Examen Complexité Session Principale 30 Mai 2023

Correction + Barème

Exercice 1 (4 points)

Pour chaque séquence d'algorithme proposée ci-dessous, calculer l'ordre de complexité **en justifiant votre réponse.** Op est une opération de coût O(n²)

Séquence 1	**Séquence 2**	**Séquence 3**	**Séquence 4**

Pour i =1 à n Faire
  Pour j =i à n Faire
    y(i)=y(i)+A(i,j)*x(j)
  Fin Pour
Fin Pour
	Pour i =1 à n Faire
	  Si (i≠n) alors
	    Pour j=i à i+1 Faire
	      y(i)=y(i)+A(i,j)*x(j)
	    Fin Pour
	  Fin Si
	  y(i) = A(i,j)*x(j)
	Fin Pour
		Pour i =1 à n Faire
		  Pour j=i à n Faire
		    Pour k = i à j Faire
		      Op /coût O(n²)/
		    Fin Pour
		  Fin Pour
		Fin Pour
			Pour i = 2 à N Faire
			  X = A(i)
			  j = i
			  Tant que (X<A(j-1)&j>1)
			    A(j)= A(j-1)
			    j = j-1
			  Fin Tant que
			  A(j) = X
			Fin Pour

O(n²) Produit matrice triangulaire supérieure et vecteur 2 boucles Pour imbriquées	**O(n)** Produit matrice diagonale et vecteur 2 boucles Pour imbriquées avec la 2ème de cous O(1)	**O(n⁵)** **=O(n³)*O(n²)** 2 boucles Pour imbriquées	**O(n²)** Tri par insertion linéaire
1pt	**1pt**	**1pt**	**1pt**

Exercice 2 (7 points)

Les six différentes versions d'algorithmes suivantes sont des solutions itératives pour résoudre le même problème. Avec **mod** retourne le reste de la division euclidienne d'un entier, **premier** une variable initialisée à **true** et **f** une fonction qui retourne la partie entière d'un nombre réel.

Version 1
For i=2 To n-1 Do
  if (n mod i == 0) then
    premier = false
  EndIf
EndFor
	Version 2
	For i=2 To n/2 Do
	  if (n mod i == 0) then
	    premier = false
	  EndIf
	EndFor
		Version 3
		For i=2 To f($\sqrt{n}$) Do //racine carré de n
		  if (n mod i == 0) then
		    premier = false
		  EndIf
		EndFor

Version 4
If (n≠2) and (n mod 2 == 0) then
  premier = false
Else
  For i=3 To n/2 Do
    If (n mod i == 0) then
      premier = false
    EndIf
    i=i+2
  EndFor
EndIf
	Version 5
	If (n≠2) and (n mod 2 == 0) then
	  premier = false
	Else
	  For i=3 To n-2 Do
	    if (n mod i == 0) then
	      premier = false
	    EndIf
	    i=i+2
	  EndFor
	EndIf
		Version 6
		If (n≠2) and (n mod 2 == 0) then
		  premier = false
		Else
		  For i=3 To f(√n) Do / racine carré de n
		    if (n mod i == 0) then
		      premier = false
		    EndIf
		    i=i+2
		  EndFor
		EndIf

1. Donner une trace d'exécution pour la **version 1** et la **version 6** d'algorithme avec n=17 (√17=4,12 et f(√17)=4). Déduire ce que font ces algorithmes.

Trace d'exécution de la version 1	Trace d'exécution de la version 6
Initialisation : premier = True iteration 1: i=2 on a (17 mod 2 == 0) = false iteration 2: i=3 et (17 mod 3 == 0)= false iteration 3: i=4 on a (17 mod 4 == 0) =false iteration 4: i=5 et (17 mod 5 == 0) =false iteration 5: i=6 on a (17 mod 6 == 0) =false iteration 6: i=7 et (17 mod 7 == 0) =false iteration 7: i=8 on a (17 mod 8 == 0)= false iteration 8: i=9 et (17 mod 9 == 0) =false iteration 9: i=10 on a (17 mod 10 == 0) =false iteration 10: i=11 et (17 mod 11 == 0) =false iteration 11: i=12 on a (17 mod 12 == 0)= false iteration 12: i=13 et (17 mod 3 == 13) =false iteration 13: i=14 on a (17 mod 14 == 0)= false iteration 14: i=15 et (17 mod 15 == 0) =false iteration 15: i=16 on a (17 mod 16 == 0)= false EndFor premier = **True** **(1.5 pt)**	Initialisation : premier = True; n=17 (17≠2 and (17 mod 2 ==0))=false iteration 1: i=3 : (17mod3==0)=false i=i+2=5 iteration 2: i=5> f(√17)=4 EndFor premier = True **(1pt)**

2. La **version 7** utilise une fonction récursive **Diviseur**. Donner le type de récursivité de la fonction **Diviseur** puis la dérécursiver.

Récursivité terminale(0.5pt)

```
Diviseur (a,b)
If (a  <=0) then Erreur
While (a<b) do
        b ← b-a
End_while
      return (a==b)
End_Algorithme
```
(2 pt)

4. Calculer la complexité de la **version 7** de l'algorithme. **(2 pt)**

Version 7
For i=2 To n-1 Do
If (Diviseur(i,n)) then
Premier = false
EndIf
EndFor
	Diviseur (a,b) : Boolean
	If (a<=0) then Affiche("Erreur")
	Else
	If(a>=b) then return (a==b)
	Else
	Return (Diviseur(a,b-a))
	EndIf
	EndIf
	EndDiviseur

$O(n^2)$=O(n)*O(n)	Complexité de la version dérécursivé de Diviseur est O(n) lorsque b=n

Exercice 3 (9 points)

On considère A un tableau d'entiers de taille n représentant les valeurs d'une fonction multimodale. On appelle un pic dans le tableau A, une valeur plus grande que les valeurs qui la voisinent (valeur précédente et valeur suivante). Il faut noter que le nombre de voisins d'une valeur peut être égal à 1 si la valeur se trouve au début ou à la fin du tableau. Notre problème consiste à trouver un pic parmi tous les pics existants.

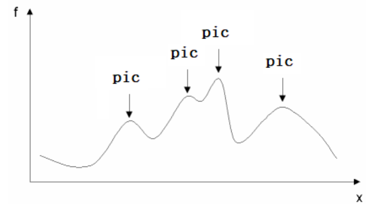

Figure 1 : Exemple de fonction multimodale

Pour résoudre ce problème, on considère l'algorithme récursif «**Rechercher_pic**» qui suit le principe diviser pour régner, son code est le suivant :

```
int Rechercher_pic(int A[], int left, int right, int n)
{
  int mid = left + (right - left)/2;
  if ((mid == 0 && A[mid + 1] <= A[mid]) || (mid == n - 1 && A[mid - 1] <=
        A[mid]) || (A[mid - 1] <= A[mid] && A[mid + 1] <= A[mid]))
         return mid;
  else if (mid > 0 && A[mid - 1] > A[mid])
         return rechercher_pic(A, left, (mid - 1), n);
  else
         return rechercher_pic(A, (mid + 1), right, n);
}
```

On se propose de calculer la complexité de cet algorithme en nombre de comparaisons.

1. Déterminer l'équation récurrente de complexité de cet algorithme(2 point)

   $$T(n)=\begin{cases} O(1) & si \quad n \le c \\ aT(n/b)+D(n)+C(n) & \text{sinon} \end{cases}$$

   T(n)=O(1) si n≤1
   T(n)=T(n/2)+O(1) avec a=1, b=2, D(n)=O(1) et C(n)=O(1)

2. Déduire la complexité de cet algorithme (1.5 point)

	Si la forme de f(n) est :	*Alors le coût est :*
1	$f(n)=O(n^{(\log_b a)-\varepsilon})$ *pour un* $\varepsilon > 0$	$T(n)=\Theta(n^{\log_b a})$
2	$f(n)=\Theta(n^{\log_b a})$	$T(n)=\Theta(n^{\log_b a}\log n)$
3	$f(n)=\Omega(n^{(\log_b a)+\varepsilon})$ *pour un* $\varepsilon > 0$, *et* $af(n/b) \le cf(n)$ *pour un* $c < 1$ *et n suffisamment grand*	$T(n)=\Theta(f(n))$

   $f(n)=O(n^{\log_2 1})=O(n^0)=1$ → $2^{ème}$ cas alors $T(n)=O(n^{\log_2 1} \log n)=\log n$

3. Cet algorithme est-t-il considéré comme rapide ?
   Oui, Algorithme rapide car il a une complexité logarithmique (1 point)

4. Écrire un algorithme itératif «**Rechercher_pic_iter**» qui permet de retourner un pic du tableau A. (2 point)

```
Rechercher_pic_iter(int A[ ], int n) : int
 Pour i=1 à n Faire
   Si A[i]>A[i+1] alors Retourner i
   Sinon
      Si A[i+1]>A[i+2] alors Retourner i+1
      Fin Si
  Fin Si
Fin Pour
Retourner n
Fin Rechercher_pic_iter
```

5. Calculer la complexité de l'algorithme «**Rechercher_pic_iter**».

   O(n) (1 point)

6. Quelle est la solution la plus performante ? Justifier votre réponse. (1.5 point)

   Rechercher_pic est plus rapide que **Rechercher_pic_iter** **O(logn)<O(n),** d'où **Rechercher_pic** est plus performant