

Corrigé du QCM n°7

1. La programmation récursive est :

1. à privilégier par rapport à la programmation itérative
2. articulée autour d'un ou plusieurs cas de base et d'un ou plusieurs appels récursifs
3. une approche adaptée à la stratégie diviser pour régner
4. un mal nécessaire

La programmation récursive permet d'obtenir d'excellentes complexités sur des situations où la stratégie « diviser pour régner » s'applique : FFT (Fast Fourier Transform), tris, ... Un programme récursif contient un ou plusieurs cas de bases et un ou plusieurs appels récursifs (récursions). La programmation récursive ne permet pas de faire toujours mieux que la programmation itérative : l'empilement des récursions induit une complexité spatiale et impose des précautions relativement à la hauteur de la pile de récursion.

2. Identifier des algorithmes dont l'écriture récursive est plus simple que l'écriture itérative :

1. exponentiation rapide
2. calcul d'un coefficient binomial
3. recherche de doublons dans une liste
4. recherche d'un élément dans une liste

L'écriture récursive de l'exponentiation rapide ne requiert pas de connaître l'écriture binaire d'un nombre. Le calcul d'un coefficient binomial récursif permet de coder la relation $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$ sans variable à initialiser et sans boucle `for`.

3. On considère la fonction `fibonacci` suivante :

```
def fibonacci(n):  
    if n<=1:  
        return 1  
    else:  
        return fibonacci(n-1)+fibonacci(n-2)
```

Sa complexité temporelle est :

1. linéaire
2. exponentielle
3. quasi-logarithmique
4. logarithmique

La complexité temporelle de `fibonacci(n)` est en $O(\varphi^n)$ avec $\varphi = \frac{1 + \sqrt{5}}{2}$ ce qui en fait une complexité exponentielle.

4. L'algorithme de la FFT est en :

1. $O(N^2)$
2. $O(\log(N))$
3. $O(N \log(N))$
4. $O(N)$

Avec $N = 2^p$, on a
$$\frac{T(2^p)}{2^p} - \frac{T(2^{p-1})}{2^{p-1}} = O(1)$$

Après télescopage, on en déduit $T(2^p) = O(p2^p)$ d'où $T(N) = O(N \log(N))$.

5. Un tri *en place* :

1. **modifie directement la liste à trier**
2. renvoie une nouvelle liste triée
3. est de complexité spatiale $O(1)$
4. duplique la liste fournie en argument

Un tri en place modifie directement la liste à trier. Il ne renvoie pas de nouvelle liste et ne duplique pas la liste transmise en argument. Il n'est pas nécessairement de complexité spatiale en $O(1)$, par exemple à cause de l'empilement des récursions dans le cas d'un tri récursif.

6. Identifier le(s) tri(s) utilisant un pivot pour placer les éléments :

1. **le tri rapide pas en place**
2. le tri par insertion
3. **le tri rapide en place**
4. le tri fusion

Les algorithmes de tri rapide en place et pas en place utilisent un pivot pour la partition de la liste.

7. Quels sont les tris codés naturellement récursivement :

1. Tri par insertion
2. Tri bulles
3. **Tri rapide**
4. **Tri fusion**

Le tri bulles et le tri par insertion se codent itérativement tandis que le tri rapide et le tri fusion sont conçus récursivement.

8. Le tri rapide a une complexité temporelle :

1. en $O(n \log(n))$
2. **dans le meilleur des cas en $O(n \log(n))$**
3. en $O(n^2)$
4. **dans le pire des cas en $O(n^2)$**

Le tri rapide a une complexité temporelle en $O(n \log(n))$ dans le meilleur des cas et en $O(n^2)$ dans le pire des cas.

9. Le tri fusion a une complexité temporelle :

1. dans le pire des cas en $O(n^2)$
2. en $O(n \log(n))$
3. en $O(n^2)$
4. dans le meilleur des cas en $O(n \log(n))$

Le tri fusion a une complexité temporelle en $O(n \log(n))$ dans tous les cas.

10. Pour éviter le pire cas lors d'un tri rapide, on peut :

1. choisir un pivot au hasard
2. choisir un pivot à la fin
3. utiliser une version pas en place
4. utiliser une version en place

Si on choisit un pivot en début ou en fin de liste, le cas d'une liste triée par ordre croissant ou décroissant produit la situation la pire pour le tri rapide. Avec un choix de pivot au hasard, on évite ce pire cas. Le caractère en place ou pas en place n'a pas d'incidence sur ce phénomène.