APP1 Structure de données et complexité

06/01/2025 14:54 Après le tutorat d'ouverture

test matin

07/01/2025 10:45 Avant séminaire

Lecture de Algorithms and Data Structures

2.1 Analyse d'algorithme

2 façon de faire une analyse d'efficacité:

1. Regarder le montant de mémoire utiliser afin d'accomplir une tâche
2. Montant de temps que ça prend pour accomplir une tâche "Execution time or running time"

*Nous pouvons mesurer le temps d'exécution de la fonction **sum_of_n** en effectuant une analyse comparative (**benchmarking**).*

Ceci est un example de comment évaluer le temps dans Python:

```
import time

def sum_of_n_2(n):
    start = time.time()

    the_sum = 0
    for i in range(1, n + 1):
        the_sum = the_sum + i

    end = time.time()

    return the_sum, end - start

for i in range(5):
    print("Sum is %d required %10.7f seconds" % sum_of_n_2(10000))
```

Résultat:

```
Sum is 50005000 required 0.0018950 seconds
Sum is 50005000 required 0.0018620 seconds
Sum is 50005000 required 0.0019171 seconds
```

```
Sum is 50005000 required 0.0019162 seconds
Sum is 50005000 required 0.0019360 seconds
```

À cause la fonction, la vitesse d'execution est proportionnelle au chiffre mis en entré. 100 000 est 10 fois plus long à executer que 10 000.

Voici un code optimiser et son résultat:

```
import time

def sum_of_n_3(n):
    start = time.time()
    return (n * (n + 1)) / 2, time.time() - start



print("Sum is %d required %10.7f seconds" % sum_of_n_3(10000))
print("Sum is %d required %10.7f seconds" % sum_of_n_3(100000))
```

Résultat:

```
Sum is 50005000 required 0.00000095 seconds
Sum is 5000050000 required 0.00000095 seconds
```

Cette fontion n'est pas influencée par la valeur de `n`. C'est une bonne manière de tester l'efficacité *mais* le résultat ***peut varier*** d'un ordinateur à l'autre. Il ce peut qu'un ordinateur plus vieux soit plus rapide avce la première méthode (**à cause de la division**).

2.2 Big *O* notation