EXAMEN

Semestre : 1 ☐ 2 ■

Session : Principale ■ Rattrapage ☐

Unité d'enseignement : Equipe Complexité
Module (s) : Complexité appliquée à la RO
Classes : 4SE1..5, 4Arctic1..7, SLEAM1..3, GAMIX1..2, SIM1..5

Nombre d'exercices : 3 Nombre de pages : 2

Date : 30/052023 Heure 11h Durée :1h30

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

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	Version 2	Version 3

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

Version 4	**Version 5**	**Version 6**

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
	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
		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.
2. La **version 7** utilise une fonction récursive **Diviseur**. Donner le type de récursivité de la fonction **Diviseur** puis la dérécursiver.
3. Calculer la complexité de la **version 7** de l'algorithme.

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

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.

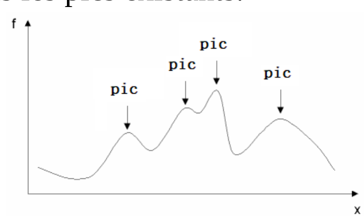

Figure 1 : Exemple de fonction multimodale

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. Déduire la complexité de cet algorithme
3. Cet algorithme est-t-il considéré comme rapide ?
4. Écrire un algorithme itératif «**Rechercher_pic_iter**» qui permet de retourner un pic du tableau A.
5. Calculer la complexité de l'algorithme «**Rechercher_pic_iter**».
6. Quelle est la solution la plus performante? Justifier votre réponse.