Plan du CM 10

Algorithmes de tri

Problème

```
Entrée T tableau de n entiers
Sortie T ou T' tableau trié contenant ces n entiers
```

Tri interne

Si la sortie est T et qu'il ne nécessite pas l'allocation d'un autre tableau, on dit que le tri est interne ou encore qu'il s'effectue sur place.

Tri externe

Si l'algorithme nécessite l'allocation d'un autre tableau T', on dit que le tri est externe.

Algorithmes de tri
Comparaisons des algorithmes de tri

Coût de l'algorithme

On compte le nombre de comparaisons entre deux éléments.

Beaucoup d'algorithmes de tri

Il existe beaucoup d'algorithmes de tri.

- sont-ils tous équivalents ?
- connaît-on la complexité du tri ?

Classification des algorithmes

Nous allons voir que l'on peut séparer les différents algorithmes en deux classes

- algorithmes lents, complexité en $\Theta(n^2)$
- algorithmes rapides, complexité en $\Theta(n \log n)$

Algorithmes de tri
Complexité

Entrées

- les n entiers à trier sont $[n] = \{1, \dots, n\}$ (pas de doublons)
- l'ensemble des entrées E est l'ensemble des permutations sur $[n]$
- $card(E) = n!$, nombre de permutations sur $[n]$.

Complexité dans le pire des cas

Notons $C(e)$ le coût de l'algorithme pour l'entrée e.
La complexité dans le pire des cas vaut

$$max\{c_e \mid e \in E\}.$$

Complexité en moyenne

Notons $p_e = \Pr(\text{on tire l'entrée } e)$. Le coût en moyenne vaut alors

$$E[C] = \sum_{e \in E} C_e\, p_e.$$

On choisit en général l'équiprobabilité ou équirépartition ou distribution uniforme

$$p_e = \frac{1}{n!}.$$

Plan du CM 10

Tri par insertion

Principe

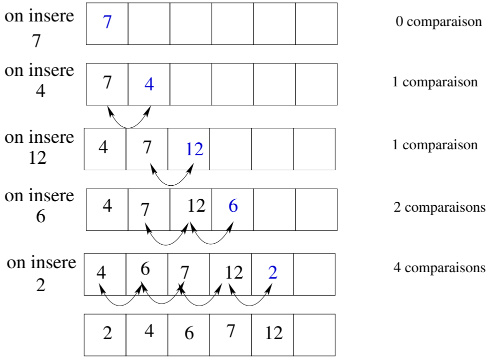
- n étapes, à l'étape i on insère le ième élément
- on insère les éléments un par un en les mettant à chaque fois à la bonne place
- si l'élément est inséré à la place i, on décale à droite tous les éléments en position j avec $j \geq i$

Insertion du ième élément

- on met le ième élément e en fin de tableau
- on le place donc en position $k = i - 1$.
- tant que e est plus petit que l'élément en position $k - 1$, on le permute avec celui-ci

Tri par insertion

Exemple



Tri par insertion

Complexités

- complexité dans le meilleur des cas
  - on insère par ordre croissant
  - on insère toujours en dernière position, 1 comparaison à chaque étape
  - complexité $0 + 1 + \ldots + 1 = n - 1$
- complexité dans le pire des cas
  - on insère par ordre décroissant
  - on insère toujours en première position, $k - 1$ comparaisons à l'étape k
  - complexité : $0 + 1 + 2 + \ldots + n - 1 = \dfrac{n(n-1)}{2}$
- complexité en moyenne (distribution uniforme)

  environ $\dfrac{(n+1)(n+4)}{4}$ (voir slide suivant pour le détail des calculs.)

La complexité dans le pire des cas et en moyenne est en $\Theta(n^2)$.

Tri par insertion

Comparaisons à l'étape i

0	1	2		i–2	i–1
23	50	121	...	1000	

au depart : i–1 elements
entre les positions 0 et i–1

e ?	e ?	e ?	e ?	e ?	e ?
i–1	i–1	i–2			1

place de e
entre les positions 0 et i–1

nombre de comparaisons
selon la position de e

$C_{i,k}$: nombre de comparaisons lorsque l'élément se retrouve en position k.

$$C_{i,k} = i - k,$$

sauf pour $k = 0$ où l'on a $C_{i,0} = i - 1$.

Tri par insertion

Nombre de comparaisons en moyenne

On suppose que e a la même probabilité d'occuper la place k pour k entre 0 et $i-1$.

$$\Pr(e \text{ est placé en position } k) = \frac{1}{i}.$$

$$\begin{aligned}
E[C_i] &= \textstyle\sum_{k=0}^{i-1} \frac{1}{i} C_{i,k} \\
&= \textstyle\frac{1}{i}(1+2+\ldots+i-1+i-1) \\
&= \textstyle\frac{1}{i}(1+2\ldots+i) - \frac{1}{i} \\
&= \frac{1}{i} * \frac{i(i+1)}{2} - \frac{1}{i} \\
&= \frac{i+1}{2} - \frac{1}{i}
\end{aligned}$$

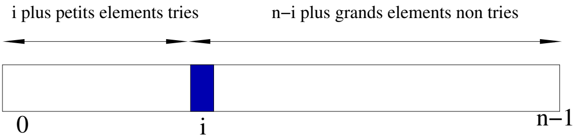
Nombre de comparaison en moyenne du tri insertion

$$\begin{aligned}
E[C] &= \textstyle\sum_{i=1} E[C_i] \\
&= \sum_{i=1}^{n} \frac{i+1}{2} - \sum_{i=1}^{n} \frac{1}{i} \\
&= \frac{1}{2} \frac{(n+1)(n+2)}{2} + o(n^2) \\
&= \frac{(n+1)(n+2)}{4} + o(n^2)
\end{aligned}$$

Tri par sélection

Principe

- rechercher le plus petit élément et le placer en tête (indice 0), puis recommencer à partir du second élément pour rechercher le deuxième plus petit élément et le placer en second (indice 1).
- après avoir placé l'élément d'indice $i - 1$, la situation est la suivante



- pour sélectionner le ième élément, on recherche l'indice du plus petit élément parmi ceux d'indice i à $n - 1$.
  On échange ensuite cet élément avec l'élément d'indice i.

Tri par sélection

Procédure auxiliaire

On définit une procédure

`minimum(tab:tableau d'entiers, debut, fin : entier)`

qui renvoie le minimum entre les positions `debut` et `fin`.
Le coût de la procédure `minimum` est `fin - debut +1`.

Etape *i* – sélection du ième élément

On appelle la procédure `minimum` avec `debut`$= i$ et `fin`$= n-1$.
la sélection du ième élément nécessite donc $n-i$ comparaisons.

Complexité

- le coût est toujours le même quelle que soit l'entrée
- soit un coût de $n-1+n-2+\ldots+1 = \dfrac{n(n-1)}{2}$

La complexité est donc en $\Theta(n^2)$.

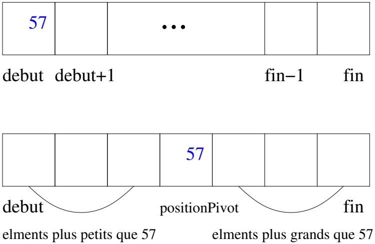
Plan du CM 10

Tri rapide (quickSort)

Placement du pivot (procédure *partition(T,debut,fin)*)

- à la première étape debut = 0 et fin = $n - 1$
- on fixe un pivot entre les positions debut et fin (généralement l'élément en position debut)
- on met à gauche du pivot tous les éléments plus petits que le pivot
- on met à droite tous les éléments plus grands que le pivot
- le pivot se retrouve à la bonne position (notée positionPivot)
- les instructions s'effectuent sur place (par des échanges entre deux éléments)

Schéma



on choisit 57 comme pivot

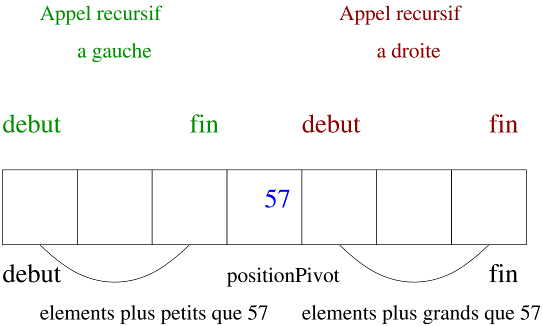
Tri rapide (quickSort)

Appels récursifs

On effectue deux appels résursifs :

- à gauche, debut = debut et fin = positionPivot - 1
- à droite, debut = positionPivot + 1 et fin = fin
- on arrête les appels récursifs lorsque debut > fin

Schéma



Tri rapide (quickSort)

Procédure quickSort

```
quickSort(T : tableau d'entiers,debut : entier,fin : entier)
   si debut < fin alors
       positionPivot=partition(T,debut,fin)
       quickSort(T,debut,positionPivot-1)
       quickSort(T,positionPivot+1,fin)
```

Procédure partition

La coût de la procédure *partition* est de fin-debut.
En effet, le pivot est comparé avec les fin-debut autres éléments entre les positions debut et fin.
(voir le TP sur les algorithmes de tri)

Tri rapide (quickSort)

Complexités

- complexité dans le meilleur des cas
  - le pivot est toujours en position $\lfloor \frac{debut + fin}{2} \rfloor$ pour $n = 2^k - 1$
  - la complexité est $\approx n \log_2 n$
- complexité dans le pire des cas
  - le tableau est déjà trié, le pivot est alors toujours en position debut
  - la complexité vaut $n - 1 + n - 2 + \ldots + 1 = \frac{n(n-1)}{2}$
- complexité en moyenne (distribution uniforme)
  - nous devons considérer toutes les partitions possibles
  - on montre que la complexité vaut $\approx 1.39 n \log_2 n$

La complexité en moyenne est donc en $\Theta(n \ \log n)$.

Tri rapide à pivot aléatoire (randomQuickSort)

Choix du pivot

- le pivot est tiré aléatoirement entre les positions debut et fin.
- chaque position a donc une chance $\frac{1}{fin - debut + 1}$ d'être tirée

Nouvelle complexité en moyenne

- on fixe l'entrée, c'est le choix du pivot qui change
- plus de meilleur des cas et pire des cas
- la complexité est en $\Theta(n \ \log n)$
- le calcul est différent, il faut considérer tous les choix de pivot possibles

Tri fusion

Principe

- on sépare le tableau en deux
- on trie chaque partie du tableau
- on fusionne les deux parties

Procédure triFusion

10	3	15	2	7	20	9	4

10	3	15	2	on separe le tableau den deux	7	20	9	4

2	3	10	15	on trie chaque demi–tableau	4	7	9	20

2	3	4	7	9	10	15	20

on fusionne les deux demi–tableaux

Tri fusion

Principe

- on sépare le tableau en deux
- on trie chaque partie du tableau
- on fusionne les deux parties

Procédure triFusion

```
triFusion(T : tableau d'entiers, debut : entier, fin : entier)
    si debut < fin alors
        milieu = partie entière de (debut+fin)/2
        T1 = triFusion(T, debut, milieu)
        T2 = triFusion(T, milieu+1, fin)
        fusion(T, T1, T2)
```

Tri fusion

Procédure fusion

- c'est cette procédure qui donne la complexité de l'algorithme
- son coût vaut fin-debut+1
- déjà étudiée : voir TD 3, exercice 2

Complexités

- complexité dans le meilleur des cas
  - avec $n = 2^k$, on divise à chaque fois en deux parties égales
  - la complexité est $\approx n \log_2 n$ (même récurrence que pour quickSort)
- complexité en moyenne
  - la complexité est en $\Theta(n \ \log n)$.
- pas de pire des cas

Autres algorithmes de tri

Tri à bulles (Bubble sort)

- on permute successivement les éléments consécutifs d'un tableau
- comme des bulles d'air qui remontent à la surface
- pas efficace, complexité dans le pire des cas et en moyenne en $\Theta(n^2)$

Tri par tas (Heap sort)

- le tri par tas code un arbre binaire avec un tableau.
- les éléments sont partiellement ordonnés par priorité
- sa complexité dans le pire des cas et en moyenne est en $\Theta(n \log n)$

Timsort

- algorithme utilisé par Python *maListe.sort*()
- mélange entre tri insertion et tri fusion
- repère si des parties sont déjà triées
- sa complexité dans le pire des cas et en moyenne est en $\Theta(n \log n)$

Variantes

- on modifie les conditions terminales
- on change d'algorithme lorsqu'il ne reste que peu d'éléments

Plan du CM 10

Complexité du problème de tri

Peut-on trouver un meilleur algorithme

On peut montrer qu'il n'existe pas d'algorithme de tri de complexité $o(n \log n)$.
Par conséquent la classe $\Theta(n \log n)$ est optimale (on ne peut pas trouver d'algorithme ayant une classe plus petite).

Idée de la preuve

Soit A un algorithme de tri fixé. Pour simplifier la preuve, on suppose qu'il est déterministe.
Une comparaison s'effectue entre le contenu de deux cases a et b comprises entre 0 et $n - 1$.
Nous avons deux issues possibles

1. $T[a] < T[b]$
2. $T[a] > T[b]$.

Complexité du problème de tri

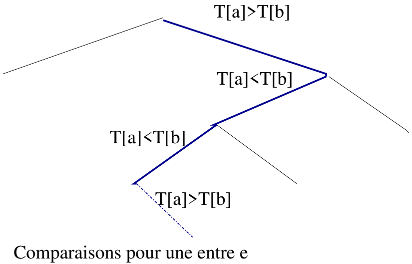
Exécution d'un tri sur une entrée

Arriver à la ième comparaison, seul les résultats successifs pour les i premières comparaisons entre $T[a] < T[b]$ ou $T[a] > T[b]$ détermine la suite de l'exécution de l'algorithme A.

Pour deux entrées différentes e_1 et e_2, nous devons donc obtenir un résultat différent pour au moins une comparaison C_i.

Arbre binaire localement complet

On construit un arbre binaire (localement complet) en mettant dans cet arbre l'issue des comparaisons successives pour chaque entrée e.



Complexité du problème de tri

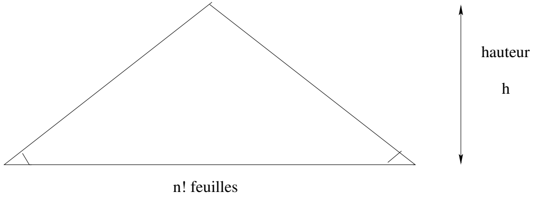
Nombre de feuilles de l'arbre

- à chaque entrée correspond une branche de l'arbre
- à une branche correspond une feuille
- l'arbre possède donc $n!$ feuilles, le nombre d'entrées

Arbre optimal

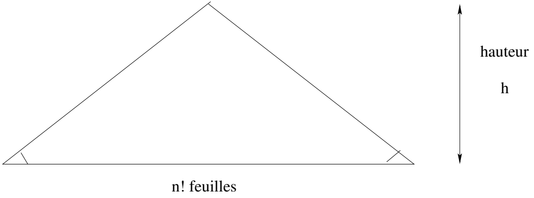
- la complexité dans le pire des cas est h, la hauteur de l'arbre.
- pour avoir un algorithme optimal, l'arbre doit se rapprocher au maximum d'un arbre complet.
- la complexité dans le pire des cas et en moyenne est alors la même.

Forme de l'arbre



Complexité du problème de tri

Forme de l'arbre



Calcul de h

Un arbre complet de hauteur h possède 2^h feuilles.
On montre en utilisant la formule de Stirling que l'on a

$$\log n! = n \log n + \Theta(n).$$

D'autre part, h vérifie l'équation

$$2^h = n!$$

D'où

$$\begin{aligned} h \ln 2 &= n \log n + \Theta(n) \\ h &= n \log_2 n + \Theta(n) \end{aligned}$$

La complexité dans le pire des cas et en moyenne ne peut pas être meilleure que $n \log_2 n$.

Plan du CM 10

Algorithmes de tri – récapitulatif

Comparaisons entre les algorithmes

- algorithmes lents en $\Theta(n^2)$
- algorithmes rapides en $\Theta(n \log n)$
- la complexité en moyenne est plus importante que celle dans le pire des cas
- il peut y avoir plusieurs complexités en moyenne (listes triées en partie, doublons)

Tableau récapitulatif

Nom de l'algorithme	complexité dans le pire des cas	complexité en moyenne
Tri sélection	$\Theta(n^2)$	$\Theta(n^2)$
Tri insertion	$\Theta(n^2)$	$\Theta(n^2)$
Tri bulle	$\Theta(n^2)$	$\Theta(n^2)$
QuickSort	$\Theta(n^2)$	$\Theta(n \log n)$
Random quickSort	$\Theta(n \log n)$	$\Theta(n \log n)$
Tri fusion	$\Theta(n \log n)$	$\Theta(n \log n)$
Tri par tas	$\Theta(n \log n)$	$\Theta(n \log n)$
Timsort	$\Theta(n \log n)$	$\Theta(n \log n)$