

Partie F

Cette partie porte sur la complexité algorithmique

1 Partie A

Voici des informations sur le temps d'exécution de six algorithmes en fonction de la taille des données mises en paramètre :

Algorithme 1
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 200Ko
Temps d'exécution : 20s

Algorithme 2
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 200Ko
Temps d'exécution : 10s

Algorithme 3
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 200Ko
Temps d'exécution : 6s

Algorithme 4
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 1000Ko
Temps d'exécution : 5s

Algorithme 5
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 101Ko
Temps d'exécution : 10s

Algorithme 6
Taille des paramètres : 100Ko
Temps d'exécution : 5s
Taille des paramètres : 200Ko
Temps d'exécution : 12,5s

1. Complexité des algorithmes

- Algorithme 1 : le temps est multiplié par 4 lorsque la taille des données double $\Rightarrow$ complexité quadratique $\mathcal{O}(n^2)$
- Algorithme 2 : le temps double lorsque la taille des données double $\Rightarrow$ complexité linéaire $\mathcal{O}(n)$
- Algorithme 3 : le temps augmente de 1 lorsque les données sont doublées $\Rightarrow$ complexité logarithmique $\mathcal{O}(\log n)$
- Algorithme 4 : le temps est indépendant de la taille des données $\Rightarrow$ complexité constante $\mathcal{O}(1)$
- Algorithme 5 : le temps double lorsque les données augmentent de 1. $\Rightarrow$ complexité exponentielle $\mathcal{O}(2^n)$
- Algorithme 6 : le temps est presque proportionnel à la taille $\Rightarrow$ complexité "quasi linéaire" $\mathcal{O}(n \log n)$

2. Estimation des temps d'exécution

Algorithme 1 -> Taille : 300Ko Temps d'exécution : 45s

La taille des données est multipliée par 3, donc le temps est multiplié par 9. $5 \times 3^2 = 45s$

Algorithme 2 -> Taille : 300Ko Temps d'exécution : 15s

La taille des données est multipliée par 3, donc le temps est multiplié par 3. $5 \times 3 = 15s$

Algorithme 3 -> Taille : 300Ko Temps d'exécution : 8s

La taille des données est multipliée par 3, donc on ajoute 3s. $5 + 3 = 8s$

Algorithme 4 -> Taille : 300Ko Temps d'exécution : 5s

le temps est indépendant de la taille des données

Algorithme 5 -> Taille : 110Ko Temps d'exécution :

le temps double lorsque les données augmentent de 1. Donc $5 \times 2^{10} = 5120s = 1h25min20s$

Algorithme 6 -> Taille : 300Ko Temps d'exécution : environ 20s (entre 15s et 45s)

C'est moins bien que linéaire mais mieux que quadratique. Donc entre 15s et 45s mais plus proche de 15s.

2 Partie B

```
def puissanceV1(n):  
    res = 1  
    for i in range(n):  
        res = res * 3  
    return res
```

```
def puissanceV2(n):  
    if n == 1:  
        return 3  
    elif n % 2 == 0:  
        tmp = puissanceV2(n//2)  
        return tmp * tmp  
    else:  
        tmp = puissanceV2(n//2)  
        return tmp * tmp * 3
```

1. Opération élémentaires

Algorithme puissanceV1

Pour calculer $3^8 = 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3$.

Il y a 7 multiplications

Algorithme puissanceV2

$3^9 = 3^4 \times 3^4$ (1 multiplication)

$3^4 = 3^2 \times 3^2$ (1 multiplication)

$3^2 = 3^1 \times 3^1$ (1 multiplication)

$3^1 = 3$

Il y a 3 multiplications

2. Complexité des algorithmes

Algorithme puissanceV1 La fonction utilise une boucle exécutée n fois et dans laquelle il y a une opération élémentaire. Complexité linéaire : $\mathcal{O}(n)$.

Algorithme puissanceV2

La fonction divise n par 2 à chaque appel récursif. Complexité logarithmique : $\mathcal{O}(\log n)$.

3. Algorithme le plus performant

L'algorithme **puissanceV2** est le plus performant car sa complexité logarithmique est bien plus efficace pour de grandes valeurs de n .