TD3 : analyse de complexité

1 Notations pour l'étude de complexité

1.1 Éléments de cours

Nous avons défini en cours les notations O et Ω. Ces notations servent principalement à simplifier les calculs de complexité en se concentrant sur ce qui est important :

— nous travaillons à un facteur près
— c'est le comportement dominant qui nous intéresse, nous pouvons négliger certains termes
— nous étudions le comportement pour n grand, et nous fichons des petites valeurs particulières

borne supérieure O **:** la notation O permet de définir un majorant. Lorsque l'on écrit que $f = \mathrm{O}(g)$, cela signifie que la fonction f est *dominée* par g (mais pas forcément négligeable devant g, qui s'écrirait $f = \mathrm{o}(g)$). Ainsi, lorsque le paramètre n indiquant la taille des données est suffisamment grand (à partir d'un certain n_0), nous avons $f(n) \leq \alpha g(n)$ pour un certain α (la constante) strictement positif. De façon plus concise et formelle :

$$\exists n_0 \geq 0, \exists \alpha > 0, \forall n \geq n_0, f(n) \leq \alpha g(n)$$

borne inférieure Ω **:** la notation Ω permet de définir un minorant. Lorsque l'on écrit que $f = \Omega(g)$, cela signifie que la fonction f *domine* g. Ainsi, lorsque le paramètre n indiquant la taille des données est suffisamment grand (à partir d'un certain n_0), nous avons $f(n) \geq \alpha g(n)$ pour un certain α (la constante) strictement positif. De façon plus concise et formelle :

$$\exists n_0 \geq 0, \exists \alpha > 0, \forall n \geq n_0, f(n) \geq \alpha g(n)$$

encadrement Θ **:** cette notation sert à dire que nous avons à la fois O et Ω, et ainsi qu'une fonction f et une autre g ont le même comportement dominant :

$$\exists n_0 \geq 0, \exists \alpha_1, \alpha_2 > 0, \forall n \geq n_0, \alpha_1 g(n) \leq f(n) \leq \alpha_2 g(n)$$

Notez que dans toutes les définitions ci-dessus, nous avons supposé que f et g étaient des fonctions positives.

1.2 Opérations, manipulation

Montrez que si f, g, f_1, f_2, g_1, g_2 sont des fonctions positives, alors :
— si $f = \Theta(g)$ et $\lambda > 0$, alors $\lambda f = \Theta(g)$;
— si $f_1 = \Theta(g_1)$ et $f_2 = \Theta(g_2)$, alors $f_1 + f_2 = \Theta(\max(g_1, g_2))$.

La première règle s'assure que nous travaillons à une constante près. La seconde signifie que si nous exécutons successivement deux algorithmes, la comportement de l'ensemble est de l'ordre du pire des deux algorithmes.

2 Tri par insertion (retour TD précédent)

Le tri par insertion consiste à trier le tableau en considérant que le début du tableau est trié et en insérant petit à petit les valeurs suivantes à leur place. Initialement, la portion triée du tableau est réduite à la seule première case.

Procédure TriInsertion
  données : tab : un tableau, n sa taille
  résultat : tab est trié
  Algorithme
    pour i allant de 1 à $n-1$ **faire**
      $j \leftarrow i$
      tant que $j > 0$ et $tab[j] < tab[j-1]$ **faire**
        échanger $tab[j]$ et $tab[j-1]$
        $j \leftarrow j - 1$

Étudiez le meilleur et le pire cas pour :
- le nombre de comparaisons entre éléments du tableau
- le nombre d'échanges d'éléments du tableau

2.1 Étude en moyenne (si le temps le permet)

Le tri par insertion vu dans l'exercice précédent n'a pas la même complexité dans le meilleur et le pire cas. Il est donc utile de savoir en général, sur un tableau quelconque, si la complexité s'approche plus du meilleur ou du pire cas. Une approche consiste à calculer la moyenne de la complexité sur toutes les entrées possibles (de taille n). On parle ainsi d'étude en moyenne. Ce genre d'étude peut s'avérer complexe, mais est abordable dans le cas du tri par insertion.

2.1.1 Position moyenne du dernier élément dans une permutation aléatoire

Considérons un ordre quelconque du tableau : chaque élément du tableau a la même probabilité de se situer dans chacune des cases du tableau. Sur un tableau de n cases, en imaginant que les $n-1$ premières cases ont été triées, quelle est la probabilité que le $n^{\text{ième}}$ élément soit plus petit que le premier ? Entre le premier et le second ? À sa place ? Quelle est donc sa position moyenne au sein des éléments précédents ?

2.1.2 Comportement en moyenne du tri par insertion

Quel est donc en moyenne le nombre d'itérations réalisées par la boucle « tant que » ? Qu'en déduisez vous sur la complexité totale (pour le nombre d'échanges) de l'algorithme de tri par insertion en moyenne ?