



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Université
de Toulouse

Faculté
des Sciences
et d'Ingénierie

Les tris

CTD 7 : Les tris

Algorithmique

Introduction

Pourquoi trier ?

Comment trier ?

Tri par insertion

Idée

Dans un tableau

Dans une liste chaînée (récursif)

Tri par sélection

Idée

Dans un tableau

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Introduction

Pourquoi trier ?

Comment trier ?

Tri par insertion

Tri par sélection

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Introduction - Pourquoi trier ?

- ▶ accès rapide à une donnée
- ▶ visualisation
- ▶ étude de propriétés
- ▶ classement...

Introduction - Comment trier ?

- ▶ Objectif : modifier l'ordre des données
- ▶ Critère de comparaison
 - ▶ $3 < 7$
 - ▶ "abs" < "ade"
 - ▶ Janvier < Mars
- ▶ Traiter chaque élément
 - ▶ parcourir tout le tableau
 - ▶ combien de parcours ?
- ▶ Déplacer les éléments mal placés
 - ▶ Insertion
 - ▶ Permutation

Introduction

Tri par insertion

Idée

Dans un tableau

Dans une liste chaînée (récursif)

Tri par sélection

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Tri par insertion - Idée

Rajout d'une carte de jeu dans une "main"

- ▶ prendre un élément et le mettre à "sa" place dans un ensemble déjà trié
- ▶ le programme n'accède aux cartes que séquentiellement

Principe

Parcourir le tableau de la gauche vers la droite :

- ▶ Traiter l'élément à la position i : placer ce i ème élément à sa place dans la partie gauche déjà triée $\text{tab}[0:i]$
 - ▶ la tranche $\text{tab}[0:i+1]$ devient triée



Tri par insertion

- Conditions initiales

- Un tableau contenant une seule valeur est forcément trié !

- indice = 1

- $\text{tab}[0 : \text{indice}]$ est trié

0	1	2	3	4	5
3	2	5	4	9	1


- Condition de boucle

- Il reste au moins une valeur à placer

- Au début du premier tour de boucle

- on va placer la 2^{ème} valeur

0	1	2	3	4	5
3	2	5	4	9	1



Tri par insertion

- Après 1 tour de boucle

- La 2^{ème} valeur a été placée par rapport à la première,

indice = indice + 1 # indice = 2

→ **tab[0 : indice] est trié**

0	1	2	3	4	5
2	3	5	4	9	1

- Après 2 tours de boucle

- La 3^{ème} valeur a été placée par rapport aux précédentes

indice = indice + 1 # indice = 3

→ **tab[0 : indice] est trié**

0	1	2	3	4	5
2	3	5	4	9	1

- Après i tours de boucle

- La i^{ème} valeur a été placée par rapport aux précédentes,

indice = indice + 1 # indice = i + 1

→ **tab[0 : indice] est trié**

0	1	2	i	...	len(tab) - 1
2	3	4	5	9	1

Tri par insertion

- Pour un tableau de n éléments, après avoir exécuté les $(n-1)$ tours de boucle
 - indice varie de 1 à $\text{len}(\text{tab})$
 - Sortie de la boucle car il n'y a plus de valeurs à classer
 - $\text{indice} = \text{len}(\text{tab})$
 - Etat final à la sortie de la boucle
 - le tableau est entièrement trié
 - $\text{tab}[0 : \text{indice}]$ est trié donc
 - $\text{tab}[0 : \text{len}(\text{tab})]$ est trié

Tri par insertion - Dans un tableau

Exercice 1

Ecrire la fonction de tri par insertion pour **un tableau**

Tri par insertion - Dans une liste chaînée (récursif)

Exercice 2

Le tri par insertion est un algorithme qui permet d'insérer un élément en bonne position dans une liste déjà triée dans l'ordre croissant. On construit une liste qui sera triée sans modifier la liste initiale.

La fonction récursive `tri_par_insertion` aura donc besoin de la fonction :

- ▶ `inserer_elt_dans_liste_triee` qui insère un élément au bon endroit dans une liste déjà triée par ordre croissant.

Introduction

Tri par insertion

Tri par sélection

Idée

Dans un tableau

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Tri par sélection

- Idée : mettre le plus petit en premier
- Principe
 - Trouver le plus petit élément
 - Le substituer avec le premier élément du tableau
 - Trouver le 2nd plus petit élément
 - Le substituer avec le 2nd élément du tableau
 - ...
- Combien de parcours du tableau ?
- Algorithme simple mais peu efficace sur de grandes quantités de données

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Tri par sélection

- Conditions initiales

- indice = 0
- Un tableau vide est forcément trié !
- $\text{tab}[0 : \text{indice}]$ est trié

0	1	2	3	4	5
3	2	5	4	9	1

- Condition de boucle

- Il reste au moins une valeur à placer

- Premier tour de boucle

- On cherche la plus petite valeur dans $\text{tab}[\text{indice} : \text{len}(\text{tab})]$
- Substituer avec $\text{tab}[\text{indice}]$
- $\text{indice} = \text{indice} + 1$
- $\text{tab}[0 : \text{indice}]$ est trié


0	1	2	3	4	5
1	2	5	4	9	3

Tri par sélection

- $i^{\text{ème}}$ tour de boucle

- On cherche la plus petite valeur dans $\text{tab}[\text{indice} : \text{len}(\text{tab})]$
- Substituer avec $\text{tab}[\text{indice}]$
- $\text{indice} = \text{indice} + 1$
- $\text{tab}[0 : \text{indice}]$ est trié

0	...	i	$\text{len}(\text{tab})-1$
1	2	x	y	z	t



- Question : Etat du tableau après avoir exécuté les $(n-1)$ tours de boucle ?
- Question : Condition d'arrêt ?
 - Sortie de la boucle car il n'y a plus de valeurs à classer
 - $\text{tab}[0 : \text{len}(\text{tab})]$ trié
 - Etat final à la sortie de la boucle
 - le tableau est entièrement trié

Introduction

Tri par insertion

Tri par sélection

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Exercice 3

Le tri par fusion est un algorithme qui propose de découper la liste en 2 parties de longueur égale (à un élément près) que l'on doit ensuite trier pour enfin fusionner ces 2 listes triées.

La fonction récursive `tri_par_fusion` aura donc besoin de deux fonctions :

- ▶ `partager_liste_en_2` qui, étant donnée une liste, retourne un couple de listes de longueurs équivalentes formées des éléments de la liste à partager ;
- ▶ `fusionner_2_listes_triees` qui, étant données 2 listes triées dans l'ordre croissant, construit dans l'ordre croissant la liste des éléments de ces 2 listes.

Introduction

Tri par insertion

Tri par sélection

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Tri rapide (ou par pivot) -

Exercice 4

Le tri rapide est un algorithme de tri très utilisé pour sa relative simplicité et sa rapidité.

On peut en donner une solution récursive :

- ▶ Si la liste a moins de 2 éléments, c'est fini, la liste est déjà triée.
- ▶ Sinon
 - ▶ On choisit un élément de la liste au hasard qu'on appelle pivot ; ici on prendra le 1^{er} élément de la liste pour simplifier ;
 - ▶ On partitionne la liste en 2 listes : la liste des éléments qui sont inférieurs au pivot et la liste des éléments qui sont supérieurs au pivot ;
 - ▶ On trie ensuite ces 2 listes suivant la même méthode (2 appels récursifs) ;
 - ▶ On construit la liste résultat en concaténant les 2 listes triées sans oublier le pivot entre les 2.

Introduction

Tri par insertion

Tri par sélection

Tri fusion

Tri rapide (ou par pivot)

Conclusion

Survol des algorithmes existants

- *Tri par insertion*
- Tri de Shell (shell sort) (amélioration du tri par insertion)
- *Tri par sélection*
- *Tri fusion (merge sort) (tri récursif : diviser pour régner ...)*
- *Tri rapide (quick sort) (tri récursif après partition selon un pivot)*
- Tri à bulles
- Sedgesort (amélioration du tri rapide)
- Tri par tas (heap sort)
- Tri par ABR (arbre binaire de recherche et parcours en profondeur)
- Smoothsort (amélioration du tri par tas)

Survol des algorithmes existants

https://fr.wikipedia.org/wiki/Tri_de_Shell

https://fr.wikipedia.org/wiki/Tri_à_bulles

https://fr.wikipedia.org/wiki/Tri_par_tas

https://fr.wikipedia.org/wiki/Tri_rapide

Critères de sélection d'un tri

- Notion de complexité
 - Nombre de boucles,
 - Boucle dans des boucles
 - nombre de permutations,
 - complexité du code
- Impact sur le temps d'exécution
- Comportement selon grand nombre de données ?
- Ce qu'on sait déjà des données (efficacité moindre si données déjà triées, ou légèrement triées...)

Comparaison tris sur tableaux moyens

