TI205 - Introduction à la complexité

Devoir écrit 2023-2024 (Corrigé et barème de correction)

Exercices (10 minutes : 4 points)

Exercice 1. Ordonner les complexités suivantes en ordre de grandeur du plus petit au plus grand.

a) $T_1(n) = \frac{n(n+1)}{2}$

b) $T_2(n) = n \log n$

c) $T_3(n) = n^3 + n$

d) $T_4(n) = 2^n - 1$

e) $T_5(n) = 10$

f) $T_6(n) = \log n - 7$

Solution: (2 points). L'ordre de grandeurs des complexités est le suivant:

$$T_5(n) < T_6(n) < T_2(n) < T_1(n) < T_3(n) < T_4(n)$$

car en simplifiant les expressions, on obtient:

1) $T_5(n) = 10 = \mathcal{O}(1)$

2) $T_6(n) = \log n - 7 = \mathcal{O}(\log n)$

3) $T_2(n) = n \log n = \mathcal{O}(n \log n)$

4) $T_1(n) = \frac{n(n+1)}{2} = \mathcal{O}(n^2)$

5) $T_3(n) = n^3 + n = \mathcal{O}(n^3)$

6) $T_4(n) = 2^n - 1 = \mathcal{O}(2^n)$

■

Exercice 2. Le nombre d'occurrences d'une valeur x dans un tableau T de taille n est donné par la relation suivante:

$$\textsc{occurrence}(T,n,x) = \begin{cases} 0 & \text{si } n = 0 \\ \textsc{occurrence}(T,n-1,x)+1 & \text{si } T[n-1] = x \\ \textsc{occurrence}(T,n-1,x) & \text{sinon} \end{cases}$$

Proposer un algorithme récursif OCCURRENCE(T,n,x) qui renvoie le nombre d'occurrences de x dans un tableau T de taille n.

Solution: (2 points). Voici deux propositions d'algorithme:

```
OCCURRENCE(T, n, x)
  if n = 0 then:
      return 0
  else:
      if T[n-1] = x then:
          return OCCURRENCE(T, n-1, x) + 1
      else:
          return OCCURRENCE(T, n-1, x)
```

```
OCCURRENCE(T, n, x)
  if n = 0 then:
      return 0
  else if T[n-1] = x then:
      return OCCURRENCE(T, n-1, x) + 1
  else:
      return OCCURRENCE(T, n-1, x)
```

■

Analyse d'algorithmes itératifs (20 minutes : 8 points)

Exercice 3. On considère l'algorithme suivant. Lors de son analyse, on considère que le nombre d'icrémentations de la variable c détermine son temps d'exécution. La valeur de c à l'issue de l'exécution de l'algorithme est donc un indicateur de sa complexité.

```
FONCTION(n)
  c ← 0
  i ← 1
  while i < n do:
      j ← i + 1
      while j ≤ n do:
          c ← c + 1
          j ← j + 1
      i ← i * 2
```

1. Que vaut la variable c à la fin de l'exécution de FONCTION(16)?
2. On suppose que n est une puissance de 2. C'est-à-dire que $n = 2^k$ pour un certain entier $k \geq 0$.
   a) Lister les valeurs successives de i pour chaque itération de la boucle externe. En déduire le nombre d'itérations de la boucle externe en fonction de k.
   b) La boucle interne étant exécutée $n - j + 1$ fois, exprimer le nombre d'itérations de la boucle interne en fonction de k et de i.
   c) En vous aidant des deux résultats précédents, montrer que $c = 2^k(k-1)+1$.
3. Exprimer enfin la valeur de c en fonction de n. En déduire la complexité de cet algorithme en notation $\mathcal{O}$.

Solution: .

1. **(1.5 point)**. En exécutant l'algorithme, on obtient:

Itération sur i	Itération sur j	Valeur de c
$i = 1$	$j = [2, 16]$	$0 + (16 - 2 + 1) = 15$
$i = 2$	$j = [3, 16]$	$15 + (16 - 3 + 1) = 29$
$i = 4$	$j = [5, 16]$	$29 + (16 - 5 + 1) = 41$
$i = 8$	$j = [9, 16]$	$41 + (16 - 9 + 1) = 49$
$i = 16$		

   Donc, $c = 49$.

2. a) **(1.5 points)**. Les valeurs successives de i sont:
   - $i = 1$ pour la première itération.
   - $i = 2$ pour la deuxième itération.
   - $i = 4$ pour la troisième itération.
   - $i = 8$ pour la quatrième itération.

   Puis i vaudra 16 à la sortie de la boucle externe. On a donc 4 itérations de la boucle externe. Si $n = 2^k$, alors $i = 2^0, 2^1, 2^2, \dots, 2^{k-1}$, on a donc k itérations de la boucle externe.

   1 point si les valeurs de i sont correctes. **0.5 point** si le nombre d'itérations de la boucle externe en fonction de k est correct.

b) **(1.5 point)**. La boucle interne est exécutée $n-j+1$ fois. Or, $j=i+1$. Donc, la boucle interne est exécutée $n-i$ fois. Et en fonction de k et de i, elle est exécutée 2^k-i fois.

c) **(2 points)**. D'une part, d'après la question (a), i prends les valeurs $2^0, 2^1, 2^2, \cdots, 2^{k-1}$. D'autre part, d'après la question (b), la boucle interne est exécutée 2^k-i fois. On a donc:

$$\begin{aligned}
i = 2^0 &\Rightarrow 2^k - 2^0 \text{ incrémentations de } c\\
i = 2^1 &\Rightarrow 2^k - 2^1 \text{ incrémentations de } c\\
i = 2^2 &\Rightarrow 2^k - 2^2 \text{ incrémentations de } c\\
&\vdots\\
i = 2^{k-1} &\Rightarrow 2^k - 2^{k-1} \text{ incrémentations de } c
\end{aligned}$$

En sommant ces valeurs, on obtient:

$$c = \sum_{p=0}^{k-1} 2^k - 2^p.$$

On développe cette somme pour obtenir:

$$\begin{aligned}
c &= \sum_{p=0}^{k-1} 2^k - \sum_{p=0}^{k-1} 2^p\\
&= k2^k - \frac{1-2^k}{1-2}\\
&= k2^k - (2^k - 1)\\
&= k2^k - 2^k + 1\\
&= 2^k(k-1) + 1
\end{aligned}$$

3. **(1.5 point)**. En remplaçant k par $\log n$, on obtient $c = n(\log n - 1) + 1 = n\log n - n + 1$. La complexité de cet algorithme est donc $\mathcal{O}(n\log n)$. ($\mathcal{O}(n\log_2 n)$ est aussi accepté).

■

Analyse d'algorithmes récursifs (20 minutes : 8 points)

Exercice 4. On souhaite calculer le maximum d'un tableau T d'entiers de taille $n \geq 1$, **non trié**, en utilisant le même principe que la recherche dichotomique. On propose l'algorithme suivant:

```
BINARYMAX(T, l, r)
  if l + 1 < r then:
      mid ← ⌊(l+r)/2⌋
      maxl ← BINARYMAX(T, l, mid)
      maxr ← BINARYMAX(T, mid + 1, r)
      if maxl > maxr then:
          return maxl
      else:
          return maxr
  else:
      return T[l]
```

On remarque que $r - l + 1 = n$. Pour simplifier l'analyse, on suppose que $n = 2^k$ pour un certain entier $k \geq 0$.

1. Pour $T = [42, 12, 38, 9, 0, 7, 29, 21]$, vérifier que BINARYMAX$(T, 0, 7)$ renvoie bien 42, en complétant le tableau suivant.

   Indication: Reporter à chaque appel de BINARYMAX les valeurs de l, r, mid, maxl, maxr et la valeur de retour de la fonction.(Les cases grisées indiquent que pour l'appel correspondant, les variables ne sont pas calculées).

l	r	mid	maxl	maxr	Valeur de retour de BINARYMAX(T, l, r)
0	7				
0	3				
0	1				
0	0				
1	1				
2	3				
2	2				
3	3				
4	7				
4	5				
4	4				
5	5				
6	7				
6	6				
7	7				

2. On suppose que l'opération qui contribue le plus à la complexité de l'algorithme est la comparaison **maxl > maxr**. On dénote par $T(n)$ le nombre de comparaisons de ce type effectué par l'algorithme pour un tableau de taille n.
   a) Que vaut $T(1)$?
   b) Exprimer $T(n)$ en fonction de $T(n/2)$ pour tout $n > 1$.
   c) En déduire la complexité de cet algorithme en notation $\mathcal{O}$. Cet algorithme est-il plus efficace que la recherche linéaire?

Solution: 1. **(4 points)**. Il n'est pas nécessaire d'avoir reporté les appels récursifs dans le tableau. Les valeurs de **maxl** et **maxr** sont suffisantes pour déterminer la valeur de retour de la fonction. En complétant le tableau, on obtient:

l	r	mid	maxl	maxr	Valeur de retour de BINARYMAX(T,l,r)
0	7	3	BINARYMAX$(T,0,3) = 42$	BINARYMAX$(T,4,7) = 29$	42
0	3	1	BINARYMAX$(T,0,1) = 42$	BINARYMAX$(T,2,3) = 38$	42
0	1	0	BINARYMAX$(T,0,0) = 42$	BINARYMAX$(T,1,1) = 12$	42
0	0				42
1	1				12
2	3	2	BINARYMAX$(T,2,2) = 38$	BINARYMAX$(T,3,3) = 9$	38
2	2				38
3	3				9
4	7	5	BINARYMAX$(T,4,5) = 7$	BINARYMAX$(T,6,7) = 29$	29
4	5	4	BINARYMAX$(T,4,4) = 0$	BINARYMAX$(T,5,5) = 7$	7
4	4				0
5	5				7
6	7	6	BINARYMAX$(T,6,6) = 29$	BINARYMAX$(T,7,7) = 21$	29
6	6				29
7	7				21

2. a) **(1 point)**. Pour $n = 1$, on a $T(1) = 0$.

   b) **(1 point)**. Pour $n > 1$, on a:

$$T(n) = 2T(n/2) + 1$$

   c) **(2 points)**. En effectuant la subsitution, on a:

$$\begin{aligned}
T(n) &= 2(2T(\frac{n}{4}) + 1) + 1 \\
&= 4T(\frac{n}{4}) + 2 + 1 \\
&= 4(2T(\frac{n}{8}) + 1) + 2 + 1 \\
&= 8T(\frac{n}{8}) + 4 + 2 + 1 \\
&\vdots \\
&= 2^k T(\frac{n}{2^k}) + 2^{k-1} + 2^{k-2} + \cdots + 2 + 1 \\
&= nT(1) + \sum_{i=0}^{k-1} 2^i \\
&= 0 + \frac{1-2^k}{1-2} \\
&= 2^k - 1 \\
T(n) &= n - 1
\end{aligned}$$

Donc, $T(n) = n - 1$. La complexité de cet algorithme est donc $\mathcal{O}(n)$. Elle a la même complexité que la recherche linéaire.

■