TD1 : Complexité algorithmique

TAPS : Techniques and Algorithms for Problem Solving

★ 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.

Rappels : 1 μs=10^{-6}s ; 1 ns=10^{-9} s ; $10^{x^y} = 10^{x \times y}$ et $\sqrt{n} = n^{1/2}$.

(a) Nombre d'opérations.

Complexité	n=10	n=100	n=1000
n			
n^2			
n^3			
2^n			
$\sqrt{n}$			
$\sqrt[3]{n}$			
$\log_2(n)$			

(b) Temps nécessaire à 10^9 op/sec.

Complexité	n=10	n=100	n=1000
n			
n^2			
n^3			
2^n			
$\sqrt{n}$			
$\sqrt[3]{n}$			
$\log n$			

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

Complexité	1s	1h	1 an
n			
n^2	$3 \cdot 10^4$		
n^3			
2^n			
$\sqrt{n}$			
$\sqrt[3]{n}$			
$\log(n)$			

Par exemple, pour déterminer la plus grande instance qu'un algorithme de complexité n^2 peut calculer en une seconde, il faut calculer :

$$\begin{aligned} & \max\{n \text{ tel que } n^2 < 10^9\} \\ = & \max\{n \text{ tel que } n < \sqrt{10^9}\} \\ = & \sqrt{10^9} \\ \approx & 3 \cdot 10^4 \end{aligned}$$

★ 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
```

★ 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, et le cas moyen en supposant que les tests ont une probabilité de $\frac{1}{2}$ d'être vrai.

▷ **Question 2:** Donnez une instance de chaque code 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
```

★ 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.

★ 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 faut 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.

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

▷ **Question 3:** Discutez cette complexité par rapport au résultat démontré à l'exercice précédent.