordonnée.

Tout comme l'algorithme précédent, le meilleur des cas serait un tableau presque déjà trié et le pire des cas serait un tableau trié à l'envers. Apparemment, le tri cocktail et le tri à bulles sont des tris intéressants à étudier mais ils ne sont pas plus performants qu'un tri par insertion : c'est ce qu'il nous reste à prouver.

Le tri Rapide :



Le tri rapide est un tri différent du tri cocktail ou du tri par insertion.

L'algorithme de tri rapide passe par plusieurs étapes.

- Dans un premier temps on définit un pied de pivot, cela peut être un élément au hasard ou un élément défini par l'utilisateur ou par un autre algorithme (nous notre pied de pivot sera l'élément central du tableaux).
- Dans un deuxième temps il ordonne le tableau en 2 parties (appel d'un autre algo), la partie de gauche composé des éléments plus petits que le pied de pivot, et la partie de droite avec les éléments de la même taille ou plus grand.
 - Il échange le pied de pivot avec l'élément de fin de tableau (ou de début mais on étudiera seulement le cas ou échange avec l'élément de fin).
 - Pour cela l'algorithme parcourt les éléments à trier dans l'ordre (grâce à un premier compteur indentation à chaque tour de boucle), et grâce à un deuxième compteur il retient le dernier élément qui est supérieur ou égal au pied de pivot.
 - Quand il tombe sur une valeur plus petite que le pied de pivot, il échange les éléments correspondant au 2 compteurs et augmente le deuxième compteur de 1.
 - Échange le pied de pivot (dernier élément du tableau) avec l'élément du deuxième compteur afin de remettre le pied de pivot entre les éléments plus petits que lui et les plus grands que lui.
- Dans un dernier temps l'algorithme utilise la récursivité en se rappelant mais en changeant les valeurs d'entrée afin de recommencer mais qu'avec une sous partie du tableau (droite ou gauche).

La récursivité s'arrête quand il n'y a plus que 1 élément dans la partie à trier.

Complexité :

$O(n^2)$ pire des cas

$O(n \log_2(n))$ en moyenne

L'algorithme est en place (ne crée pas de nouveau tableau) mais il n'est pas en place car certain élément de même valeurs peuvent être permuté.

L'algorithme de tri rapide est moins rapide sur de petits tableaux que les algorithmes de tri par insertion et de tri cocktail, mais sur de grands tableaux il devient beaucoup plus rapide que le tri cocktail et par insertion.

Tâche 3 : Comparer

Introduction :

Pour tester les algorithmes indépendamment, nous allons leur donner le même nombre de valeurs car le système est le même. Mais au moment de comparer les types d'algorithmes entre eux, nous allons créer des

tableaux plus ou moins grands car c'est sur cette différence que les algorithmes vont se montrer plus ou moins performants les uns par rapport aux autres.

Explications :

Test des 3 cas possibles pour les 3 algos :

Rappel, pour les 2 cas suivant, le pire cas est un tableau d'entiers de 1 à 10 trié par ordre décroissant, le cas aléatoire est un tableau d'entiers aléatoires entre 0 et 10 et le meilleur cas est un tableau d'entiers de 1 à 10 presque trié par ordre croissant (nos algorithmes trient les tableaux par ordre croissant).

Tri par insertion :

```
TRI PAR INSERTION
0.003456899999999985 pire cas
0.002782200000000068 aleatoire
0.000770999999999962 meilleur cas
>>> |
```

Tri cocktail :

```
TRI COCKTAIL
0.0020693999999999999 pire cas
0.0030301000000000355 aleatoire
0.0006720000000000059 meilleur cas
>>>
```

Tri rapide :

Rappel, pour le tri rapide et pour un tableaux de taille 10 et contenant les chiffres allant de 1 à 10, le pire cas est le tableaux [3,10,5,9,1,2,4,6,8,7], le cas aléatoire et le meilleur cas sont identiques au 2 autres tris.

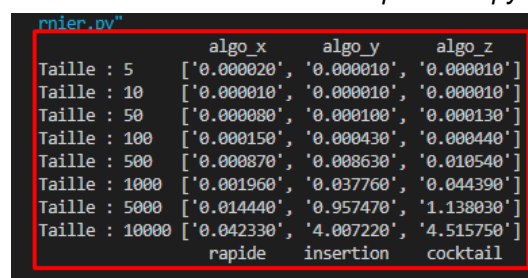
```
TRI RAPIDE
0.0053016000000000017 pire cas
0.006890399999999963 aleatoire
0.0014954000000000356 meilleur cas
>>> |
```

Après les tests, pour les 3 algorithmes, nous pouvons conclure que le cas aléatoire et le pire cas font bel et bien partie des ensembles de valeurs qui prennent le plus de temps à être triés mais qu'en pratique, le tableau d'entiers aléatoires donne plus de mal aux algorithmes de tri par insertion et de tri cocktail.

Parfois le cas aléatoire prend moins de temps que le meilleur cas mais ça n'est pas représentatif de la moyenne.

Test des 3 algos en fonctions de la taille du tableaux (valeurs aléatoires) :

il est possible d'afficher la comparaison en exécutant le fichier comparaison.py



Taille	algo_x	algo_y	algo_z
5	'0.000020'	'0.000010'	'0.000010'
10	'0.000010'	'0.000010'	'0.000010'
50	'0.000080'	'0.000100'	'0.000130'
100	'0.000150'	'0.000430'	'0.000440'
500	'0.000870'	'0.008630'	'0.010540'
1000	'0.001960'	'0.037760'	'0.044390'
5000	'0.014440'	'0.957470'	'1.138030'
10000	'0.042330'	'4.007220'	'4.515750'
	rapide	insertion	cocktail

Pour des tableaux très courts, les algorithmes de tri cocktail et tri par insertion (qui sont du même type) vont s'avérer plus efficaces que le tri rapide.

Quand le tableau est de taille 5 ou 10, les algorithmes de tri par insertion et tri cocktail sont **2.000** fois plus rapides que le tri rapide car la liste à trier est relativement petite.

A partir de tableaux de plus grande taille, les algorithmes de tri par insertion et tri cocktail deviennent de moins en moins efficaces, et l'algorithme de tri rapide commence à prouver son efficacité.

Quand le tableau est de taille 50, nous pouvons constater cela: l'algorithme de tri rapide est **1.250** fois plus rapide que l'algorithme de tri par insertion.

Jusque là, la différence entre les algorithmes de tri par insertion et tri cocktail n'est pas à noter car soit leur temps d'exécution est identique soit le temps d'exécution le plus rapide alterne entre les deux algorithmes en fonction des tableaux générés.

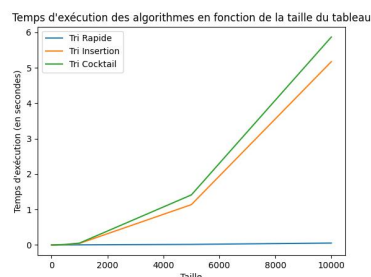
La différence entre le tri par insertion et le tri cocktail commence à se faire sur des tableaux de plus en plus grands.

Grâce à notre test sur un tableau de see entiers, nous avons pu repérer un écart récurrent entre le tri cocktail et le tri par insertion.

Quand le tableau est de taille see, l'algorithme de tri par insertion est **1.221** fois plus rapide que le tri cocktail.

Si nous faisons un saut dans l'analyse jusqu' au dernier test qui est une liste de taille 10 00, nous finissons avec un écart non négligeable entre chaque algorithme

- L'algorithme de tri rapide est 94.666 fois plus rapide que l'algorithme de tri par insertion.
- L'algorithme de tri rapide est 1e6.680 fois plus rapide que l'algorithme de tri cocktail
- L'algorithme de tri par insertion est 1.127 fois plus rapide que l'algorithme de tri cocktail.



*Un graphique est censé apparaître lors de l'exécution du programme.

Conclusion :

Pour conclure cette comparaison, nous avons estimé d'après nos recherches que le tri par insertion était égal sinon meilleur que le tri cocktail, et que ces deux derniers seraient plus rapides que le tri rapide sur de petits tableaux mais qu'avec de longues listes le tri rapide prend largement l'avantage.

Ces estimations ont été confirmées par la comparaison en pratique des trois algorithmes.

Tâche 4 : Écrire

nom : trisae.py

Nous avons écrit notre propre algorithme de tri, pour cela nous avons tout d'abord réfléchi à quelle type d'algorithmes nous allions coder. Notre premier choix c'est porter sur un algorithme non étudié en cours, le tri par interclassement montone.

Après plusieurs heures de codage notre algorithme fonctionnait et était capable d'écrire le tableaux trié, mais pour une raison indéterminée dès que nous sortions des fonctions de l'algorithme le tableaux ne gardait pas ces modifications et redevenait non trié. C'est pour cela que nous avons écrit un nouveau programme de tri, un tri par sélection qui tri pour un parcours du tableau la plus petite ainsi que la plus grande valeur.

Explication :

- On parcourt $n/2$ fois le tableaux (n est la longueur du tableau)
 - Pour chaque parcours on repère la valeur minimum et maximum
 - On échange la valeur minimum avec le 1er élément du tableau et la valeur maximum avec le dernier élément du tableau (cas spécial si la valeur maximum est le 1er élément du tableau)

- on définit la taille du tableau à trier, le tableau commencera un élément plus loin et terminera un élément plus tard

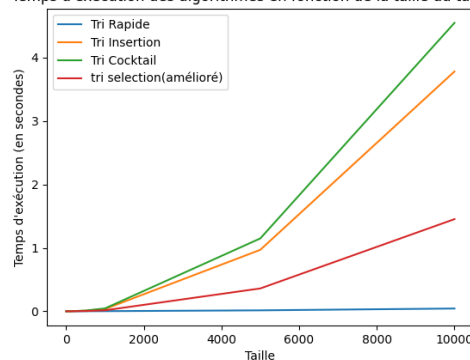
Complexité :

spatial : $O(1)$
 temporelle : $O((n/2)n)$
 en place : oui
 stable : non

Comparaison :

	algo_x	algo_y	algo_z	trisa
Taille : 5	['0.000020', '0.000010', '0.000010', '0.000010']			
Taille : 10	['0.000010', '0.000010', '0.000010', '0.000010']			
Taille : 50	['0.000090', '0.000130', '0.000120', '0.000070']			
Taille : 100	['0.000150', '0.000380', '0.000470', '0.000210']			
Taille : 500	['0.000860', '0.010220', '0.011870', '0.003880']			
Taille : 1000	['0.001950', '0.044270', '0.053350', '0.017450']			
Taille : 5000	['0.016330', '1.017230', '1.197430', '0.402090']			
Taille : 10000	['0.044160', '3.819620', '4.683800', '1.489450']			
	rapide	insertion	cocktail	selection (amélioré)

Temps d'exécution des algorithmes en fonction de la taille du tableau



*Un graphique est censé apparaître lors de l'exécution du programme.

Comme nous pouvons le voir, pour des tableaux de petite taille, notre algorithme a des résultats similaires à un algorithme de tri cocktail ou insertion. Or dès que la taille du tableau à trier augmente il devient meilleur que le tri cocktail ou tri par insertion (2,5 fois plus rapide pour 10000 valeurs). Mais ce n'est pas pour autant qu'il est le meilleur, il reste néanmoins beaucoup plus long qu'un algorithme de tri rapide (40 fois moins rapide pour 10000 valeurs).

Tâche réaliséé:

Valentin :

Je me suis chargé de comprendre, corriger, expliquer l'algorithme de tri rapide nommé algo_x. Puis je suis directement passé à la rédaction de notre algorithme de tri (tri par sélection amélioré). Et pour finir j'ai expliqué et testé cet algorithme. J'ai bien sûr contribué à la rédaction et l'optimisation de ce rapport.

Mohcine :

Je me suis occupé de tout ce qui concerne les algorithmes de tri par insertion et tri cocktail (correction, explication, test, comparaison des résultats), de la rédaction de ce rapport, et de la comparaison entre les trois algorithmes.

lien vers google docs pour voir les gifs :

https://docs.google.com/document/d/1K7QSyZrKrG0_2QdIHdlgS4CtiXLMfTgjR-Wf50d_RcA/edit?usp=s
 haring