ALGORITHMES

ITÉRATIFS

Master1 : MRID
ISSAT Gafsa

EXERCICE 1 : Complexité de séquences itératives

Déterminer (en fonction de n à $\mathcal{O}$ ( ) près) la complexité en nombre « d'opérations » de chaque séquence :

Séq-1 :
```
FOR i ← 1 TO N DO
    Opération ;
END-FOR
```

Séq-2 :
```
FOR i ← 1 TO N DO

    FOR j ← 1 TO i DO
        Opération ;
    END-FOR

END-FOR
```

Séq-3 :
```
i ← 1
WHILE ( i < N) DO

    i ← 2*i
    Opération ;

END-WHILE
```

Séq-4 :
```
FOR i ← 1 TO N DO

    J ← 1
    WHILE (J < N ) DO
        J ← 2 * J
        Opération;
    END-WHILE

END-FOR
```

Séq-5 :
```
i ← 1

WHILE ( i < N ) DO

    i ← 2 * i

    FOR j ← 1 TO i DO
        Opération;
    END-FOR

END-WHILE
```

Séq-6 :
```
i ← 1
FOR j ← 1 TO n DO
    i ← 2*i
END-FOR

FOR j ← 1 TO i DO
    Opération;
END-FOR
```

Séq-7 :
```
i ← 1
FOR k ← 1 TO n DO
    i ← 2*i

    FOR L ← 1 TO i DO
        FOR m ← 1 TO k DO
            Opération ;
        END-FOR

    END-FOR

END-FOR
```

Séq-8 :
```
FOR k ← 1 TO n DO
    i ← 1

    FOR j ← 1 TO k DO
        i ← 2*i
    END-FOR

    FOR j ← 1 TO i DO
        Opération
    END-FOR

END-FOR
```

EXERCICE 2 : Recherche du maximum

Calculer la complexité des fragments de code suivants :

SEQ-1 :

```
i ← n
s ← 0

WHILE (i > 0) DO

      j ← 2 * i

      WHILE (j > 1) DO
            s ← s + (j - i) * (s + 1)
            j ← j - 1
      END-WHILE

      i ← i div 2

END-WHILE
```

SEQ-2 :

```
P ← 1

FOR I ← 1 TO n DO
    J ← 1
    K ← 1

    WHILE (K <= n) DO
          P ← P * (K + J)
          K ← K + 1

          IF (K > n) THEN
              J ← J + 1

             IF (J <= n) THEN
                K ← 1
             END-IF

          END-IF

    END-WHILE

END-FOR
```


EXERCICE 3 : Recherche du maximum

1. Concevoir un algorithme de recherche du maximum dans un ensemble à n éléments
2. Quelle est la complexité de votre algorithme en nombre de comparaisons ?
3. Montrer qu'il est optimal.

EXERCICE 4 : Recherche du maximum et du minimum

Nous supposons ici que l'ensemble considéré ne contient pas deux fois la même valeur.

1. Proposer un algorithme naïf de recherche du maximum et du minimum d'un ensemble de *n* éléments.
2. Quelle est sa complexité en nombre de comparaisons ?
3. Proposer un algorithme plus efficace.

   Indication : dans une première phase les éléments sont comparés par paire.
4. Quelle est sa complexité en nombre de comparaisons ?

EXERCICE 5 : Recherche du deuxième plus grand élément

Nous supposons ici que l'ensemble considéré ne contient pas deux fois la même valeur.

1. Proposer un algorithme simple de recherche du deuxième plus grand élément.
2. Quelle est sa complexité en nombre de comparaisons ?
3. Réécrire l'algorithme de recherche du maximum sous la forme d'un tournoi de tennis. Il n'est pas nécessaire de formaliser l'algorithme ici, une figure explicative suffit.
4. Dans combien de comparaisons, le deuxième plus grand élément de l'ensemble s'est rendu compte qu'il est le plus petit des deux éléments comparés ?
5. Proposer un nouvel algorithme de recherche du deuxième plus grand élément.
6. Quelle est sa complexité en nombre de comparaisons ?

EXERCICE 6 : Motif 1D

Etant donnée un tableau entier A de taille N et un tableau entier F de taille U. On suppose que U est inférieur à N.

1. Ecrire un algorithme naïf pour chercher les sous-tableaux F de A égaux au tableau F. L'algorithme doit afficher la position début à laquelle le tableau F est trouvé.
2. Estimer sa complexité en fonction de N et U