8INF259 - STRUCTURES DE DONNEES

BALJ17058609 – Baldo Jérôme

TRAVAIL#3

PROJET#1 EFFICACITE

1. Tableau des temps d'exécution selon la fonction et n

Fonctions	n = 10	n = 100	n = 1000	n = 10000	O(n)
f1(X)	0.0003 millisecondes	0.0005 millisecondes	0.0026 millisecondes	0.0213 millisecondes	cn
f2(X)	0.0008 millisecondes	0.0261 millisecondes	2.2032 millisecondes	193.1723 millisecondes	cn^2
f3(X)	0.0036 millisecondes	3.0805 millisecondes	2850.1415 millisecondes	1812197.6276 millisecondes	cn^3
f4(X)	0.0006 millisecondes	0.0127 millisecondes	1.1134 millisecondes	115.8961 millisecondes	$cn^2 - cn + c$
f5(X)	0.0209 millisecondes	2803.4012 millisecondes	L'ordinateur ne suit pas	L'ordinateur ne suit pas	$cn^5 + cn^3 + c$
f6(X)	0.0048 millisecondes	19.1863 millisecondes	163014.2594 millisecondes	L'ordinateur ne suit pas	$cn^5 - cn^4 + 2cn^3 - 2cn^2 + cn$

2. Comparaison des résultats d'implémentation avec analyse

L'implémentation fonctionne mais le temps de traitement est trop long pour les cas suivants :

- f5(1000)
- f5(10000)
- f6(1000)
- f6(10000)

Si je réutilise mon indice d'incrémentation d'une boucle for dans une nouvelle boucle for pour la limite, celui-ci aura pour valeur n-1.

Ex :

For (int i = 0 ; i < 5 ; i ++)

Résultat i == 4

Fonctions :

- f1(X) : c0 + n * c1 = cn

  c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. c1 provient de l'incrémentation de somme.

- f2(X) : c0 + n * n * c1 = cn^2

  c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. le deuxième n est pour la deuxième boucle for. c1 provient de l'incrémentation de somme.

- f3(X) : $c0 + n * (n*n) * c1 = cn^3$

  c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. la condition d'arrêt est un n^2 car n*n. c1 provient de l'incrémentation de somme.

- f4(X): $c0 + n * (i * c1) = c0 + n* ((n-1) * c1) = c0 + n* (c1*n – c1) = cn^2 -cn + c$

  C'est une exécution en temps polynomiale de degré 2. c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. La deuxième boucle a pour condition int i. Int i a pour taille n-1. c1 provient de l'incrémentation de somme.

- f5(X): $c0 + n * (i^2 * (j * c1)) = c0 + n * ((n-1)^2 * ((n-1)^2-1) * c1) = cn^5 + cn^3 + c$

  C'est une exécution en temps polynomiale. c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. La deuxième boucle a pour condition int i. Int i a pour taille $(n-1)^2$ tandis int j coute $(n-1)^2-1$. c1 provient de l'incrémentation de somme.

- f6(X): $c0 + n * ((i^2 * (c1 + (k * c2)) = c0 + (n-1) * ((n-1)^2 * (c1 + (((n-1)^2 -1)* c2))$
  $= cn^5 -cn^4+2cn^3-2cn^2+cn$
  C'est une exécution en temps polynomiale. c0 est le constante pour initialiser la variable somme. n donne le nombre d'itérations dans la boucle for. La deuxième boucle a pour condition int i. Int i a pour taille $(n-1)^2$ tandis int j coute $(n-1)^2-1$ .c1 est une constante qui vérifie une condition. c2 provient de l'incrémentation de somme.

Les exécutions en temps polynomiales sont bien moins couteuses. En effet on peut pour quand on compare la f5 avec f6, cette dernière est plus avantageuse. (Pour une exécution en temps polynomiale : degré 2 ~ cn^2 et degré 5 ~ cn^5). Celle qui coute le plus est donc la f5. Le temps d'exécution reste long une fois arrivé à cn^3.

PROJET#2 ALGORITHMIQUE

Mise en situation #1

1. Algorithme de la mise en situation

   Vérification du max
   - Si valeur = max alors affichage max fin de l’application
   - Si valeur > max alors :
     - Valeur n’est pas dans tableau alors fin application
   - Si valeur < max alors :
     Vérification si valeur recherchée est paire ou impaire
     - Si valeur % 2 = 0
       - Boucle for : en partant du minimum de la liste côté paire
         - Si valeur trouvée alors affichage et sortie de l’application
         - Si valeur non trouvée alors sortie de l’application
     - Si valeur % 2 != 0
       - Boucle for : en partant du minimum de la liste côté impaire
         - Si valeur trouvée alors affichage et sortie de l’application
         - Si valeur non trouvée alors sortie de l’application

2. Description et analyse des temps d’exécution (Meilleur, moyen et pire cas)

Meilleur cas : la valeur recherchée est le maximum
    c1 + c2 + c3
Pire cas : la valeur recherchée est inférieure au minimum du sous-tableau.
    c1 + c2 + c3 + n * c4
Moyen cas :
    c1 +c2 + c3

Mise en situation #2

1. Algorithme de la mise en situation

   Vérification du max
   - Si valeur = max alors affichage max fin de l’application
   - Si valeur > max alors :
     - Max est recueilli dans une variable temporaire
     - Valeur est placé à la place du Max
     Vérification si max est paire ou impaire
       - Si valeur % 2 = 0
         - Boucle for : en partant du maximum de la liste côté paire -> comparaison des valeurs successives avec le max

            - Si max < valeur comparée alors continuation de la boucle jusqu'à la fin de la liste
            - Si max > valeur comparée alors max affecté entre les deux.
    - Si valeur % 2 != 0
        - Boucle for : en partant du maximum de la liste côté impaire -> comparaison des valeurs successives avec le max
            - Si max < valeur comparée alors continuation de la boucle jusqu'à la fin de la liste
            - Si max > valeur comparée alors max affecté entre les deux.
- Si valeur < max alors :

  Vérification si valeur recherchée est paire ou impaire
    - Si valeur % 2 = 0
        - Boucle for : en partant du maximum de la liste côté paire -> comparaison des valeurs successives avec le max
            - Si max < valeur comparée alors continuation de la boucle jusqu'à la fin de la liste
            - Si max > valeur comparée alors max affecté entre les deux.
    - Si valeur % 2 != 0
        - Boucle for : en partant du maximum de la liste côté impaire -> comparaison des valeurs successives avec le max
            - Si max < valeur comparée alors continuation de la boucle jusqu'à la fin de la liste
            - Si max > valeur comparée alors max affecté entre les deux.

2. Description et analyse des temps d'exécution (Meilleur, moyen et pire cas)

Meilleur cas : valeur à insérer = max

C1

Pire cas

c1 + c2 + c3 + c4 + n * c5

cas moyen

Mise en situation #3

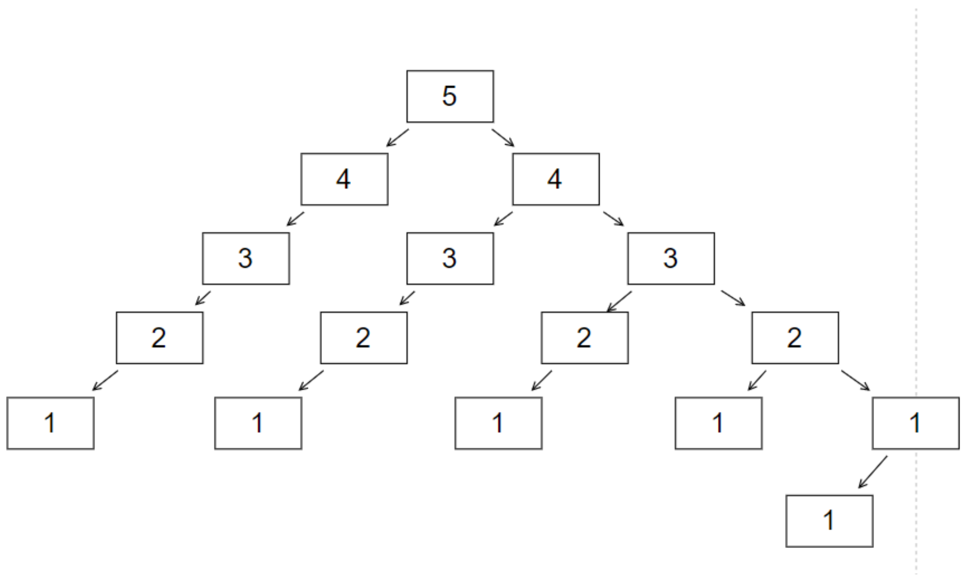
1. Algorithme de la mise en situation

N'arrivant à fournir l'algorithmie de cette mise en situation, j'ai directement codé le problème qui me servira pour réaliser l'analyse de la complexité algorithmique.

Code d'exécution sans la mise en forme en décalé

```
#include <iostream>

using namespace std;

long int Factoriel(int n)
{
    long int fact = 0;
    if (n == 1)
    {
        return 1;
    }
    else
    {
        cout << n << endl;
        return fact = (Factoriel (n-1)*n )* (Factoriel (n-1));
    }
}

int main()
{
    long fact = Factoriel(5);
    cout << fact << endl;

    return 0;
}
```

Dessin sur le fonctionnement de la mise en situation 3



2. Description et analyse des temps d’exécution (Meilleur, moyen et pire cas)

L’analyse de la fonction Factoriel

<u>Meilleur cas</u>

n = 1 ;

Le meilleur cas serait que n soit égal = 1 pour arrêter la récursion.

<u>Pire cas</u>

$O(n) = 4c + n^3 - 2n^2 - n$

<u>Moyens cas</u>

$4c + n^3 - 2n^2 - n$