TD1 : Complexité algorithmique

TOP : Techniques and tOols for Programming – 1A

★ Exercice 1: Complexité asymptotique et Faisabilité pratique.

▷ **Question 1:** Complètez les tableaux suivants. Dans le tableau (a), il s'agit de calculer le nombre d'opérations nécessaire pour exécuter l'algorithme en fonction de sa complexité et de la taille d'instance du problème traité. Dans le tableau (b), il s'agit de calculer le temps nécessaire à cela en supposant que l'on dispose d'un ordinateur capable de réaliser 10^9 opérations par seconde. Dans le tableau (c), on calcule la plus grande instance de problème traitable dans le temps imparti.

On rappelle que une nanoseconde (ns) vaut 10^{-9}s et que une microseconde (μs) vaut 10^{-6}. De plus, $10^{x^y} = 10^{x \times y}$ et $\sqrt{n} = n^{1/2}$.

(a) Nombre d'opérations.

Complexité	n=10	n=100	n=1000
n	10^1	10^2	10^3
n^2	10^2	10^4	10^6
n^3	10^3	10^6	10^9
2^n	10^3	10^{30}	10^{300}
$\sqrt{n}$	3.3	10	31
$\sqrt[3]{n}$	2.15	4.64	10
$\log_2(n)$	3.3	6.6	9.9

(b) Temps nécessaire à 10^9 op/sec.

Complexité	n=10	n=100	n=1000
n	10ns	100ns	1μs
n^2	100ns	10μs	1ms
n^3	1μs	1ms	1s
2^n	$>1\mu$s	(note 1)	10^{284} ans
$\sqrt{n}$	3.3ns	10ns	31ns
$\sqrt[3]{n}$	2.15ns	4.64ns	10ns
$\log n$	3.3ns	6.6ns	9.9ns

Réponse

(note 1) : 30000 milliards d'années, 2300 fois l'âge de l'univers (estimé à 13.7 milliards d'années).

Fin réponse

(c) Plus grande instance faisable à 10^9 op/sec.

Complexité	1s	1h	1 an
n	10^9	10^{12}	10^{16}
n^2	$3 \cdot 10^4$	10^6	10^8
n^3	10^3	10^4	$2 \cdot 10^5$
2^n	29	39	53
$\sqrt{n}$	10^{18}	10^{24}	10^{32}
$\sqrt[3]{n}$	10^{27}	10^{36}	10^{48}
$\log(n)$	$10^{30\,000\,000}$	∞	∞

Réponse

1s=10^9 opérations. Donc l'algo n fait 10^9 opérations, $n^2 \rightsquigarrow \sqrt{(10^9)}$, $2^n \rightsquigarrow \log_2(10^9)$, $\log n \rightsquigarrow 2^{10^9}$

1h, c'est idem $\times$ 3600. En gros, on calcule le nb d'op sur la première ligne, et après on extrait des racines carrés, des racines cubiques, des logs et des exp.

$2^{(10^9)} = (2^{100})^{10^7} \approx (10^{30})^{10^7} = 10^{300,000,000}$

A comparer au nombre estimé de particules dans l'univers : 10^{80}...

Fin réponse

CORRECTION CORRECTION

★ **Exercice 2:** Donnez la complexité des programmes suivants. Vous donnerez une borne supérieure avec un $O()$ dans un premier temps, puis vous affinerez votre calcul en utilisant la notation $\Theta()$.

listing 1
```
pour i = 1 à n faire
   pour j = 1 à n faire
      x += 3
```

listing 2
```
pour i = 1 à n faire
   pour j = 1 à i faire
      x += 3
```

listing 3
```
pour i = 5 à n-5 faire
   pour j = i-5 à i+5 faire
      x += 3
```

listing 4
```
pour i = 1 à n faire
    pour j = 1 à n faire
        pour k = 1 à n faire
            x := x+a
```

listing 5
```
pour i = 1 à n faire
    pour j = 1 à i faire
        pour k = 1 à j faire
            x := x+a
```

listing 6
```
for (i = n; i>1; i = i/2)
   for (j=0;j<i;j++)
      x := x+a
```

Réponse

Le listing 6 a des for(; ;) car on divise i par 2 à chaque étape. C'est impossible à écrire en pseudo-pascal, il me semble.

1. $\Theta(n^2)$ trivialement (gérald : ca veut dire $0(n^2)$ et $\Omega(n^2)$ en même temps ;)
2. $O(n^2)$ facilement (n est une borne sup du nombre de tours de boucles)
   $\Theta(n^2)$ un peu plus péniblement (en dénombrant exactement les étapes comme on a fait en cours pour le tri sélection qui ressemble beaucoup)
3. $O(n)$ la boucle interne a tjs un nombre constant d'opérations (10 étapes). Les variations de longueur sur la boucle externe étant constantes, on les ignore.
4. $\Theta(n^3)$ trivialement
5. $O(n^3)$ facilement car les boucles internes font au plus n tours.

   En dénombrant, on trouve $\sum_{i\in[1,N]}\sum_{j\in[1,i]}\sum_{k\in[1,j]} = \sum_{i\in[1,N]}\sum_{j\in[1,i]} j = \sum_{i\in[1,N]} \frac{i(i+1)}{2} = \frac{1}{2}\left(\sum_{i\in[1,N]} i^2 + \sum_{i\in[1,N]} i\right)$
   $= \frac{1}{2}\left(\frac{(2N+1)(N+1)N}{6} + \frac{N(N+1)}{2}\right) \rightsquigarrow \Theta(n^3)$
6. $O(n\log_2(n))$ assez trivialement, une fois qu'on sait que diviser le travail à chaque étape introduit du log dans les coûts. L'an prochain, je vous dirais comment le montrer. Je me souviens juste que c'est pas si dur...
   Mais en fait, ce code est dans $\Theta(n)$ (et donc aussi dans $O(n)$; Gérald, t'es pas attentif !). C'est parce que la boucle intérieure s'arrête à i et pas à n. Donc, si on compte les étapes faites par j dedans, on trouve : 1+2+4+8+...+n.
   Autrement dit, si on pose $n = 2^p$,

$$\sum_{k=1}^{p}\sum_{j=1}^{2^k} 1 = \sum_{k=1}^{p} 2^k = \frac{2^{p+1}-1}{2} = n - \frac{1}{2} \in \Theta(n)$$

   Ben oui, y'a quand meme des fois où c'est plus coquin, les décomptes de complexité, hein.

Fin réponse

★ **Exercice 3: Cas favorable, défavorable. Coût moyen.**

▷ **Question 1:** Étudiez le nombre d'additions réalisées par les algorithmes suivants dans le meilleur cas, le pire cas, puis dans le cas moyen en supposant que les tests ont une probabilité de $\frac{1}{2}$ d'être vrai.

▷ **Question 2:** Donnez une instance du code 2 de coût t_{avg}.

code 1
```
pour i de 1 à n faire
   si T[i]>a alors
      s := s + T[i]
```

code 2
```
si a > b alors
  pour i = 1 à n faire
     x := x+a
sinon x := x+b
```

Réponse

Question 4

code 1 : 0 au mieux, n au pire, n/2 en moyenne.

code 2 : 1 au mieux, n au pire. Donc, en moyenne, on a aussi n/2

Question 5 Y'en a pas, bien sûr. Dieu que je suis machiavélique.

Fin réponse

CORRECTION

★ **Exercice 4: Un peu de calculabilité.**

▷ **Question 1:** Démontrez que tous les algorithmes de tri comparatif sont dans $\Omega(n \log n)$.
Indice : Il faut repartir de la spécificiation du problème, dénombrer le nombre de solutions candidates, et quantifier la somme d'information accumulée lors de chaque test. Cela permet de calculer la borne inférieure de tests à réaliser pour sélectionner la bonne solution parmi les candidates.

Cet exercice est hors sujet vis-à-vis des évaluations, mais faut rester *challenging* :)
Il faut donc repartir de la spec du problème :
- INPUT : tableau d'éléments
- OUTPUT : permutation sur les éléments telle que ...

Combien de permutations sont possibles ? $n!$ bien sûr. On cherche donc la bonne permutation parmis $n!$ existantes.

Les algos comparatifs basent leur décision sur des comparaisons (si, si). Donc, à chaque étape, ils gagnent une information booléenne. Donc, en 3 étapes, ils peuvent trouver le bon élément dans un ensemble de 2^3 possibles.

Si on a un algo comparatif (correct) qui répond en $f(n)$ étapes, on est donc sûr que $2^{f(n)} > n!$ car s'il avait pas assez d'info, il pourrait pas trouver la bonne permutation.

Donc, $f(n) > log(n!)$. Hors, les taupins dans la salle devraient savoir montrer en utilisant l'approximation de Stirling que $\log(n!) \in \Omega(n \log n)$

Donc un algo comparatif correct est forcément dans $\Omega(n \log n)$. Voyez, les preuves de calculabilité sont pas forcément infaisables.

Fin réponse

★ **Exercice 5: Tri par dénombrement** [Seward 1954].

Si on sait que les valeurs sont comprises entre 0 et *max* (avec *max* pas trop grand), on peut trier les valeurs en comptant tout d'abord le nombre de 0, le nombre de 1, le nombre de 2 ... le nombre de *max* en entrée. Ensuite, il suffit de parcourir le tableau à nouveau en indiquant la bonne quantité de chaque valeur.

▷ **Question 1:** Écrire cet algorithme. On utilisera un tableau annexe *count* où $count[i]$ indique le nombre de i dans le tableau initial.

Réponse

```
for (i=0;i<length;i++)
  count[ tab[i] ]++;
j=0
for (u=0;u<max;u++)
  for (v=0;v<count[u];v++)
    tab[j] = u
    j++
```

Fin réponse

▷ **Question 2:** Calculer la complexité asymptotique de cet algorithme.

Réponse

La première boucle a clairement *len* tours.

Pour le second nid de boucle, c'est agacant à calculer au premier abord. Le plus simple est de compter le nombre de valeurs successives que la variable j prend : pile-poil *len*.

Du coup, on est en $\Theta(n + n) = \Theta(n)$. Ben oui, c'est linéaire...

Fin réponse

▷ **Question 3:** Discutez cette complexité par rapport à la borne théorique inférieure démontrée à l'exercice précédent.

Réponse

Bon, la question est pas claire, mais j'arrive pas bien à le dire sans que l'énoncé de la question 6 spoile le travail de la question 5.

Nous avons vu en cours que la complexité des algorithme de tri basés sur la comparaison des éléments est $\Omega(n \log n)$, mais il est bien possible que le fait que ce soit que pour les tris comparatifs soit un peu passé à la trappe. Là, on fait mieux car on ne compare pas les éléments entre eux, mais on utilise une connaissance supplémentaire sur les données (le fait que le max est petit).

CORRECTION

Remarquons également qu'on est plus en $\Theta(1)$ en mémoire, mais en $\Theta(max)$.

Ce résultat est amusant vis-à-vis de l'exercice précédent, mais ce n'est pas si choquant que le tri par dénombrement aille plus vite que la borne inférieure des tris comparatifs vu qu'il acquiere ses infos par un autre moyen que les comparaisons.

Fin réponse

CORRECTION

CORRECTION