Université de Lille

FST- Département Informatique

Algorithmique et ComplexiTé

2018–2019
Master d'Informatique-S1
TD 1

Rappels sur la complexité et correction des algorithmes

Ordres de grandeur

Exercice 1 : Histoire d'O et de Θ ?

Pour chacune des affirmations suivantes, dire si elle est vraie ou fausse :

Q 1. $n \in O(n^2)$?
Q 2. $n^2 \in O(n)$?
Q 3. $n \in \Theta(n^2)$?
Q 4. $\log n \in O(n)$?
Q 5. $\log n \in \Theta(n)$?
Q 6. $n \in O(n \log n)$?
Q 7. $n \log n \in \Theta(n)$?
Q 8. $n \in O(n + \log n)$?
Q 9. $n + \log n \in O(n)$?
Q 10. $4n^2 \in O(n)$?
Q 11. $100n^2 + n + 5 \in O(n^2)$?
Q 12. $n^2 + n + n \log n \in \Theta(n^2)$?
Q 13. $(\log n)^2 \in O(n)$?
Q 14. $2^n \in O(3^n)$?
Q 15. $3^n \in O(2^n)$?

Exercice 2 : A classer

On veut trier des fonctions par "ordre" asymptotique de grandeur : si $f = o(g)$, f sera "avant" g pour cet ordre. On notera que $f \equiv g$ si $f = \Theta(g)$ et $f(n) \ll g(n)$ si $f = o(g)$. Trier les fonctions suivantes :

$$n \quad 3^n \quad n \lg n \quad \lg n \quad \lg \lg n \quad n^2 \quad 2^{\ln n} \quad n\sqrt{n} \quad n^n \quad n! \quad 2^n$$

Pour mémoire :
— $\lg n = \log_2 n$, $\ln n = \log_e n$, $\lg \lg n = \lg(\lg n)$.
— $\log_b c = \log_b d * \log_d c$, $\log_b(a^c) = c \log_b a$.
— $a^{bc} = (a^b)^c = (a^c)^b$.

Comparer les complexités

Exercice 3 : D'un algorithme à l'autre

Q 1. *Pour le meilleur et pour le pire*
Q 1.1. La complexité exacte dans le pire des cas de A est n^2, celle de B est en n. Peut-on en déduire que B est plus rapide que A sur certaines données ? Sur toutes les données ?
Q 1.2. Que peut-on dire si la complexité exacte dans le meilleur des cas de A est n^2, celle dans le pire des cas de B est n ? Si la complexité exacte dans le meilleur des cas de A est n, celle dans le pire des cas de B est n^2 ?
Q 1.3. Si un algorithme A a une complexité dans le meilleur des cas en $\Theta(n)$ et une complexité dans le pire des cas en $\Theta(n)$, qu'en déduire pour sa complexité en moyenne ?
Q 2. *Ne négligeons pas trop vite les constantes !*
On suppose que la complexité exacte de l'algorithme A est $10^6 n^2$, celle de B n^3.
Q 2.1. A partir de quelle taille de données A est-il plus rapide que B ?
Q 2.2. Si on suppose qu'une instruction s'exécute en $1\mu s$, cela correspond approximativement à quel temps d'exécution ? Même question pour $10ns$?
Q 3. Un algorithme prend 1 seconde sur votre machine pour traiter une donnée de taille n_0. Vous échangez votre machine contre une machine deux fois plus rapide. Quelle est la taille maximale d'une donnée pouvant maintenant être traitée en 1 seconde si vous supposez que le temps d'exécution de l'algorithme est proportionnel à n ? proportionnel à n^2 ? proportionnel à $\log n$? proportionnel à 2^n ?

Analyse d'algorithmes.

Exercice 4 : *Quelques basiques*

Q 1. Pour chacun des algorithmes suivants, dire si sa complexité (temporelle) dans le pire des cas est en $\Theta(log\ n)$, $\Theta(n)$, $\Theta(n\ log\ n)$ ou $\Theta(n^2)$, en justifiant brièvement :

```
// n entier >=0
int myst1 (Int n) {
  int   r=0;
  for (i=n; i>=1; i--)
     for (j=i; j <=n; j++) r++;
  return r; }

// n entier >=0
int myst2 (Int n) {
  res=0;
  for (i=n; i>1; i=i div 2) res=res+1;
  return res;}

// n entier >=0
int myst3(int n){
  int c=0;
  for (int j=0;j<3;j++)
    for (int i=n;i>0;i=i-4)  c++;
  return c;}
```

```
// n entier >=0
int myst4(int n){
  int c=0;
  for (int i=1;i<n;i=4*i) c++;
  return c;}

// n entier >=0
int myst5(int n){
  int c=0;
  for (int i=0;i<n;i++)
    for (int j=1;j<i;j=j+3) c++;
  return c;}

// n entier >=0
int myst6(int n){
  int c=0;
  for (int i=n;i>0;i--)
    for (int j=i; j<i+3;j++) c++;
  return c;}
```

Q 2. *Analyse d'algorithmes récursifs.* Pour chacun des algorithmes récursifs suivants, estimer l'ordre de grandeur de la complexité temporelle en évaluant l'ordre de grandeur du nombre d'appels récursifs effectués

```
int A1(int n){
  if (n>1) return A1(n-1)
  else return 1; }
```

```
int A2(int n){
  if (n>1)  return A2(n-2) ;
  else return 1;}
```

```
int A3(int n){
  if (n>1)  return A3(n div 2 )
  else return 1;}
```

Q 3. *Vision globale (cf QCM)* Evaluer l'ordre de grandeur de la complexité de la fonction ci-dessous :

```
//n entier >0, t tableau  de n entiers
void balayage_inutile(){
  int j=0;
  for (int i=0; i<=n-1; i++){
     while ((j<n) && (t[j]==t[i]))) j++;}
}
```

corr$\Theta(n)$: le j est incrémenté au plus n fois.

Exercice 5 : *Multiplication et Exponentiation.*

Soit le problème suivant :
Entrée : n et k deux entiers positifs
Sortie : n^k

Q 1. Quelle est la taille de l'entrée en fonction de n et k ?

Q 2. *D'un coût à l'autre...*
Soit l'algorithme suivant :

```
product=1;
for (i=1;i<=k;i++) product=product *n;
```

Q 2.1. En *coût uniforme*, i.e. si on estime le coût d'une multiplication est $\Theta(1)$, quelle que soit la taille de ses opérandes, quelle est la complexité de l'algorithme ci-dessus ? Est-il polynomial ?

Q 2.2. On suppose maintenant que le modèle est celui du *coût logarithmique*, i.e. que les opérations élémentaires sont les opérations bit à bit. Si l'algorithme utilisé pour la multiplication est "l'algorithme de l'école primaire", quel est le coût de la multiplication d'un

nombre à x bits par un nombre à y bits ?

Calculer alors le coût de l'algorithme.

Q 3. Quels algorithmes pour la multiplication de deux entiers connaissez-vous ?

Q 4. *On trouve de tout sur le Web.*

Sur un site Web consacré à RSA , on trouve l'algorithme suivant (à peu de choses près) pour calculer l'exponentiation de deux entiers n et k :

```
if (k mod 2==0) res=1; else  res=n;
k=k div 2;
for (i=1;i<=k;i++) {
     p=n*n;
     res= res *p;}
```

L'algorithme est-il correct ? Que pensez-vous de sa complexité ? Pouvez-vous l'améliorer ?

Défis de la semaine

Voici des petits défis algorithmiques. Vous pouvez soumettre les solutions des 3 premiers sur la plate-forme.

Exercice 6 : Atteindre la cible

Soit T un tableau trié de n nombres distincts et un entier cible c. Proposez un algorithme en $O(n)$ qui détermine si il existe deux éléments x et y de T tels que $x + y = c$. Justifiez que l'algorithme est correct.

Exercice 7 : Equilibre

Soit un tableau dont les éléments sont 0 ou 1. Une tranche [i..j] d'éléments consécutifs est équilibrée si elle contient autant de 0 que de 1. On cherche la longueur maximale d'une tranche équilibrée. Pouvez-vous proposer un algorithme en $O(n^3)$? en $O(n^2)$? en $O(n)$?.

Exercice 8 : Majorité absolue ?

Soit une liste de n entiers. Proposer et prouver un algorithme linéaire (en $O(n)$) pour tester si il existe une valeur majoritaire (absolue) et connaître cette valeur si elle existe. Le problème est donc :

Entrée : $x_1, \ldots, x_n$

Sortie :

a si $x_i = a$ pour au moins $n/2 + 1$ indices i.

Non, si il n'existe pas de tel a.

Exercice 9 : Nuts and Bolts

Soit un sac contenant n écrous et n boulons. Chaque écrou correspond exactement à un boulon mais vous ne savez pas auquel. On peut comparer un écrou avec un boulon et déterminer si il est trop petit, trop grand ou de la bonne taille. Il n'est pas possible de comparer directement les boulons entre eux et les écrous entre eux. Donner un algorithme efficace pour associer chaque boulon à l'écrou correspondant. Discuter à la fois le pire des cas et la complexité moyenne (vous pouvez vous baser sur l'analyse en moyenne d'un algorithme classique, sans refaire une analyse en moyenne complète).

Exercice 10 : Lancer de balles

On dispose d'un stock de boules en verre. Pour tester leur résistance, on veut déterminer à partir de quel étage d'un immeuble, une boule se casse si on la jette par la fenêtre. L'immeuble a n étages et vous disposez de k boules. Il n'y a qu'une seule opération possible pour tester si la hauteur d'un étage est fatale : jeter une boule par la fenêtre. Si elle ne se casse pas, vous pouvez la réutiliser ensuite, sinon vous ne pouvez plus.

On cherche un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal (on renvoit n + 1 si elle résiste au nième étage).

Q 1. Si on a une seule balle, quel algorithme proposez-vous ? Combien de lancers fera-t-il au moins ? au plus ?

Q 2. On suppose maintenant que $k \geq \log_2(n)$: proposer un algorithme en $O(log_2(n))$ lancers.

Q 3. On suppose que $k < \log_2(n)$; proposer un algorithme en $O(k + \frac{n}{2^{k-1}})$ lancers.

Q 4. On suppose maintenant qu'on ne dispose que de 2 balles.

Q 4.1. Que donne la complexité de l'algorithme précédent dans ce cas ? **Q 4.2.** Proposer un algorithme en $2\sqrt{n}$ lancers. **Q 4.3.** Dans ce dernier cas, proposer aussi un algorithme en $\sqrt{2n}$ lancers.