IFT436 - Série d'exercices #2 : la notation asymptotique

Manuel Lafond

Exercice 1: Donnez la notation O des fonctions suivante de façon aussi précise que possible. Justifiez vos réponses.

- $f(n) = 30n^2/n + 15n^{0.7} + 100\sqrt{n}$

  Solution. On montre que c'est $O(n)$:

  $$\begin{aligned} f(n) &= 30n^2/n + 15n^{0.7} + 100\sqrt{n} \\ &\leq 30n + 15n + 100n \\ &= 145n \end{aligned}$$

  Donc pour tout $n \geq 1$, $f(n) \leq 145n$, et donc $f(n) \in O(n)$.

- $f(n) = \sin(n)$

  Solution. Puisque $sin(n) \leq 1$ pour tout n, $sin(n) \in O(1)$.

- $f(n) = 2^{n+10}$

  Solution. $2^{n+10} = 2^{10} \cdot 2^n \in O(2^n)$ car 2^{10} est une constante.

- $f(n) = 2^n/1000 + n^{100} + n \log n$

  Solution. C'est $O(2^n)$. Voici une première version abrégée que j'accepterais comme réponse:

  Considérant que $n^{100} \in O(2^n)$ et $n \log n \in O(2^n)$, il existe n_0 et des constantes c, d telles que $f(n) = 2^n/1000 + n^{100} + n \log n \leq 2^n/1000 + c2^n + d2^n = 2^n(c + d + 1/1000) \in O(2^n)$.

  Voici une version un peu plus détaillée. On utilise le théorème donné en classe, qui disait en particulier que $n^{100} \in O(2^n)$. Ceci veut dire que à partir d'un certin n_0, on peut dire, pour une constante c, que

$$\begin{aligned} f(n) &= 2^n/1000 + n^{100} + n\log n \\ &\leq 2^n/1000 + c2^n + n\log n \end{aligned}$$

En utilisant le même théorème, on peut déduire que $n\log n \in O(2^n)$, et il y a donc une constante d et n_1 tels que pour tout $n \geq \max(n_0, n_1)$,

$$\begin{aligned} & 2^n/1000 + c2^n + n\log n \\ \leq & 2^n/1000 + c2^n + d2^n \\ = & 2^n(c + d + 1/1000) \end{aligned}$$

Puisque $c+d+1/1000$ est une constante, ceci démontre que $f(n) \in O(2^n)$.

- $f(n) = 1000/n$

  Solution. $O(1)$, puisque $1000/n \leq 1000$ pour tout $n \geq 1$, et 1000 est une constante.

- $(2^{\log n} + \sqrt{n}) \cdot \frac{1}{8}(\log(n^3) + \log(\log n))$

  Solution. C'est $O(n\log n)$. Appliquez le fait que $2^{\log n} = n$ et $\log n^3 = 3\log n$, calculez l'expansion et vous verrez que chaque terme est dominé par $n\log n$.

Exercice 2: Montrez que la base du log n'a pas d'importante en O. C'est-à-dire, montrez que $\log_b(n) \in O(\log n)$ pour tout $b > 1$.

Solution. Selon la règle du changement de base, $\log_b(n) = \log n/\log b$. Puisque $1/\log b$ est une constante, ceci montre que $\log_b(n) \in O(\log n)$.

Exercice 3: On va démontrer qu'ignorer les constantes est souvent bien, mais pas toujours.

a. Supposez que vous avez deux algorithmes de tri. Le premier exécute n^2 instructions et le deuxième en exécute $100n$. À partir de quel n est-ce que le deuxième algorithme est avantageux?

b. Supposez que vous avez deux algorithmes de tri. Le premier exécute $n\log n$ instructions et le deuxième en exécute $100n$. À partir de quel n est-ce que le deuxième algorithme est avantageux?

Je vous laisse le calculer (prenez e.g. wolframalpha.)

Exercice 4: Montrez que $\log(n!) \in \Theta(n \log n)$.
Solution. Notons que

$$\log(n!) = \log(n(n-1)\dots(2)(1)) = \log n + \log(n-1) + \log(n-2) + \ldots + \log(1)$$

Pour montrer que c'est $O(n \log n)$, on a

$$\log n + \log(n-1) + \log(n-2) + \ldots + \log(1) \leq \log n + \log n + \log n + \ldots + \log n = n \log n$$

Pour montrer que c'est $\Omega(n \log n)$, on garde les $n/2$ premiers termes de $\log n + \log(n-1) + \log(n-2) + \ldots + \log(1)$. On obtient

$$\begin{aligned} &\log n + \log(n-1) + \log(n-2) + \ldots + \log(n/2) \\ \geq &\log(n/2) + \log(n/2) + \log(n/2) + \ldots + \log(n/2) \end{aligned}$$

où $\log(n/2)$ est additionné $n/2$ fois. On a ensuite

$$\begin{aligned} &\log(n/2) + \log(n/2) + \log(n/2) + \ldots + \log(n/2) \\ =&(n/2)\log(n/2) \\ \geq&n/2(\log n)/2 \\ =&n/4 \log n \end{aligned}$$

le dernière inégalité étant vraie pour $n \geq 4$ parce que $\log(n/2) = \log n - 1 \geq (\log n)/2$ pour tout $n \geq 4$.
Nous avons donc montré que pour tout $n \geq 4$, $\log(n!) \geq n/4 \log n$, et donc $\log(n!) \in \Omega(n \log n)$.

Exercice 5: Considérez le problème de la sous-somme donné en devoir, mais avec la variante dans laquelle on cherche un sous-ensemble de exactement k éléments qui somment à un entier donné t. Un algorithme naïf va tester chaque combinaison de k éléments, ce qui prendra un temps au moins $\binom{n}{k}$. Quel temps cela prendra-t-il, en terme asymptotique?

Pour le savoir, montrez que $\binom{n}{k} \in \Theta(n^k)$ pour toute constante $k \in \mathbb{N}$.

Solution. Il est facile de voir que $\binom{n}{k} \leq n^k$ en utilisant $\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$. Donc $\binom{n}{k} \in O(n^k)$. Montrons que $\binom{n}{k} \in \Omega(n^k)$.
Puisque k est une constante, on choisit n_0 tel que pour tout $n \geq n_0$, on a $n - k + 1 \geq n/2$ (il est facile de voir qu'un tel n_0 existe, mais il ne nous est pas utile de le calculer). Lorsque $n \geq n_0$, on a

$$\begin{aligned}\binom{n}{k} &= \frac{n(n-1)\ldots(n-k+1)}{k!} \\ &\geq \frac{(n/2)(n/2)\ldots(n/2)}{k!} \\ &= \frac{(n/2)^k}{k!} \\ &= n^k \cdot \frac{1}{2^k k!}\end{aligned}$$

ce qui est $\Omega(n^k)$ puisque k est une constante (i.e. $\dfrac{1}{2^k k!}$ est aussi une constante).

Exercice 6: Il est facile de voir que $2 \cdot 2^n \in O(2^n)$. Par contre, on ne peut pas multiplier la base de l'exposant et rester dans le même O. Pour le voir, montrez que $4^n \notin O(2^n)$.
Si vous voulez aller un peu plus loin (un peu), montrez que pour toute constante réelle $c > 1$ et tout réel $\epsilon > 0$, $(c+\epsilon)^n \notin O(c^n)$.

Solution. Pour des fins de contradictions, supposons que $(c+\epsilon)^n \in O(c^n)$. Donc il existe $n_0 \in \mathbb{N}$ et $d \in \mathbb{R}^{>0}$ tels que pour tout $n \geq n_0$, $(c+\epsilon)^n \leq dc^n$. Soit α tel que $c^\alpha = c + \epsilon$. La valeur de α n'est pas importante, mais on sait que $\alpha > 1$. On a

$$(c+\epsilon)^n = (c^\alpha)^n = c^{\alpha n} \leq dc^n$$

où l'inégalité vient de notre supposition. En isolant d,

$$\frac{c^{\alpha n}}{c^n} \leq d \Rightarrow c^{n(\alpha-1)} \leq d$$

Puisque $\alpha > 1$, $c^{n(\alpha-1)}$ tend vers l'infini, ce qui contredit le fait que d est une constante. Donc $(c+\epsilon)^n \notin O(c^n)$.

Exercice 7: Montrez que $\log n \in O(\sqrt{n})$. En fait, montrez donc que $\log n \in O(n^\epsilon)$ pour toute constante réelle $\epsilon > 0$. Essayez sans utiliser la règle de l'Hôpital.
Indice: Utilisez les faits que $\log n < n$ pour tout $n \geq 2$ et $\log n = \log((\sqrt{n})^2)$. Ceci se généralise bien avec les ϵ.

Solution. On fait la version générale $\log n \in O(n^\epsilon)$. On a

$$\log n = \log((n^\epsilon)^{1/\epsilon}) = \frac{1}{\epsilon}\log(n^\epsilon)$$

On sait qu'il doit exister n_0 tel que pour tout $n \geq n_0$, on a $\log(n^\epsilon) < n^\epsilon$. Donc,

$$\log n = \frac{1}{\epsilon}\log(n^\epsilon) < \frac{1}{\epsilon}n^\epsilon$$

On a donc la constante $c = \frac{1}{\epsilon}$ et n_0 qui démontrent que $\log n \in O(n^\epsilon)$ pour tout $\epsilon > 0$.

Exercice 8: Considérez l'algorithme suivant qui décide si un élément r se trouve dans un tableau 2D.

```
fonction findMin(T, n, m, r) //n est la largeur, m est la hauteur
    r = ∞;
    pour i = 1..n faire
        pour j = 1..m faire
            si T[i][j] == r alors
                return true;
            fin
        fin
    fin
    return false;
```

Donnez la complexité de cet algorithme avec la notation Θ. Justifiez votre réponse.

Solution. C'est $\Theta(mn)$. Dans le pire cas, r n'est pas dans le tableau et il faut parcourir tous les éléments du tableau. Il y a mn tels éléments à parcourir, et le **if** sera donc évalué exactement nm fois.

Exercice 9: Considérez l'algorithme suivant qui prend en entrée deux mots P et T (i.e. des strings), et trouve une occurrence de P dans T, s'il y en a une.

```
fonction trouverMot(T, P) //T est le texte, P est le mot à trouver
    //|T| est le nombre de caractères de T;
    pour i = 1..|T| − |P| + 1 faire
        k = 1;
        tant que k ≤ |P| ∧ T[i + k − 1] == P[k] faire
            k + +;
        fin
        si k == |P| + 1 alors
            return k;
        fin
    fin
    return nil;
```

Vous devriez d'abord vous convaincre que cet algorithme fonctionne. Une fois que c'est fait, donnez la complexité de l'algorithme en terme de $|T|$ et de $|P|$ avec la notation O.
Quel est le pire cas, en terme de notation Ω? Ce n'est pas évident: la deuxième boucle sort dès qu'elle ne "match" pas un caractère de P. Pouvez-vous donner un exemple du pire cas?

Solution. L'algorithme prend un temps $O(|T||P|)$, car au pire on va s'arrêter sur chaque caractère de T et boucler sur k de 1 à $|P|$.
Pour montrer que c'est $\Omega(|T||P|)$, considérez $|P| = a^p b$ (i.e. le caractère a répété p fois, suivi d'un caractère b). Puis posez

$$T = a^{p-1}b$$

Quand $i = 1$, l'algorithme boucle sur k de 1 à $p - 1$.
Quand $i = 2$, l'algorithme boucle sur k de 1 à $p - 2$.
En général pour $i \in \{1, \ldots, p-1\}$, l'algorithme boucle sur k de 1 à $p - i$.

Le nombre de fois que la boucle **while** est évaluée est donc au moins

$$\begin{aligned}\sum_{i=1}^{p-1} p - i &= \sum_{i=1}^{p-1} p - \sum_{i=1}^{p-1} i \\ &= (p-1)p - (p-1)p/2 \\ &= (p-1)p/2 \\ &\geq (p/2)(p/2)\end{aligned}$$

Je vous laisse vérifier que $(p/2)(p/2) \in \Omega(|T||P|)$.
Notons aussi que l'on pourrait prolonger T en mettant $T = (a^{p-1}b)^q$ pour n'importe quel q. Est-ce que le temps serait quand même $\Omega(|T||P|)$?

Exercice 10: Considérez l'algorithme suivant, qui détermine s'il y a une stratégie gagnante pour les blancs aux échecs à partir de la configuration actuelle. Ici, T un tableau 2D 8×8 dont les valeurs indiquent quelle pièce se trouve sur la case, $b = true$ si c'est aux blancs de jouer et $false$ sinon. Aussi, V est la liste des configurations rencontrées jusqu'à maintenant.

fonction *analyserEchecs*(T, b, V)
  si *si T est présent au moins trois fois dans V* **alors**
    return $false$; //selon les règles des échecs, c'est égalité;
  si *les blancs sont échec et mat sur T* **alors**
    return $false$;
  si *les noirs sont échec et mat sur T* **alors**
    return $true$;
  pour chaque *mouvement possible m sur T* **faire**
    $T' =$ la configuration obtenue après avoir effectué M;
    $ret = analyserEchecs(T', \neg b, V \cup \{T'\})$;
    si $ret == true$ **alors**
      return ret;
  fin
  return $false$;

Croyez-vous que cet algorithme est correct? Justifiez informellement. Puis, donnez la complexité de cet algorithme avec la notation Θ.
Indice: il n'y a rien à calculer ici.

Solution.
L'algorithme prend un temps $\Theta(1)$.
L'algorithme teste toutes les configurations que l'on peut atteindre à partir de T. Il est donc correct. Ceci prendra un temps énorme et l'exécution ne se terminera probablement jamais (sinon, on saurait s'il y a effectivement une stratégie gagnante aux échecs).
Toutefois, le tableau a 64 cases (une constante) et le nombre de configurations possibles est donc borné par une constante. Même si cette constante est gargantuesque, ça reste $\Theta(1)$.

Exercice 11: (Défi) Montrez que $\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$ pour toute constante

$k \in \mathbb{N}$.

Solution. Pour voir que c'est $O(n^{k+1})$, notez que $\sum_{i=1}^{n} i^k \leq \sum_{i=1}^{n} n^k = n^{k+1}$ pour tout $n \geq 1$.
Montrons que c'est $\Omega(n^{k+1})$. On se débarasse des $n/2$ premiers termes de la sommation, puis on remplace les autres par $n/2$. Ceci donne

$$\sum_{i=1}^{n} i^k \geq \sum_{i=\lceil n/2 \rceil}^{n} i^k \geq \sum_{i=\lfloor n/2 \rfloor}^{n} \left(\frac{n}{2}\right)^k \geq \frac{n}{2}\left(\frac{n}{2}\right)^k = \left(\frac{n}{2}\right)^{k+1}$$

pour tout $n \geq 1$. Ceci est égal à $n^{k+1}\frac{1}{2^{k+1}}$, et puisque $\frac{1}{2^{k+1}}$ est une constante, c'est $\Omega(n^{k+1})$.

Exercice 12: Montrez que la notation O est transitive. C'est-à-dire, si $f(n) \in O(g(n))$ et $g(n) \in O(h(n))$, alors $f(n) \in O(h(n))$ pour toute fonctions $f, g, h \in \mathcal{F}$. ($\mathcal{F}$ est l'ensemble des fonctions des naturels vers les réels, voir notes de cours)

Solution. Si $f(n) \in O(g(n))$ et $g(n) \in O(h(n))$, il existe $n_0, n_1 \in \mathbb{N}$ et $c, d \in \mathbb{R}^{>0}$ tels que pour tout $n \geq n_0$, $f(n) \leq cg(n)$ et pour tout $n \geq n_1$, $g(n) \leq dh(n)$. Il s'ensuit que pour tout $n \geq \max(n_0, n_1)$, $f(n) \geq cg(n) \geq cdh(n)$. Puisque, $f(n) \geq cdh(n)$, on obtient $f(n) \in O(h(n))$.

Exercice 13: Un *arbre binaire plein* est une structure de données constituée de noeuds dans laquelle chaque noeud a soit 2 enfants, soit 0 enfants. Un noeud avec 0 enfants est appelé une *feuille*. Chaque noeud a aussi un parent, excepté la racine qui n'en n'a pas. Vous devriez avoir vu les arbres en structures de données.
La hauteur d'un arbre est le nombre de noeuds entre la racine et son noeud le plus éloigné. Montrez que si un arbre binaire plein a n feuilles, alors sa hauteur est au minimum $\log n$.
Indice: par induction sur la hauteur. Chaque enfant d'un noeud est la racine de son propre sous-arbre. Considérez le sous-arbre le plus feuillu.

Solution. On prouve par induction sur la hauteur.
Case de base: si la hauteur est de 1, l'arbre n'a qu'un noeud (et une feuille) et sa hauteur $1 \geq \log 1 = 0$.
Induction: on suppose pour tout arbre binaire plein avec $m < n$ feuilles a une hauteur au moins $\log m$. Soit T un arbre binaire plein de $n > 1$ feuilles. La racine r de T a deux enfants x et y. Il faut que x ou bien y ait au moins

$n/2$ feuilles, disons x (sans perdre de généralité). Par induction, le sous-arbre enraciné en x a une hauteur au moins $\log(n/2) = \log n - 1$. Avec la racine r, il s,ensuit que T a une hauteur au moins $\log n$.

Exercice 14: Il y a une erreur dans la preuve qui suit. Quelle est-elle?
Nous allons démontrer que $n^2 \in O(n)$ en prouvant que pour tout $n \geq 1$, il existe une constante c telle que $n^2 \leq cn$.
Case de base: $n = 1$. Dans ce cas, $n^2 = 1 \leq cn$ avec $c = 1$.
Induction: on suppose que pour tout $m < n$, il existe une constante c telle que $m^2 \leq cm$. On a

$$n^2 = (n-1)^2 + 2n - 1 \leq c(n-1) + 2n - 1 = (c+2)n - c - 1 \leq (c+2)n$$

où nous avons utilisé l'hypothèse d'induction pour la première inégalité. Donc il existe une constante $\hat{c} = c + 2$ telle que $n^2 \leq \hat{c}n$, démontrant que $n^2 \in O(n)$.

Solution. Le problème est qu'il faut fixer la constante et garder la même pour tous les $n \geq n_0$. La preuve ci-haut utilise une nouvelle constante $c + 2$ dans l'induction.

Exercice 15: Ordonnez les fonctions suivantes avec la notation O, de celle qui grandit le moins rapidement à celle qui grandit le plus rapidement. Il est possible que certaines aient le même taux de croissance, si elle sont Θ l'une de l'autre.

$$n \log n \qquad n^{10} \qquad n^{1.1} \qquad 1.1^n \qquad n^2/\log n \qquad (n^2 + n - 2)^5$$

$$O(n \log n) \subset O(n^{1.1}) \subset O(n^2/logn) \subset O(n^{10}) = O(n^2 + n - 2)^5 \subset O(1.1^n)$$

Exercice 16: Ordonnez les fonctions suivantes avec la notation O, de celle qui grandit le moins rapidement à celle qui grandit le plus rapidement. Il est possible que certaines aient le même taux de croissance, si elle sont Θ l'une de l'autre.

$$n! \qquad (n+1)! \qquad 2^n \qquad 2^{2n} \qquad n^n \qquad n^{\sqrt{n}} \qquad n^{\log n}$$

La règle des limites est sans doute la façon la plus simple d'ordonner ces fonctions. On a

$$O(n^{\log n}) \subset O(n^{\sqrt{n}}) \subset O(2^n) = O(2^{n+1}) \subset O(2^{2n}) \subset O(n!) \subset O((n+1)! \subset O(n^n)$$