COMPLEXITE

PREMIERE PARTIE

EXERCICE 1

Soit P(n) un polynôme en n. Pour quelles valeurs de p a-t-on P(n)=O(n^p)?

P(n)= 0(n^p) pour tous les p supérieurs ou égaux au degré de P.

EXERCICE 2

Pour quelles valeurs de p a-t-on P(n)= Θ(n^p)?

Uniquement pour p égal au degré du polynôme

EXERCICE 3

Montrer que pour tout entier k, la somme des n premiers entiers à la puissance k est en $\Theta(n^{k+1})$

Première démonstration.

$\sum_{i=1}^{n} i^k \le \sum_{i=1}^{n} n^k = n^{k+1}$

D'autre part $\left(\frac{n}{2}\right)^{k+1} \le \sum_{i=n/2}^{n} \left(\frac{n}{2}\right)^k \le \sum_{i=n/2}^{n} i^k \le \sum_{i=1}^{n} i^k$

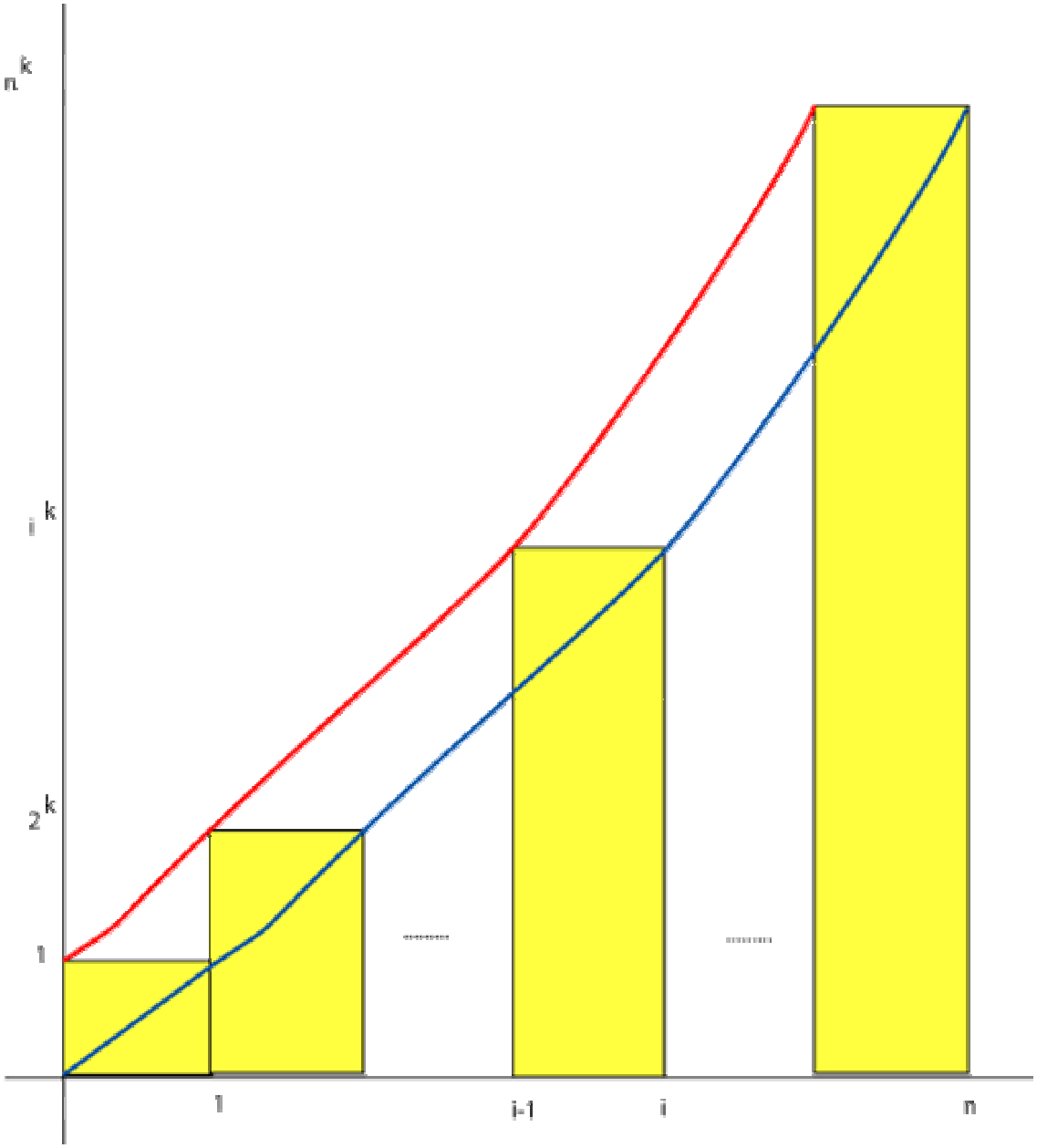
On a donc bien $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$

Deuxième démonstration :

La somme $\sum_{i=1}^{n} i^k$ est égale à la surface des rectangles en jaune sur la figure ci dessous, cette surface est comprise entre l'aire sous la courbe bleue et l'aire sous la courbe rouge, on a donc

$\int_0^n x^k dx \le \sum_{i=1}^{n} i^k \le \int_0^n (x+1)^k dx = \int_1^{n+1} x^k dx$

On en déduit donc que $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$.



EXERCICE 4

Soient les fonctions

$f_1(n)=n$, $f_2(n)=2^n$,
$f_3(n)=n^2$, $f_4(n)=2n$,
$f_5(n)=n^n$, $f_6(n)=\log n$, $f_7(n)=n!$, $f_8(n)= n\log n$

Pour chaque couple (i, j) dire si on a $f_i=o(f_j)$,
$f_i=O(f_j)$
$f_i=\Theta(f_j)$.

ficolonne fj ligne	n	2^n	n^2	2n	n^n	log n	n !	nlogn
n	O, Θ			O , Θ		o,O		
2^n	o,O	O, Θ	o,O	o,O		o,O		o,O
n^2	o,O		O, , Θ	o,O		o,O		o,O
2n	O , Θ			O, Θ		o,O		
n^n	o,O	o,O	o,O	o,O	O, Θ	o,O	o,O	o,O
log n						O, Θ		
n !	o,O	o,O	o,O	o,O		o,O	O, Θ	o,O
nlogn	o,O			o,O		o,O		O, Θ

On peut classer ces fonctions de la manière suivante : $\log n \prec\prec n \prec\prec n \log n \prec\prec n^2 \prec\prec 2^n \prec\prec n! \prec\prec n^n$

EXERCICE 5

Si une instruction I se trouve au cœur de k boucles for imbriquées, chacune d'elle de la forme
for (int i_m=0 ; i_m < n ; i_m++)
où 0 < m < (k+1)

Combien de fois l'instruction I est elle exécutée ?

n^k fois.

EXERCICE 6

Si une instruction I se trouve au cœur de k boucles for imbriquées, chacune d'elle de la forme
for (int i_m=0 ; i_m < i_{m-1} ; i_m++)
où 0 < m < (k+1) , avec i_0=n
Combien de fois l'instruction I est elle exécutée ?

Le nombre exact est $\sum_{i_1=0}^{n}\sum_{i_2=0}^{i_1-1}....\sum_{i_k=0}^{i_{k-1}-1}1.$

On montre par récurrence sur k que cette somme est en $\Theta(n^{k+1})$
Le résultat est vrai, si k=1.
Supposons le résultat vrai pour un entier k-1 donné.
Montrons que le résultat est vrai pour k.

On a $\sum_{i_2=0}^{i_1-1}....\sum_{i_k=0}^{i_{k-1}-1}1=\Theta(i_1^{\,k})$, par hypothèse de récurrence.

Comme de plus, $\sum_{i_1=0}^{n} i_1^k=\Theta(n^{k+1})$, on peut en déduire que l'instruction I est exécutée $\Theta(n^{k+1})$ fois.

EXERCICE 7

Estimer la complexité du morceau de code suivant, sachant que l'instruction I1 est en Θ(1) et I1 ne modifie pas les entiers i, j, k et n

```
for (int i=0 ; i < n ; i++) {
        for (int j=i ; j < n ; j++) {
                for (int k=0 ; k < j ; k++) {
                        I1
        }
        }
}
```

I1 est exécuté un nombre de fois égal à

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} j = \sum_{i=0}^{n-1}\left(\sum_{j=1}^{n-1} j - \sum_{j=1}^{i-1} j\right) = \sum_{i=0}^{n-1}\left(\frac{n(n-1)}{2} - \frac{j(j-1)}{2}\right)$$

$$= \frac{n^2(n-1)}{2} - \frac{1}{2}\sum_{i=0}^{n-1}(j^2) + \frac{1}{2}\sum_{i=0}^{n-1}(j) = \frac{n^2(n-1)}{2} - \frac{n(n-1)(2n-1)}{12} + \frac{n(n-1)}{4} = \Theta(n^3)$$

EXERCICE 8

Estimer la complexité du morceau de code suivant, sachant que les instructions I1, I2, I3, (et l'évaluation de la condition) sont en Θ(1) et ne modifient pas les entiers i, j, k et n

```
int i=1 ;
int j=1 ;
while (i <n) {
        i++ ;
        I1 ;
        while (( j < n) && Condition) {
                j++ ;
                I2
        }
        I3 ;
}
```

Ce code a une complexité linéaire. En effet clairement I1 et I3 sont exécutés n fois. Il en est aussi de même de I2 et de l'évaluation de la condition, car la variable j est incrémentée de un chaque fois que I2 est exécutée et n'est jamais modifiée ailleurs.

EXERCICE 9

On connaît l'écriture d'un nombre en base b, et l'on veut convertir ce nombre en base usuelle (base dix).

1. On utilise la méthode "convertionDirecte". Quelle en est la complexité?

```
public int convertionDirecte(int[] a, int b) {
        int résultat =a[0] ;
        int auxiliaire ;
        for (int rang =1 ; rang < a.length ; rang++) {
                if (a[rang] != 0) {
```

```
                auxiliaire = a[rang] ;
                for (int indice =0 ;
            indice <rang ; indice ++) {
                        auxiliaire = auxiliaire *b ;
                 }
                résultat = résultat + auxiliaire;
            }
        return résultat ;
    }
```

2. On utilise la méthode "convertionDirecteV2" dans laquelle on mémorise dans une variable monôme b^{rang}. Ecrire cette méthode "convertionDirecteV2" et en donner la complexité ?

3. Prouvez que la méthode suivante dite de Horner, effectue bien le même travail. Quelle en est la complexité ?

```
public int horner(int[] a, int b) {
        int n = a.length ;
        int résultat =a[n-1] ;
        for (int rang = n-2 ; rang >= 0 ; rang--){
            résultat = b* résultat +a[rang] ;
    }
    return résultat ;
}
```

1.La méthode convertionDirecte à une complexité quadratique en fonction de la longueur de la table a (c'est dire proportionnelle au carré de cette longueur)

2.
```
public int convertionDirecteModifiee (int[] a, int b) {
        int résultat =a[0] ;
        int auxiliaire ;
        int brang=b ;
        for (int rang =1 ; rang < a.length ; rang++) {
            if (a[rang] != 0) {
                auxiliaire = a[rang] *brang ;
                brang=brang*b ;
                résultat = résultat + auxiliaire;
            }
        return résultat ;
    }
```

Cette nouvelle version a une complexité linéaire.

3. La méthode de Horner a aussi une complexité linéaire. Pour prouver que cette méthode est correcte, on utilise un invariant :

```
public int horner(int[] a, int b) {
        int n = a.length ;
        int résultat =a[n-1] ;
        for (int rang = n-2 ; rang >= 0 ; rang--){
            résultat = b* résultat +a[rang] ;
    /* resultat= \sum_{i=rang}^{n-1} a[i] b^{i-rang} */
    }
    return résultat ;
```

}

EXERCICE 10

On désire élever un entier à la puissance n,

1 Quelle est la complexité de la méthode suivante ?

```
public int puissance(int n, int a) {
        int résultat = a ;
        for(int i =1 ; i <n ;i++){
                résultat=résultat*a ;
        }
        return résultat ;
}
```

2.Montrez que le code suivant est correct. Quelle en est la complexité ?

```
public int puissance(int n, int a) {
        int aux = n ;
        int puissanceDea=a ;
        int résultat=1 ;
        while ( aux  != 0 ) {
                if (aux mod 2 == 1) {
                résultat =résultat * puissanceDea ;
                }
                aux=aux/2 ;
                puissanceDea = puissanceDea *puissanceDea ;
        }
        return résultat ;
}
```

1. La première méthode nécessite un nombre linéaire (par rapport à la puissance) de multiplications.
2. La seconde méthode nécessite un nombre de multiplications proportionnel au logarithme de la puissance. Pour montrer que le code est juste, on introduit l'invariant suivant

```
public int puissance(int n, int a) {
        int aux = n ;
        int puissanceDea=a ;
        int résultat=1 ;
        while ( aux  != 0 ) {
                if (aux mod 2 == 1) {
                résultat =résultat * puissanceDea ;
                }
                aux=aux/2 ;
                puissanceDea = puissanceDea *puissanceDea ;
/* aux^puissanceDea * resultat =a^n */
        }
        return résultat ;
}
```