Corrigé du QCM n°6

1. La quantité $\mathrm{O}(n) + \mathrm{O}(n\log(n))$ est un :
   1. $\mathrm{O}(n^2)$
   2. $\mathrm{O}(n\log(n))$
   3. $\mathrm{O}(n)$
   4. $\mathrm{O}(\log(n))$

On a $\mathrm{O}(n) = \mathrm{O}(n\log(n))$ puisque $\dfrac{n}{n\log(n)} = \dfrac{1}{\log(n)} \xrightarrow[n\to\infty]{} 0$. Par conséquent, par sommation de O, on a $\mathrm{O}(n) + \mathrm{O}(n\log(n)) = \mathrm{O}(n\log(n))$. On ne peut dire mieux : considérer $0 + n\log(n)$ par exemple.

2. La quantité $\mathrm{O}(n\log(n)) + \mathrm{O}(n^2)$ est un :
   1. $\mathrm{O}(n\log(n))$
   2. $\mathrm{O}(n^2)$
   3. $\mathrm{O}(\log(n))$
   4. $\mathrm{O}(n)$

On a $\mathrm{O}(n\log(n)) = \mathrm{O}(n^2)$ puisque $\dfrac{n\log(n)}{n^2} = \dfrac{\log(n)}{n} \xrightarrow[n\to\infty]{} 0$ par croissances comparées. Ainsi, on trouve $\mathrm{O}(n\log(n)) + \mathrm{O}(n^2) = \mathrm{O}(n^2)$ par sommation de O et on ne peut dire mieux : considérer $0 + n^2$ par exemple.

3. La quantité $\displaystyle\sum_{k=1}^{n} \mathrm{O}(1)$ est un :
   1. $\mathrm{O}(\log(n))$
   2. $\mathrm{O}(n)$
   3. $\mathrm{O}(n^2)$
   4. $\mathrm{O}(n\log(n))$

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

4. La quantité $\displaystyle\sum_{k=1}^{n} \mathrm{O}(k)$ est un :
   1. $\mathrm{O}(n\log(n))$
   2. $\mathrm{O}(n)$
   3. $\mathrm{O}(n^2)$
   4. $\mathrm{O}(\log(n))$

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

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

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

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

6. Identifier un ou des algorithmes en complexité logarithmique :

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

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

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. $\mathrm{O}(n\log(n))$
2. $\mathrm{O}(n)$
3. $\mathrm{O}(\log(n))$
4. $\mathrm{O}(n^2)$

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 $\mathrm{O}(\log(n))$.

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):
    flag=False
    for x in L:
       if elt==x:
           flag=True
    return flag
```

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

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

La boucle `for` est systématiquement parcourue intégralement d'où une complexité en $O(n)$ sans distinction de cas.

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

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

La fonction `maxi_sort(L)` est en $O(1)$: il suffit de renvoyer le dernier élément puisque la liste est triée.

10. La fonction `ecartmoy(L)` d'argument `L` une liste de taille $\geqslant 2$ 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. $O(1)$
2. $O(n)$
3. $O(n\log(n))$
4. $O(n^2)$

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