PLAN

Optimalité et bornes inférieures algorithmique
a/ - L'exemple de Quicksort
b/ - La méthode dite de l'adversaire

1 Optimalité et bornes inférieures algorithmique

Etant donné un algorithme $\mathcal{A}$ avec un temps d'exécution $_{\mathcal{A}}(n)$ (resp. nécessitant l'utilisation d'une mémoire $M_{\mathcal{A}}(n)$) où n est la taille des données en entrée, l'idée derrière le concept d'"*optimalité*" consiste à savoir s'il n'existe pas une manière de faire mieux que $\mathcal{A}$ (en d'autres termes, on se pose la question – légitime – est-ce-que $\mathcal{A}$ serait ce qui se fait de mieux en la matière ?).

Considérons le problème du tri.

1.1 L'exemple de Quicksort

L'algorithme du *tri-rapide* ou *Quicksort* a été inventé par HOARE en 1961. Il est basé sur le concept algorithmique *diviser pour régner*.

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui lui sont inférieurs soient à sa gauche et que tous ceux qui lui sont supérieurs soient à sa droite. Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Le pseudo-code est donné ci-dessous

```
tri_rapide(tableau t, entier premier, entier dernier)
    debut
      si premier < dernier alors
        pivot := choix_pivot(t,premier,dernier)
        pivot := partitionner(t,premier,dernier,pivot)
        tri_rapide(t,premier,pivot-1)
        tri_rapide(t,pivot+1,dernier)
      fin si
    fin
```

où l'algorithme de partitionnement est le suivant

```
partitionner(tableau T, premier, dernier, pivot)
     echanger T[pivot] et T[dernier]
     j := premier
     pour i de premier a dernier - 1
         si T[i] <= T[dernier] alors
             echanger T[i] et T[j]
             j := j + 1
     echanger T[dernier] et T[j]
     renvoyer j
```

Le partitionnement peut être fait en temps linéaire, en place. La mise en œuvre la plus simple consiste à parcourir le tableau du premier au dernier élément, en formant la partition au fur et à mesure : à la i-ème étape de l'algorithme ci-dessous, les éléments $[0], \cdots, [j-1]$ sont inférieurs au pivot, tandis que $[j], \cdots, [i-1]$ sont supérieurs au pivot.

1.2 Analyse de l'algorithme Quicksort

Pour analyser l'algorithme, on observe tout d'abord que pour pouvoir trier on doit faire des comparaisons et des déplacements d'éléments.

Proposition. Dans le *pire cas*, Quicksort nécessite un nombre quadratique $\Theta(n^2)$ de déplacements.

Preuve. On peut supposer que le tableau est trié à l'envers. Dans le pire cas, le choix du pivot se porte sur un élément extrême (celui du début du tableau ou celui de la fin). Dans ce cas, on déplace tous les autres éléments et on recommence d'où

$$\text{Nbr déplacements}(n) = (n-1) + \text{Nbr déplacements}(n-1) = \sum_{k=1}^{n-1} \quad = \Theta(n^2)\,.$$

Pour pouvoir faire une analyse en *moyenne* de l'algorithme, il est nécessaire de définir une *distribution de probabilité* sur les données en entrée. Dans notre cas, on se limite à la *distribution uniforme* : chacune des $n!$ permutations des n entiers distincts a la même probabilité $\frac{1}{n!}$ d'être le tableau à trier. De plus, on fait l'*hypothèse* supplémentaire que le pivot est choisi uniformément aléatoirement sur le sous-tableau courant à traiter (dans les appels récursifs). On a alors le résultat suivant.

Proposition. Si la distribution des tableaux d'entiers à trier est la distribution uniforme alors Quicksort nécessite en moyenne $O(n \log n)$ **comparaisons**.

Preuve. Le nombre moyen (espérance) du nombre de comparaisons est donné par la formule (expliquer !)

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \mathbb{P}\left[\text{les éléments aux positions i et j sont comparés}\right] .$$

Nous devons donc étudier la probabilité que deux éléments aux positions i et j soient comparés. La comparaison arrive si l'un de ces deux éléments est choisi comme pivot à un moment de l'algorithme. A tout moment, cet évènement arrive avec une probabilité

$$\underbrace{(\quad 1 + 1 \quad)}_{\text{1 fois pour i, l'autre pour j}} \times \frac{1}{\text{taille du sous-tableau courant}}$$

puisque le pivot est choisi aléatoirement uniformément. Mais la taille d'un tableau contenant à la fois les éléments d'indices i et j est au moins $j - i + 1$. Donc le nombre moyen recherché est au plus

$$\begin{aligned}
\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \mathbb{P}\left[\text{les éléments aux positions i et j sont comparés}\right] &\leq \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \frac{2}{j+i-1} \\
&\leq \sum_{i=0}^{n-2} 2\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right) \\
&\leq O(n \log n)\,.
\end{aligned}$$

Remarque. Il est tout à fait possible de calculer le nombre moyen de **déplacements** effectués par l'algorithme. Pour simplifier le calcul, on choisit le pivot comme le milieu du tableau courant à traiter. De plus, on peut supposer que n est une puissance de 2. Si ce n'est pas le cas de toute faqn nous avons toujours l'existence d'un tel que $2^k \leq n < 2^{k+1}$ et on peut raisonner sur pour avoir les ordres de grandeur en grand-O.

Si $D(n)$ est le nombre moyen de déplacements pour un tableau de taille n, nous avons récursivement

$$\begin{aligned}
D(n) &= O(n) + 2D\left(\frac{n}{2}\right) \\
&= O(n) + 2 \times \left[O\left(\frac{n}{2}\right) + 2\,D\left(\frac{n}{4}\right)\right] \\
&= \cdots
\end{aligned}$$

et on montre que $D(n) = O(n \log n)$ (on peut détailler !).

1.3 La méthode dite de l'adversaire

La question principale porte sur l'**optimalité** ou **non** d'un algorithme de tri. Pour le tri, on se pose des questions comme :
EXISTE-T-IL UN ALGORITHME DE TRI DONT LA COMPLEXITÉ (EN NBR. DE COMPARAISONS ET/OU DÉPLACEMENTS) DANS LE PIRE DES CAS (RESP. EN MOYENNE) EN BEAUCOUP MOINS QUE $\mathbf{O(n \log n)}$?

PEUT-ON FAIRE UN TRI NÉCESSITANT UN NOMBRE LINÉAIRE $O(n)$ DE COMPARAISONS ?

PEUT-ON PROUVER UN **THÉORÈME** DU GENRE :
Il existe une distribution de probabilité sur les entrées et une fonction (n) telles que tout algorithme de tri nécessite un nombre moyen de $O(\ (n))$ comparaisons sur cette distribution.

La méthode dite de l'adversaire consiste à piéger l'algorithme à l'aide d'un adversaire