Ordres de Grandeur Asymptotiques

Objectif L'objectif du TP est d'illustrer les notions d'ordres de grandeur asymptotiques ($\mathcal{O}$, Ω, Θ) et d'observer la complexité asymptotique de certaines méthodes des structures de données des bibliothèques java.

Travail à rendre

- les fichiers source java dont Mesure.java complété pour la génération de tous les fichiers de données requis pour répondre aux questions du TP,
- tous les fichiers de données générés par l'exécution de votre code java,
- un unique fichier de commande gnuplot permettant la génération de tous les graphiques en une seule exécution,
- un rapport documentant vos tests et contenant les graphiques, leurs commentaires et les réponses aux questions.

Vous devrez compléter le `main` présent dans le fichier `Mesure.java`. Grâce à la structure `switch` toutes les exécutions que vous aurez eu besoin de faire pourront figurer dans votre version personnelle de `Mesure.java`.

1 Gnuplot

En guise de petit exemple pour démarrer, recopiez ces lignes dans un fichier texte `fichier.txt`.

```
1   1  10
2   5  25
3  25  30
```

Chaque ligne représente par exemple (au hasard) une taille de tableau suivie par un nombre de comparaisons et un nombre d'échanges.

Lancez `gnuplot` puis tapez :

```
plot 'fichier.txt' using 1:2 title 'comparaisons' with lines
```

La commande `plot` est la commande de production de graphique, `fichier.txt` est le fichier où se trouvent les données à afficher, `using 1:2` indique que les points servant à construire le graphique ont leur abscisse dans la première colonne et leur ordonnée dans la deuxième et enfin, `with lines` précise que le tracé doit être continu.

Pour superposer deux tracés sur un même graphique il suffit de séparer par une virgule leurs descriptions respectives directement dans la commande comme suit :

```
plot 'fichier.txt' using 1:2 title 'comparaisons' with lines, \
'fichier.txt' using 1:3 title 'echanges' with lines
```

Notez que le caractère \ vous permet de poursuivre la commande à la ligne suivante.

Pour sauvegarder les schémas il suffit de préciser avant de lancer la commande `plot` le format de sauvegarde de l'image (png, eps, etc) et le nom du fichier à créer, ce qui donne :

```
>set term png
>set output 'graphique.png'
>plot 'fichier.txt' using 1:2 title 'comparaisons' with lines, \
'fichier.txt' using 1:3 title 'echanges' with lines
```

Pour ne pas toujours retaper ces commandes, il est possible de copier ces lignes en les terminant par le caractère `';'` dans un fichier, `exemple.plot` par exemple, puis de l'executer grâce à

la commande `gnuplot exemple.plot`. Vous pouvez ajouter des commentaires dans votre fichier `exemple.plot` en les démarrant par le caractère `#`.

Vous pouvez ensuite admirer les résultats de vos mesures dans le fichier `graphique.png` qui vient d'être créé.

Un autre exemple vous est donné, pour le tester vous devez exécuter les instructions suivantes :

```
> javac *.java
> java -Xint Mesure 0 > cas0.txt
> gnuplot tpog.plot
```

Le graphique `lin_aleaT.png` doit maintenant apparaître dans le répertoire courant.

En suivant ce modèle vous pourrez produire tous les résultats demandés dans le TP.

2 Comparaison de méthodes java et d'algorithmes fait maison

Dans cette partie vous allez estimer puis comparer entre eux les temps d'exécution moyens de plusieurs algorithmes pour plusieurs tailles de données. Pour chaque algorithme vous disposez d'une version utilisant comme structure de données une ArrayList et d'une version utilisant un tableau, ce qui vous permettra de comparer les performances de ces deux structures.

Généralement lorsqu'on étudie analytiquement la complexité d'un algorithme on s'intéresse à ses pires et meilleurs cas et on essaie de décrire les données pour lesquelles ses complexités extrêmes sont atteintes. La complexité en moyenne est souvent beaucoup plus difficile à calculer mais il est possible de l'estimer expérimentalement.

En toute rigueur, pour obtenir le temps moyen exact sur une machine particulière pour une donnée de taille n, il faudrait mesurer les temps d'exécutions de l'algorithme sur *toutes* les données de taille n, puis faire la moyenne.

Par exemple pour un algorithme de tri de tableaux d'entiers, il faudrait exécuter l'algorithme sur *tous* les tableaux de n entiers, ce qui fait une infinité de possibilités. En restreignant l'ensemble des entiers à ceux que l'on peut coder de manière standard sur la machine il reste tout de même trop de cas pour qu'à partir d'une certaine taille de données il devienne impossible d'effectuer les mesures en une seule vie humaine.

C'est pourquoi dans ce TP vous ne ferez que des estimations sur des échantillons de données choisis aléatoirement.

Les différentes tailles de données à tester lors d'une mesure sont spécifiées par le paramètre `int[] tSize_` du constructeur de la classe `Mesure`. Le paramètre `int[] tRep_` sert quant à lui à fixer pour chaque taille de données le nombre d'exécutions total à effectuer.

2.1 Influence des données

Q1 . Comparer sur un même graphique les temps d'exécution des méthodes `aleaL` et `aleaT` avec un très faible nombre de répétitions pour chaque taille de données mesurée ainsi qu'avec un nombre de répétitions plus élevé, soit quatre courbes en tout pour ce premier graphique.

Q2 . Le nombre de répétitions a-t-il une influence sur les courbes ?

Q3 . Montrer que chaque permutation des éléments du tableau est équiprobablement obtenue par la méthode `aleaT` en admettant que la machine dispose d'un générateur de nombres aléatoires, et pas seulement pseudo-aléatoire.

Q4 . Comparer sur un même graphique les temps d'exécution des méthodes `rechercheL` et `rechercheT` avec un très faible nombre de répétitions pour chaque taille de données mesurée ainsi qu'avec un nombre de répétitions plus élevé, soit quatre courbes en tout pour ce deuxième

graphique.

Q5 . Le nombre de répétitions a-t-il une influence sur les courbes ?

Q6 . Que constatez-vous d'autre sur les huit mesures précédentes ?

Q7 . En observant le fonctionnement des algorithmes, comment expliquez-vous les variations d'influence du nombre de répétitions de la mesure pour les quatre méthodes testées ?

Q8 . Que pouvez-vous conclure de ce qui précède sur le choix d'un nombre de répétitions pertinent par rapport à un algorithme ?

2.2 Recherches de minima

Q9 . Comparer sur un même graphique les courbes correspondant aux deux méthodes `minimumSimpleL`, `minimumSimpleT`, `minimumTriL` et `minimumTriT`.

Q10 . Quelles observations pouvez vous faire sur les structures d'ArrayList et de tableau ?

Q11 . Quelle est votre conclusion sur la manière la plus efficace d'effectuer une recherche de minimum dans une ArrayList ou un tableau ?

3 Comparaisons des fonctions de référence

3.1 Vue d'ensemble

Q12 . Pouvez vous facilement tracer sur un même graphique les courbes des fonctions de référence données dans la classe `Mesure` ?

Q13 . Que pouvez-vous déduire de vos observations sur le tracé des courbes précédentes ? Si nécessaire n'hésitez pas à faire plusieurs graphiques pour appuyer votre réponse.

3.2 n^ε et $\log n$ pour $0 < \varepsilon$

Q14 . Tracez la fonction $\frac{\log n}{n^\varepsilon}$ pour les valeurs de $\varepsilon \in \{2, 1, 0.5, 0.2, 0.1\}$ sur un même graphique.

Q15 . Quelle relation - $\mathcal{O}$, Ω ou Θ - existe entre les deux fonctions $\log n$ et n^ε pour $\varepsilon > 0$?

3.3 n^k et 2^n pour $k \geq 1$

Q16 . Tracez la fonction $\frac{n^k}{2^n}$ pour plusieurs valeurs de $k \geq 1$ sur un même graphique.

Q17 . Quelle relation - $\mathcal{O}$, Ω ou Θ - existe entre les deux fonctions 2^n et n^k pour $k \geq 1$.

4 Détermination expérimentale de la complexité d'un algorithme

Dans cette section vous devez comparer les courbes de certains algorithmes donnés dans la classe `Methodes` avec les courbes des fonctions de référence afin de trouver la classe de complexité

à laquelle ils appartiennent.

Q18 . Que fait l'algorithme codé par les deux méthodes `mystereL` et `mystereT` ?

Q19 . D'après votre analyse du nombre d'opérations effectuées par chacun de ces deux algorithme dans un pire cas et un meilleur cas, à quelle(s) "classe(s)" de complexité appartiennent-ils ?

Q20 . Représentez à l'aide d'un graphique ce que vous avez obtenu par analyse : vous ferez apparaître sur un même graphique la courbe de l'algorithme (fonction f) et les courbes $c_1.g$ et $c_2.g$ où c_1 et c_2 sont des valeurs possibles pour les constantes de la définition de $f \in \Theta(g)$.

Pour déterminer les constantes c_1 et c_2 vous pourrez trouver utile la méthode `scaledRes` de la classe `Mesure` qui retourne un tableau contenant les valeurs présentes dans le tableau `tRes` multipliées par un même facteur.

5 Rappels : $\mathcal{O}$, Θ et Ω

5.1 Majoration : $\mathcal{O}$

La notation $f \in \mathcal{O}(g)$ signifie que la fonction f est majorée asymptotiquement par la fonction g, c'est-à-dire que pour des valeurs de $n \in \mathbb{N}$ suffisamment grandes $f(n)$ est toujours plus petit que $g(n)$ multiplié par une certaine constante. En d'autres termes la fonction f croît moins vite que la fonction g. La définition suivante est un peu plus formelle.

Definition 1 *Soient f et g deux fonctions de $\mathbb{N} \to \mathbb{N}$, $f \in \mathcal{O}(g) \Leftrightarrow \exists c > 0$ une constante et $n_c \in \mathbb{N}$ tels que $\forall n \geq n_c$, $f(n) \leq c.g(n)$.*

Dire qu'il existe une constante c telle que pour des valeurs de n assez grandes $f(n) \leq c.g(n)$ revient à dire que pour n assez grand le rapport $\frac{f(n)}{g(n)}$ est majoré par une constante : il ne peut pas croître au delà de la valeur c, en particulier quand n tend vers $+\infty$, la limite de ce rapport ne peut pas être $+\infty$. Ceci amène à une autre manière de montrer qu'une fonction f est majorée par une fonction g, il suffit de calculer la limite de $\frac{f(n)}{g(n)}$ quand n tend vers $+\infty$:

$\rightarrow$ Si $\lim_{n\to+\infty} f(n)/g(n) = k$ où k est une constante, alors $f \in \mathcal{O}(g)$.

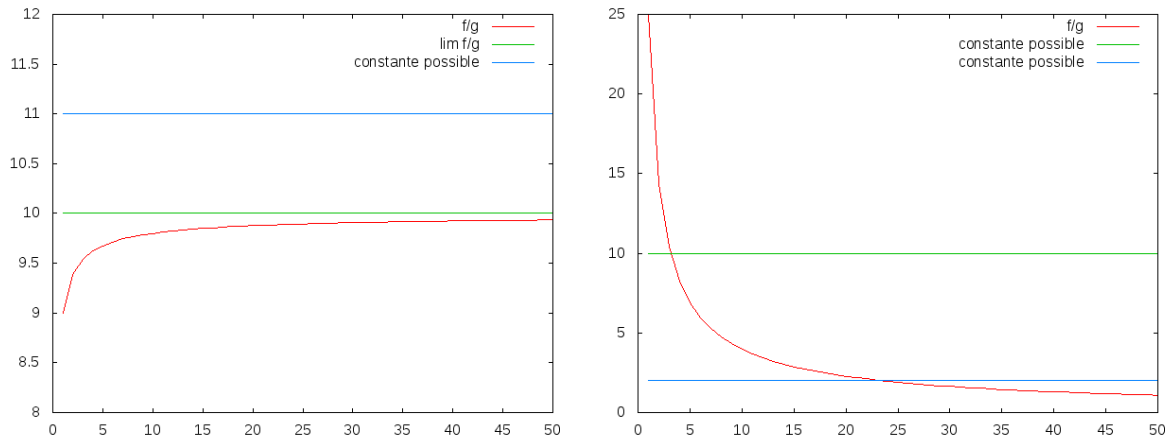

FIGURE 1 – Exemple de rapports f/g tels que $f \in \mathcal{O}(g)$.

5.2 Minoration : Ω

La notation $f \in \Omega(g)$ signifie que la fonction f est minorée asymptotiquement par la fonction g, c'est-à-dire que pour des valeurs de $n \in \mathbb{N}$ suffisamment grandes $f(n)$ est toujours plus grand que $g(n)$ multiplié par une certaine constante. En d'autres termes la fonction f croît plus vite que la fonction g.

Il est très facile de voir que si $f \in \Omega(g)$ alors $g \in \mathcal{O}(f)$ et vice versa. Ainsi la définition suivante n'est en fait que la définition de $f \in \mathcal{O}(g)$ dans laquelle les rôles de f et g sont inversés.

Definition 2 *Soient f et g deux fonctions de $\mathbb{N} \to \mathbb{N}$, $f \in \Omega(g) \Leftrightarrow \exists c > 0$ une constante et $n_c \in \mathbb{N}$ tels que $\forall n \geq n_c$, $c.g(n) \leq f(n)$.*

Comme la majoration, la minoration peut être traduite en terme de limite. Pour n assez grand le rapport $\frac{f(n)}{g(n)}$ est minoré par une constante $c > 0$: il ne peut pas décroître au delà de la valeur c, en particulier quand n tend vers $+\infty$, la limite de ce rapport ne peut pas être 0. Ainsi pour montrer qu'une fonction f est minorée par une fonction g, il suffit de calculer la limite de $\frac{f(n)}{g(n)}$ quand n tend vers $+\infty$:

- → Si $\lim_{n\to+\infty} f(n)/g(n) = k > 0$ où k est une constante, alors $f \in \Omega(g)$: ici on peut prendre $c \in]0, k[$ comme constante c de la définition.
- → Si $\lim_{n\to+\infty} f(n)/g(n) = +\infty$, alors $f \in \Omega(g)$: ici n'importe quelle constante $c > 0$ convient pour se ramener à la définition de $f \in \Omega g$.

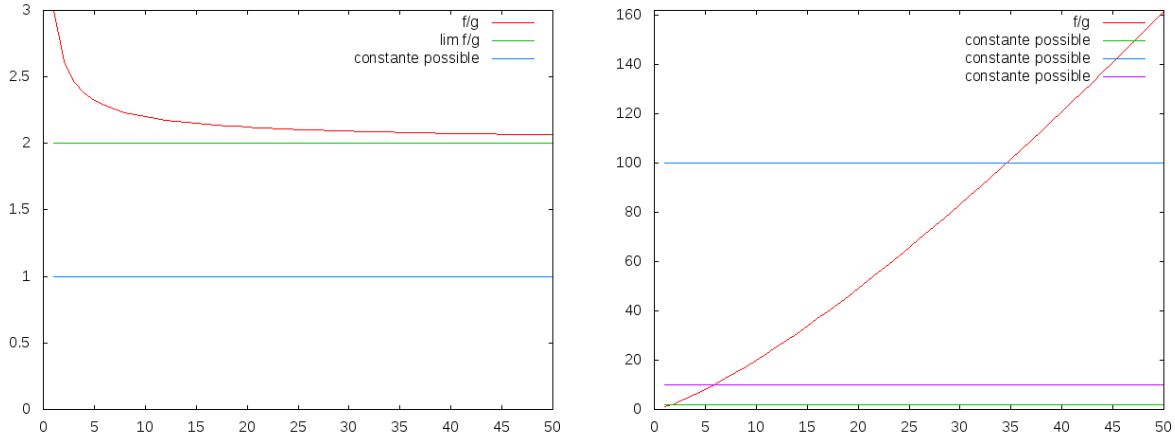

FIGURE 2 – Deux cas de rapport f/g tels que $f \in \Omega(g)$.

5.3 Équivalence : Θ

Deux fonctions f et g sont asymptotiquement équivalentes lorsqu'on a à la fois $f \in \mathcal{O}(g)$ et $f \in \Omega(g)$, ce qui revient aussi à $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(f)$. Intuitivement cela signifie que les deux fonctions croissent à une vitesse identique.

Definition 3 *Soient f et g deux fonctions de $\mathbb{N} \to \mathbb{N}$, $f \in \Theta(g) \Leftrightarrow \exists c_1, c_2 > 0$ deux constantes et $n_0 \in \mathbb{N}$ tels que $\forall n \geq n_0$, $c_1.g(n) \leq f(n) \leq c_2.g(n)$.*

Cette définition indique que f est à la fois minorée et majorée par g, et donc g est majorée et minorée par f. Ainsi $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$.

En terme de limite les définitions précédentes donnent les relations suivantes :

- $\lim_{n\to+\infty} f(n)/g(n) = 0 \Rightarrow f \in \mathcal{O}(g), f \notin \Omega(g)$
- $\lim_{n\to+\infty} f(n)/g(n) = +\infty \Rightarrow f \notin \mathcal{O}(g), f \in \Omega(g)$
- $\lim_{n\to+\infty} f(n)/g(n) = k \neq 0 \Rightarrow f \in \mathcal{O}(g), f \in \Omega(g) \Leftrightarrow f \in \Theta(g)$

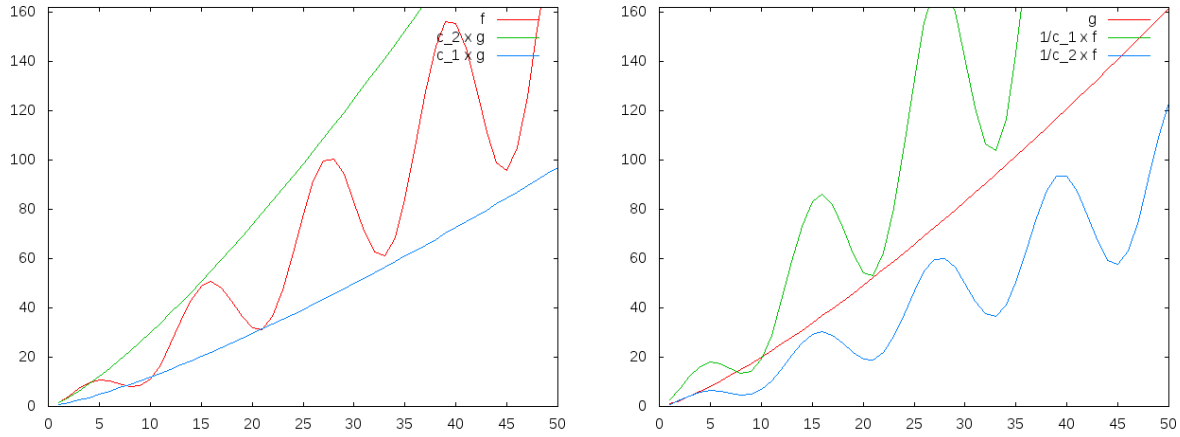

FIGURE 3 – La courbe de f est encadrée par les courbes de $c_1.g$ et $c_2.g$, inversement, la courbe de g est encadrée par les courbes de $\frac{1}{c_2}f$ et $\frac{1}{c_1}f$.