

Corrigé du QCM n°6

1. La quantité $O(n) + O(\log(n))$ est un :

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

On a $O(\log(n)) = O(n)$ puisque $\frac{\log(n)}{n} \xrightarrow{n \rightarrow \infty} 0$ par croissances comparées. Par conséquent, par sommation de O , on a $O(n) + O(\log(n)) = O(n)$. On ne peut dire mieux : considérer $n + 0$ par exemple.

2. La quantité $O(n) + O(n^2)$ est un :

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

On a $O(n) = O(n^2)$ d'où $O(n) + O(n^2) = O(n^2)$ par sommation de O et on ne peut dire mieux : considérer $0 + n^2$ par exemple.

3. La quantité $\sum_{k=1}^n O(1)$ est un :

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

Par sommation de O , on a $\sum_{k=1}^n O(1) = O\left(\sum_{k=1}^n 1\right) = O(n)$.

4. La quantité $\sum_{k=1}^n O(k)$ est un :

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

Par sommation de O , on a $\sum_{k=1}^n O(k) = O\left(\sum_{k=1}^n k\right) = O(n^2)$.

5. La quantité $\sum_{i=1}^{n-1} \sum_{j=i+1}^n O(1)$ est un :

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

Par sommation de O , on a $\sum_{i=1}^{n-1} \sum_{j=i+1}^n O(1) = O\left(\sum_{i=1}^{n-1} \sum_{j=i+1}^n 1\right) = O\left(\sum_{i=1}^{n-1} (n-i)\right) = O\left(\sum_{k=1}^{n-1} k\right) = O(n^2)$.

6. Identifier un ou des algorithmes en complexité logarithmique :

1. Exponentiation rapide
2. Recherche dichotomique dans une liste triée
3. Algorithme de Horner
4. Test d'appartenance d'un élément à une liste non triée

Le test d'appartenance d'un élément à une liste est $O(n)$ dans le pire des cas. L'algorithme de Horner est en $O(n)$.

7. La fonction `binaire(n)` d'argument `n` entier renvoie son écriture binaire sous forme de liste.

```
def binaire(n):
    res, a = [], n
    while a > 0:
        res.append(a%2)
        a //= 2
    return res
```

La complexité de `binaire` est en :

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

L'écriture binaire de a initialisée à $\lfloor \log_2(n) + 1 \rfloor$ décroît de un à chaque passage dans la boucle puisque l'opération $a //= 2$ lui fait perdre le bit de poids faible. On en déduit une complexité en $O(\log(n))$.

8. La fonction `detect(elt,L)` d'argument `elt` et `L` une liste renvoie `True` si `elt` appartient à `L` et `False` sinon.

```
def detect(elt,L):
    for x in L:
        if elt==x:
            return True
    return False
```

Notant n la taille de la liste `L`, la complexité de `detect(elt,L)` est en :

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

Si `elt` est en première position dans `L`, la boucle est cassée dès le premier passage d'où une complexité en $O(1)$ dans le meilleur des cas. Si `elt` est en dernière position ou absent de `L`, la boucle est parcourue intégralement d'où une complexité en $O(n)$ dans le pire des cas.

9. La fonction `maxi(L)` d'argument `L` une liste non vide de nombres renvoie son plus grand élément. Notant n la taille de la liste `L`, la complexité de `maxi(L)` est en :

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

La fonction `maxi(L)` est en $O(n)$: on est obligé de parcourir intégralement la liste pour déterminer son plus grand élément.

10. La fonction `ecartmoy(L)` d'argument `L` une liste de taille ≥ 2 renvoie l'écart moyen entre deux éléments de `L` d'indices deux à deux distincts. Notant n la taille de la liste `L`, la complexité de `ecartmoy(L)` est en :

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

Notant $L = [x_0, \dots, x_{n-1}]$, il s'agit de calculer $\frac{2}{n(n-1)} \sum_{0 \leq i < j \leq n-1} |x_i - x_j|$. Il faut donc balayer tous les indices i et j tels que $0 \leq i < j \leq n-1$ ce qui induit une complexité en $O\left(\sum_{0 \leq i < j \leq n-1} 1\right) = O(n^2)$.