ISEN
ALL IS DIGITAL!
yncréa

Algorithmes de Tri
CIR 2

Partie 2

Leandro MONTERO
leandro.montero@isen-ouest.yncrea.fr

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : $\Omega(n \log(n))$

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : Ω(n log(n))
- On peut faire mieux ???

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : $\Omega(n \log(n))$
- On peut faire mieux ???

  NON !!!!

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : Ω(n log(n))
- On peut faire mieux ???

  NON !!!!

- Sauf si on commence à supposer quelque chose sur les données d'entrée...

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : Ω(n log(n))
- On peut faire mieux ???

  NON !!!!

- Sauf si on commence à supposer quelque chose sur les données d'entrée...ET on ne compare plus les valeurs comme dans les algos classiques !

Algos de Tri

- On vient de voir que n'importe quel algo de tri qui utilise des comparaisons sera toujours : Ω(n log(n))
- On peut faire mieux ???

  NON !!!!

- Sauf si on commence à supposer quelque chose sur les données d'entrée...ET on ne compare plus les valeurs comme dans les algos classiques !

- On verra deux algos qui n'utilisent pas de comparaisons !
  - Counting sort (tri casier ou tri comptage)
  - Radix sort (tri par base ou tri radix)

ISEN
ALL IS DIGITAL!
yncréa

Critères d'analyse (rappel)

- Complexité temporelle
- Complexité spatiale
- Stabilité
- Cas d’utilisation

Counting sort (tri par casier)

- Et si on suppose que les données d'entrée sont bornées ? Par exemple des entiers entre 1 et 100 !

- Idée :

  - On crée un tableau de 100 éléments initialisé tout en 0.

  - On parcourt notre tableau d'entrée et pour chaque élément, on incrémente de 1 dans sa position du tableau de comptage.

  - On parcourt notre tableau de comptage et on rempli le tableau d'entrée selon les occurrences de chaque élément !

Counting sort (tri par casier)

Input Data

0	4	2	2	0	0	1	1	0	1	0	2	4	2

Count Array

0	1	2	3	4
5	3	4	0	2

Sorted Data

0	0	0	0	0	1	1	1	2	2	2	2	4	4

Counting sort (tri par casier)

- L'algo peut être vraiment très performant par rapport aux algos classiques!
- Même par rapport à la complexité spatiale !
- Peut-on toujours l'utiliser ?

Counting sort (tri par casier)

- L'algo peut être vraiment très performant par rapport aux algos classiques!
- Même par rapport à la complexité spatiale !
- Peut-on toujours l'utiliser ?
    - NON !!

Counting sort (tri par casier)

- L'algo peut être vraiment très performant par rapport aux algos classiques!
- Même par rapport à la complexité spatiale !
- Peut-on toujours l'utiliser ?
    - NON !!
- Même pas si on veut trier un tableau de floats tous entre 1 et 10…
- Attention aux cas pertinents d'utilisation !

Radix sort (tri par base)

- C'est un algo pour trier des éléments qui peuvent être identifiés par une clef unique !
- Par exemple, des entiers (non bornés!), mots, cartes, etc.
- Peut être utilisé pour trier des floats MAIS il faut l'adapter (on ne verra pas ça).
- Appelé «Radix» car il trie les éléments selon leur «base» (symboles différents).

- Idée (LSD=least significant digit)[1]:
  - On trie nos éléments par rapport à leur chiffre (ou symbole de la base) moins significatif.
  - On passe au chiffre suivante et on répète l'opération tout en conservant l'ordre précédent.
  - On continue jusqu'à ce que tous les chiffres soient considérés.

[1] Il y a aussi MSD=most significant digit

Radix sort (tri par base)