Université Savoie Mont Blanc,
UFR SceM
INFO601_CMI, L3 INFO

November 27, 2025

TD 4, Complexité des algorithmes récursifs

Exercice 1. Méthode générale sur des formules de récurrences

Donnez les complexités asymptotiques induites par les récursivités suivantes:

1. $T(0) = 1, T(n) = 3T(n-1) + 1$

```
De la forme T(n+1)=a T(n) + 1, donc T(n)= Theta( 3^n )
```

2. $T(0) = 1, T(n) = T(n/2) + n/2$

```
Forme diviser pour régner. T(n) = a T( n / b ) + f(n), avec a = 1, b = 2.
On calcule e = log_b a = log_2 1 = 0. On a donc Theta( n^0 ) = constante noeuds.
Clairement f(n)= n/2 = Omega( 1 ) domine n^e et f(n/2) > c f(n) pour 0 < c < 1/2.
On conclut (cas 3 du Théorème) que T(n)=Theta(f(n)) = Theta(n)
```

3. $T(0) = T(1) = 1, T(n) = T(n-1) + 2T(n-2)$

```
De la forme T(n+2) = c2 T(n+1) + c1 T(n) + c0, avec c2=1, c1=2, c0=0.
On calcule le discriminant de x^2 - x - 2.
D = 1 + 8, d = sqrt(9) = 3
r1 = (1 - 3)/2 = -1 et r2 = (1+3)/2 = 2.
On conclut que
T(n) = Theta( r1^n ) + Theta( r2^n ) = Theta( (-1)^n ) + Theta( 2^n )
     = Theta(2^n)
```

4. $T(0) = T(1) = 1, T(n) = 2T(n-2)$

```
De la forme T(n+2) = c2 T(n+1) + c1 T(n) + c0, avec c2=0, c1=2, c0=0.
On calcule le discriminant de x^2 - 2.
D = 8, d = 2 sqrt(2)
r1 = -sqrt(2) et r2 = sqrt(2)
On conclut que
T(n) = Theta( r1^n ) + Theta( r2^n ) = Theta( (-sqrt(2))^n ) + Theta( sqrt(2)^n )
     = Theta(sqrt(2)^n)

Autre façon. on définit T'(n) := T(2n), T'(0) = 1.
Donc T'(n) = 2T(2n-2) = 2T(2(n-1)) = 2T'(n-1).
On a donc T'(n) = Theta(2^n) (suite géométrique)
D'où
T(n) = T'(n/2) = Theta(2^(n/2) ) = Theta(sqrt(2)^n)
```

Exercice 2. Analyse d'un algorithme original de tri

Soit la fonction C suivante:

```
void tri3( int* T, int i, int j )
{
  if ( T[ i ] > T[ j ] ) {
    int tmp = T[ i ]; T[ i ] = T[ j ]; T[ j ] = tmp;
  }
```

```
  if ( i+1 >= j ) return;
  int k = (j-i+1)/3;
  tri3( T, i, j-k );
  tri3( T, i+k, j );
  tri3( T, i, j-k );
}
```

1. Montrer que la fonction `tri3` trie correctement le tableau T entre les indices i et j.

```
Si j-i < 2 cet algo trie correctement.
Cas général: après le 2ème tri, le tiers plus grand est le bon dans l'ordre.
En effet, on prend un élément du tiers + grand. S'il est déjà dedans,
ou si il est au milieu, c'est bon.
Si il est tout à gauche, il sera dans la deuxième moitié après la première passe,
car au moins n/3 des 2n/3 sont plus petits que lui.
Et donc sera trié après.
Du coup le dernier tri trie tous les autres.
```

2. Quel est le temps d'exécution asymptotique de la fonction `tri3` ?

```
T(n) = 3 T(n / (3/2) ) + O(1)
e = log_b a = log(3/2) 3 = ln 3 / ( ln 3 - ln 2 ) ~= 2.709
Or f(n)=O(1), donc dans le cas du théorème.
T(n)=Theta(n^e)=O(n^2.71)
```

3. Cet algorithme est-il meilleur que: le tri bulle, le tri fusion, le quicksort ?

```
Non !
```

4. Pouvez-vous imaginer des algorithmes `tri4`, `tri5` voire `triN` bâtis selon le même principe ? Quelles seraient alors leur complexité algorithmique ? Seraient-ils meilleurs, moins bons ?

```
On divise plus (4, 5, etc) en raccourcissant les sous-calculs.
On tend vers le tri bulle !
```

Exercice 3. (Killer) théorème pour "diviser pour régner"

Quelques questions:

- Peut-on analyser QuickSort ainsi ?

```
Non !
```

- Dans TriFusion, comptez précisément le nombre de fois où une valeur est copiée. Cela vous pourrait-il plus ou moins qu'un QuickSort ? Même question pour le nombre de comparaisons.

```
fusion copies: 2*n*log_2 n
fusion comparaisons: n*log_2 n
quicksort (k prof max): <= n * k
quicksort comparaisons: <= n * k
```

- Utiliser ce théorème pour montrer les bornes asymptotiques dans les cas suivant:
  1. $T_1(n) = 4T_1(n/2) + n$,
  2. $T_2(n) = 4T_2(n/2) + n^2$,
  3. $T_3(n) = 4T_3(n/2) + n^3$.
- Peut-on appliquer le théorème précédent pour une récurrence de la forme $T(n) = 2T(n/2) + n \log n$?

Exercice 4. Analyse d'algorithme: la transformée de Fourier rapide (FFT)

La transformée de Fourier est une transformation mathématique qui décompose une fonction périodique en une somme de fonctions sinusoïdales de fréquences de plus en plus rapides. Cette transformée est utilisée partout notamment pour le traitement des sons (c'est d'ailleurs ce que fait notre oreille lorsqu'elle entend des notes) et des images. Vos images JPEG et vos flux vidéos MPEG sont des formats compressées grâce à une variante de la transformée de Fourier. La théorie nous importe peu ici, ce qui nous intéresse c'est l'algorithme de calcul de la transformée de Fourier discrète. Pourquoi discrète ? C'est juste que la fonction sera décrite comme un tableau T de n valeurs (complexes, mais ça ne change rien) au lieu d'une fonction continue (on ne sait pas représenter les fonctions continues en informatique). A partir de T, l'algorithme, appelé FFT, calcule un autre tableau Y de n valeurs (complexes).

Note (pour votre culture) : par exemple, si on veut éliminer tous les sons aigüs d'un son t avec n valeurs, on calcule $y =$FFT(t), on met à zéro tous les coefficients d'indices $n/2$ à n, et on recalcule $t' =$FFT(y). Le son t' ne contient plus que les graves !

```
// Calcule la transformée de Fourier discrète Y d'un tableau T de n valeurs,
   où n est une puissance de 2.
Action FFT( E n: entier, E T: tab[0..n-1] de complexes, S Y: tab[0..n-1] de complexes );
Var : i : entier ;
Var : A, B, U, V: tab[0..n/2-1] de complexes ;
Var : ω, r: complexe ;
début
    si n = 1 alors Y[0] ← T[0];
    else
        r ← 1; ω ← e^(2πi/n) ;        // Racines de 1, mais vous n'avez pas besoin de le
         savoir.
1       pour i de 0 à n/2-1 faire
            A[i] ← T[2*i];
            B[i] ← T[2*i+1];
        FFT(n/2, A, U);
        FFT(n/2, B, V);
2       pour i de 0 à n/2-1 faire
            Y[i] ← U[i] + r*V[i];
            Y[i+n/2] ← U[i] - r*V[i];
            r ← r*ω;
```

1. Quelle est la complexité de la première boucle (ligne 1) en fonction de n ?

   Theta(n/2)=Theta(n)

2. Quelle est la complexité de la seconde boucle (ligne 2) en fonction de n ?

   Theta(n/2)=Theta(n)

3. On voit facilement que le temps d'exécution de la fonction FFT ne dépend que de la taille n du tableau en entrée. Soit $T(n)$ le temps d'exécution en pire cas de la fonction FFT. Utiliser le théorème général donnant le temps d'exécution des fonctions récursives pour calculer $T(n)$. Sommes-nous dans le cas 1), 2) ou 3) de ce théorème ? Donnez un autre exemple de fonction récursive classique ayant la même complexité.

```
Theta(n) = 2Theta(n/2) + n
b = 2, a = 2, e = log_b a = 1.
On compare f(n)=n avec Theta(n^e)=Theta(n). Cas 2.
On arrive à T(n)=Theta(n log n ).
Ex: tri fusion
```

Exercice 5. Mesures de temps et complexité expérimentale

On effectue les mesures suivantes de temps d'exéution.

n	10000	20000	40000	80000	160000	320000	640000	1280000
temps (s)	0.008	0.014	0.027	0.056	0.119	0.266	0.540	1.189

n	2560000	5120000	10240000	20480000	40960000	81920000	163840000	327680000
temps (s)	2.651	5.895	13.262	30.783	70.050	160.830	375.450	834.820

Estimez la complexité de cet algorithme. Il faudra s'aider d'un outil type gnuplot ou sage pour tracer les courbes.

corrections

corrections