Examen de complexité IMDD330

# Gaëtan Corin

# Consignes

Le devoir est à rendre au format Word exclusivement. Le travail est individuel. La présentation et le respect du formalisme ainsi que le raisonnement seront déterminant dans la notation. Le document Word sera à déposer avant le vendredi 28 novembre 20h dans la boîte de livrable prévue à cet effet.

Pour l’ensemble des fonctions suivantes déterminer leur complexité dans le pire, le meilleur et le cas moyen après les avoir identifiés. **En bonus**, démontrer à quelle classe de complexité appartient chacun de ces algorithmes à l’aide de la méthode vue en cours.

# Exercice 1

1. fonction f1(entier x, entier n) :entier{
2. entier i=0
3. entier retour=1
4. tantque (i<=n) {
5. retour=retour\*x
6. i++
7. }
8. retourne(retour)
9. }

## Description de l’algorithme

L’algorithme retourne un entier ayant la valeur de x exposant de la valeur entre 0 et n.

## Caractérisation des cas

Il n’y a pas d’appel conditionnel dans cette algorithmie, uniquement une itération.  
Ainsi, tous les cas sont des cas moyens.

## Mesure de la complexité de f1()

### Cas général

Comme nous l’avons vu dans la caractérisation des cas, tous les cas sont des cas moyens.  
x n’étant pas itéré sur cette algorithme, il ne crée pas d’influence sur le nombre d’ordres machines.  
Ainsi, ne prendra que n en paramètre.

Nous savons aussi que i = 0.

Ainsi, suivant les valeurs données pour x et pour n, le nombre d’ordres machines sera toujours de

## Détermination de la classe de complexité de f1()

### Appartenance de à la classe

### Appartenance de à la classe

### Appartenance de à la classe

# Exercice 2

1. fonction f2(entier[n] t1, entier[n] t2) :entier[]{
2. entier[] retour
3. entier k=0
4. pour i allant de 0 à t1.taille()-1 {
5. pour j allant de 0 à t2.taille()-1 {
6. si(t1[i]==t2[j]){
7. retour[k]=t1[i]
8. k++
9. }
10. }
11. }
12. retourne(retour)
13. }

## Description de l’algorithme

L’algorithme crée dans la variable « retour » une liste de valeurs représentant toutes les valeurs identiques contenues dans le tableau t1 ainsi que dans le tableau t2.

## Caractérisation des cas

* Le pire des cas est lorsque toutes les valeurs du tableau t1 sont contenu dans le tableau t2
* Le meilleur des cas est lorsque toutes les valeurs du tableau t1 ne sont pas contenu dans le tableau t2
* Le cas moyen est lorsque la moitié des valeurs du tableau t1 est contenu dans le tableau t2

## Mesure de la complexité de f2()

### Cas général

L’algorithme possède un appel conditionnel qui vérifie une égalité de deux valeurs données en paramètre.

L’appel conditionel sera donc une nouvelle complexité appelée

1. fonction f2(entier[n] t1, entier[n] t2) :entier[]{
2. entier[] retour
3. entier k=0
4. pour i allant de 0 à t1.taille()-1 {
5. pour j allant de 0 à t2.taille()-1 {
6. g2(t1[i],t2[j]);
7. }
8. }
9. retourne(retour)
10. }
11. fonction g2(t1[i],t2[j]){
12. si(t1[i]==t2[j]){
13. retour[k]=t1[i]
14. k++
15. }

La mesure de la complexité de est la somme de la complexité de deux affectations(2), ainsi que un appel itératif contenant un autre appel itératif contenant la complexité de la fonction .

On obtient donc pour la mesure de complexité la valeur suivante :

Il nous reste à déterminer la complexité de la fonction g2().

Or la fonction g2() contient un appel conditionnel. Elle va donc avoir une complexité différente en fonction de son pire ou meilleur cas.

#### Mesure de la complexité de g2() dans le meilleur cas

D'après la formule Appel conditionnel, on peut écrire que la complexité de g2() est :

#### Mesure de la complexité de g2() dans le pire cas

D'après la formule Appel conditionnel, on peut écrire que la complexité de g3() est :

#### Mesure de la complexité g2() dans le cas moyen

On étend la formule Appel conditionnel, en déterminant la valeur du cas moyen comme la moyenne des pire et meilleur cas :

### Mesure de la complexité de f2() dans le pire des cas ()

et ont tous les deux la même taille donné par le paramètre d’entrer n. Ils peuvent donc être tous les deux remplacé par la valeur n.

On en déduit que dans le pire des cas est :

### Mesure de la complexité de f2() dans le meilleur cas ()

et ont tous les deux la même taille donné par le paramètre d’entrer n. Ils peuvent donc être tous les deux remplacé par la valeur n.

On en déduit que dans le meilleur des cas est :

### Mesure de la complexité de f2() dans le cas moyen ()

et ont tous les deux la même taille donné par le paramètre d’entrer n. Ils peuvent donc être tous les deux remplacé par la valeur n.

On en déduit que dans le cas moyen est :

## Détermination de la classe de complexité de f2()

### Appartenance de à la classe

### Appartenance de à la classe

### Appartenance de à la classe

# Exercice 3(%=modulo) itératif séquentiel conditionnel

Si b ===== ½ alors il accepte

1. fonction f3(entier n) :entier[]{
2. entier[] retour
3. retour[0]=2
4. entier k=1
5. pour i allant de 3 à n {
6. bool b=Vrai
7. j=2
8. tantque (j<i) et b {
9. si(i%j==0){
10. b=Faux
11. }
12. j++
13. }
14. si b{
15. retour[k]=i
16. k++
17. }
18. }
19. retourne(retour)
20. }

## Description de l’algorithme

L’algorithme retourne une liste de tous les nombres premiers comprenant le chiffre 2 ainsi que tous ceux entre entre 3 et n

## Caractérisation des cas

* Le pire des cas est lorsqu’il y a le plus possible de nombre premier entre 3 et n
* Le meilleur des cas est lorsqu’il y a le moins possible de nombre premier entre 3 et n
* Le cas moyen est lorsqu’il y a un nombre aléatoire de nombre premier entre 3 et n

## Mesure de la complexité de f3()

### Cas général

L’algorithme possède deux appels conditionnels.

Le premier appel conditionnel (ligne10) se déclenche si la division de i par j donne comme résultat un nombre entier.

Le premier appel conditionnel sera donc une nouvelle complexité appelée

Le second appel conditionnel (ligne15) se déclenche si b vaut « vrai ».

Le second appel conditionnel sera donc une nouvelle complexité appelée

1. fonction f3(entier n) :entier[]{
2. entier[] retour
3. retour[0]=2
4. entier k=1
5. pour i allant de 3 à n {
6. bool b=Vrai
7. j=2
8. tantque (j<i) et b {
10. j++
11. }
13. }
14. retourne(retour)
15. }
17. Fonction
18. si(i%j==0){
19. b=Faux
20. }
21. }
23. Fonction
24. si b{
25. retour[k]=i
26. k++
27. }
28. }

La mesure de la complexité de est la somme de la complexité de trois affectations(3), un retourne, ainsi que un appel itératif de 3 à n contenant deux affectations(2), la complexité de , et une boucle conditionnel contenant la complexité de ainsi qu’une incrémentation,

On obtient donc pour la mesure de complexité la valeur suivante :

### Mesure de la complexité de f3() dans le pire des cas ()

### Mesure de la complexité de f3() dans le meilleur cas ()

### Mesure de la complexité de f3() dans le cas moyen ()

## Détermination de la classe de complexité de f2()

### Appartenance de à la classe

### Appartenance de à la classe

### Appartenance de à la classe