NOM : **Prénom :** Damien **Jury :**

Algèbre ← Entourez l'épreuve → **Analyse**

Sujet choisi : 903. Exemples d'algorithmes de tri. Complexité.

Autre sujet :

I. Tris par comparaison

1. Tris naïfs

Algo 1: le tri par insertion trie un tableau en insérant un à un ses éléments dans la partie triée de ce tableau (Figure 1)

Pseudo-code 1: TriInsertion (T, n) =

```
pour i de 1 à n
    Insérer(T[i], T, i)
Retourner T
```

où Insérer(a, T, i) place l'élément a dans le sous-tableau $T[1..i]$ de telle sorte que $T[1..i]$ soit trié, sachant que $T[1..i-1]$ était trié.

Remarque 1, C'est un algorithme de tri en place, donc optimal en espace

Prop 1: le tri par insertion trie un tableau en temps $O(n^2)$ dans le pire cas et en moyenne

Algo 2: le tri à bulle trie un tableau par la fin, en faisant remonter le maximum du sous-tableau restant à trier jusqu'au bord du sous-tableau trié (Figure 2)

Pseudo-code 2: TriBulle (T, n) =

```
pour i de n à 1
  pour j de 1 à i-1
    Comparer-et-Échanger(T, j, j+1)
Retourner T
```

où Comparer-et-Échanger(T, i, j) compare $T[i]$ et $T[j]$ et les échange s'ils ne sont pas dans le même ordre que i et j.

Prop 2: le tri à bulle trie un tableau en temps $O(n^2)$ dans le pire cas et en moyenne, en place.

2. Diviser pour régner

Remarque 2: le paradigme diviser pour régner permet d'améliorer sensiblement la complexité temporelle du tri.

Algo 3: le tri fusion trie un tableau en le découpant en deux moitiés, en les triant récursivement et en les fusionnant ensuite. (Figure 3)

Pseudo-code 3: TriFusion (T, i, j)

```
Si i < j
    k = ⌊(i+j)/2⌋
    TriFusion(T, i, k)
    TriFusion(T, k+1, j)
    Fusion(T, i, k, j)
```

Prop 3: le tri fusion trie un tableau en temps $O(n \log(n))$ dans le pire cas et en moyenne

Algo 4: le tri rapide trie un tableau en le découpant en deux parties via le choix d'un pivot, et en triant récursivement ces parties.

Remarque 3: le découpage nécessite plus de temps que pour celui du tri fusion, mais il n'y a pas de fusion à réaliser ensuite.
Le choix du pivot peut être déterministe, ou pas.

Pseudo-code 4 : Tri Rapide (T, i, j) =

```
Si i < j
    k = Partition(T, i, j)
    Tri Rapide(T, i, k-1)
    Tri Rapide(T, k+1, j)
où Partition(T, i, j) =
    Choisir l entre i et j
    x = T[l]
    Placer les valeurs ≤ x avant x et les valeurs > x après x
    Retourner l'indice de x.
```

Thm 1 : le tri rapide trie un tableau en temps $O(n^2)$ dans le pire cas pour un choix déterministe de pivot, en temps espéré $O(n \log(n))$.

Remarque 4 : le temps de calcul dépend de l'équilibre de taille entre les deux sous-tableaux calculés. C'est pourquoi le choix de la médiane comme pivot est optimal.

Malgré sa mauvaise complexité dans le pire cas, cet algorithme est en général rapide et préféré au tri fusion.

3 - Tri optimal

Thm 2 : La complexité dans le pire cas d'un algorithme de tri par comparaison est au moins $\Omega(n \log n)$.

Remarque 5 : le tri fusion est donc optimal

Déf 1 : Un tas est une structure de tableau représentant un arbre binaire presque complet.

Pour tout indice i, l'indice du père de $T[i]$ est $\lfloor \frac{i}{2} \rfloor$, l'indice du fils gauche est $2i$ et celui du fils droit est $2i+1$. (Figure 4)

Un tas max est un tas tel que pour tout i $T[\lfloor \frac{i}{2} \rfloor] \geq T[i]$

Algo 5 : le tri par tas trie un tableau en en faisant un tas max. La racine étant alors le maximum, on peut la placer à la fin du tableau et recommencer sur le sous-tableau qui ne contient pas cette case.

Thm 3 : le tri par tas est réalisable en place en temps $O(n \log(n))$ dans le pire cas.

II Tris en temps linéaire

Remarque 6 : Il est possible de trier un tableau en temps linéaire si l'on ne procède pas par comparaison.

1 - Le tri par seaux.

Algo 6 : Si l'on veut trier un tableau d'entiers dont on connaît un majorant, on peut les répartir dans des seaux correspondant à chaque valeur possible puis vider les seaux dans l'ordre.

Pseudo-code 5 : Tri Seau (T, n, k) =

```
Construire un tableau A de taille k initialisé à 0
pour i de 1 à n
    Incrémenter A[T[i]]
pour i de 1 à k
    afficher A[i] fois i.
```

Prop 4 : le tri par seau trie un tableau de n entiers tous inférieurs à k en temps $O(n+k)$

Remarque 7 : On peut écrire une version stable de cet algorithme ie si deux éléments de T sont égaux, leur ordre sera conservé dans le résultat du tri.

2. Tri par radicaux

On veut trier n entiers écrits sur d digits, chaque digit pouvant prendre k valeurs

Développement 1

Thm 4 : Le tri par radicaux permet de trier un tel ensemble d'entiers ~~en~~ temps $O(d(n+k))$

III Applications

1. Algorithmique du texte

Pour repérer les occurrences d'un motif dans un texte, il peut être utile de trier dans l'ordre lexicographique les suffixes d'un ~~mot~~.

Thm 5 : Il est possible de ranger les suffixes d'un mot de longueur ~~n~~ dans l'ordre lexicographique en temps $O(n \log(n))$ et en espace $O(n)$

Idée de la preuve : ranger les préfixes de longueur k des suffixes du mot pour $k = 1, 2, 4 \ldots 2^{\lceil \log_2 n \rceil}$

2. Théorie des graphes.

Déf 2 : ~~Un~~ tri topologique pour un graphe orienté $G = (V, A)$ est un ordre $v_1 < \ldots < v_n$ sur les sommets de G tel que pour tout $(i, j) \in [\![1, n]\!]^2$, si $v_i \to v_j \in A$ alors $v_i < v_j$.

Développement 2

Thm 6 : G est un graphe orienté acyclique ssi G admet un tri topologique

Démonstration avec construction en temps linéaire du tri topologique.

IV Quelques mots d'algorithmique parallèle

Il existe des algorithmes de tri parallèles basés sur des réseaux de comparateurs, comme le tri Odd-Even, ou le tri bitonique.

Leur complexité est $O(\log(n))$ en utilisant n processeurs pour trier n éléments.

Le coût est alors $O(n \log(n))$, la complexité optimale des tris par comparaison.

Figure 1:

1 9 4 2 → 1 9 4 2 → 1 9 4 2 → 1 4 9 2 → 1 2 4 9

Figure 2:

1 9 4 2 → 1 9 4 2 → 1 4 9 2 → 1 4 2 9 → 1 4 2 9 → 1 2 4 9 → 1 2 4 9

Figure 3:

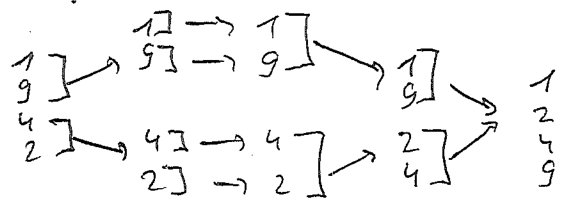

Figure 4: 16 14 10 8 7 9 3 2 4 1