Devoir sur table 1

Sans documents.

L'ensemble des questions demandant la rédaction d'un algorithme peuvent être rédigées en **pseudo-code, python, C, C++ ou Java**. La syntaxe du langage n'a pas d'importance tant que celle-ci reste **cohérente** et **compréhensible**.

Validation et calcul de la note : pour **valider** le partiel (obtenir 10/20), vous devez obtenir C ou plus sur au moins 4 des exercices de 1 à 6 et aucun E. L'exercice 7 est "bonus" : il compte dans la note finale mais ne compte pas pour la validation / non validation du partiel.

La note est ensuite calculée en prenant en compte votre validation (<10 ou >10) et la moyenne des notes obtenues aux 7 exercices selon la grille donnée. Par ailleurs,

— si vous obtenez 2 E ou plus : votre note maximale est 6
— si vous obtenez 1 E ou 3 D ou plus : votre note maximale est 9
— si vous obtenez 2 D : votre note maximale est 12.

Exercice 1 (Calcul de complexité)**.**
Donner la complexité de chacun des algorithmes suivants (sans justification). Chaque algorithme prend en entrée un entier positif n.

```
Algo_a :
    c <- 0
    Pour i allant de 1 à n:
        c <- c+2

Algo_b :
    c <- 1
    Tant que c < n:
       c <- c*2

Algo_c :
    c <- 0
    Pour i allant de 1 à n:
        c <- c * c
        Pour j allant de 1 à n:
                c <- c * 2
```

```
Algo_d :
    c <- 1
    Pour i allant de 1 à n:
        c <- c + 1
        Pour j allant de 1 à i:
            c <- c + 1

Algo_e :
    Tant que n > 0:
        n <- n//2

Algo_f :
    c <- 0
    Pour i allant de 1 à n:
        Pour j allant de 1 à i:
            Pour k allant de 1 à j:
                c <- c + 1
```

Grille d'évaluation indicative

A (20)	Toutes les complexité ont été données de façon correcte. La notation O a été bien utilisée.
B (16)	Les complexités sont toutes correctes mais quelques imprécisions sur la notation O.
C (11)	Entre 1 et 3 erreurs sur les complexités, mais la classe de complexité (logarithmique, linéaire /polynomiale ou exponentielle) est respectée.
D (8)	Entre 1 et 3 erreurs dont des erreurs de classe de complexité.
E (1)	4 erreurs ou plus

Exercice 2 (Conception d'algorithme)**.**
On cherche à évaluer une donnée numérique non entière $x > 1$ par un entier (le plus grand entier n tel que $n < x$). On ne peut pas lire directement la donnée, on n'a seulement accès à la fonction suivante :

```
infX(k) :
Renvoie True si k est strictement inférieur à la donnée x et False sinon.
```

(1) **Premier cas** : on sait que x est compris entre deux entiers $v1 < x < v2$. Écrivez un algorithme optimal qui prend en paramètre $v1$ et $v2$ et renvoie k, l'estimation entière de x. **Donnez sa complexité.** (On suppose que `infX` a une complexité $O(1)$).

(2) **Deuxième cas** : on ne sait rien de la donnée x. Écrivez un algorithme optimal qui renvoie k, l'estimation entière de x. **Donnez sa complexité.**

Remarque : vous avez le droit d'utiliser la première question pour résoudre la deuxième.

Grille d'évaluation indicative

A (20)	Les deux algorithmes répondent correctement au problème posé, ils sont écrits de façon claire et compacte. Les deux ont une complexité optimale et qui a été donnée correctement.
B (16)	Le principe général des deux algorithmes est le bon pour obtenir une complexité optimale cependant certains cas ont été oubliés ou des éléments non essentiels rajoutés (tests inutiles, écriture compliquée) ou bien on trouve des "petites" erreurs type indice, signes, etc. Au moins une des complexité est juste.
C (11)	Le principe général d'au moins un des algorithmes est bon avec une complexité optimale.
D (8)	Les deux algorithmes sont faux ou de complexité non optimale. Il y a cependant une tentative d'améliorer la complexité.
E (1)	Soit les deux algorithmes sont faux ou manquants soit ils sont justes mais avec une complexité non optimale sans aucune tentative d'amélioration.

Exercice 3 (Structures de données)**.**

(1) Quelle est la complexité de l'opération suivante : ajouter un élément dans un ensemble codé efficacement par table de hachage ?

(2) Je cherche une structure telle que je puisse stocker une liste d'éléments et accéder à chacun d'eux en $O(1)$ par son index dans la liste. Que dois-je choisir ?

(3) Une Pile est une structure qui possède deux opérations *push* qui ajoute un élément et *pop* qui supprime le dernier élément ajouté et le retourne. Si je code ma pile par une liste chaînée, ou dois-je ajouter / supprimer les éléments pour que la complexité des opérations *push* et *pop* soit $O(1)$?

Grille d'évaluation indicative

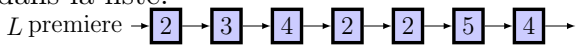
A (20)	C (11)	D (8)	E (1)
3 bonnes réponses	2 bonnes réponses	1 bonne réponse	0 bonne réponse

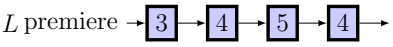
Exercice 4 (Listes chaînées)**.**
On utilise la structure de donnée suivante :

```
Structure Cellule :
    Entier valeur
    Cellule suivante

Structure File :
    Cellule premiere
```

Donner un algorithme qui prend en entrée une liste chaînée et une valeur a et qui supprime **toutes les occurences** de a dans la liste.

Par exemple, si la liste est 

Le résultat après le passage de l'algorithme supprimant 2 sera 

Grille d'évaluation indicative

A (20)	L'algorithme répond au problème posé de façon claire et exhaustive.
B (16)	Le principe général de l'algorithme est le bon. Cependant, il y a une ou deux erreurs / oublis sur les cas particuliers ou les conditions d'arrêts et vérification de pointeurs nuls. Il peut y avoir des petites erreurs dans la manipulation de la liste (erreur de nom, confusion cellule / valeur)
C (11)	Le principe général de l'algorithme est le bon mais il y de nombreuses erreurs ou oublis de cas particuliers.
D (8)	Le principe général de l'algorithme ne permet pas de répondre au problème, cependant les opérations de manipulation sur la liste chaînée sont écrites correctement.
E (1)	L'algorithme est faux ou inexistant et la manipulation de la liste n'est pas correcte.

Exercice 5 (Analyser un algorithme récursif)**.**
Voici un algorithme récursif

```
MonCalcul
Input :
    - n, un entier
Procédé :
    Si n = 0 ou n = 1 :
        Retourner 1
    Retourner MonCalcul(n-1) + MonCalcul(n-2)
```

(1) Calculer la valeur retournée pour les entrées : 1, 2, 3, 4.

(2) Sur quelles valeurs d'entrée cet algorithme termine-t-il ? (On ne demande pas les cas d'arrêt mais toutes les valeurs d'entrée où l'on obtient une réponse).

(3) Combien d'appels récursifs sont effectués pour la valeur d'entrée 5.

(4) Exprimer le nombre d'appels récursifs sous forme d'une fonction mathématique récursive.

(5) Donner la complexité de la fonction.

Grille d'évaluation indicative

A (20)	Toutes les réponses sont correctes et précises.
B (16)	Il y a quelques imprécisions ou bien les cas de terminaisons n'ont pas été donnés.
C (11)	La complexité et le calcul sur des valeurs données sont correctes.
D (8)	La complexité ou le calcul sur des valeurs données sont correctes.
E (1)	Ni la complexité, ni le calcul particulier n'est correct.

Exercice 6 (Concevoir un algorithme récursif)**.**

(1) Donner un algorithme **récursif** qui calcule le nombre de chiffre (en base 10) d'un nombre entier positif. On considère que 0 s'écrit avec un chiffre. Par exemple, pour 16546, on renvoie 5.

(2) Donner la complexité de votre algorithme.

Grille d'évaluation indicative

A (20)	L'algorithme est récursif et répond au problème donné. La complexité calculée est correcte (cohérente avec l'algorithme).
B (16)	La partie récursive de l'algorithme est correcte mais il y des erreurs sur les cas d'arrêts ou cas particuliers OU sur la complexité calculée.
C (11)	La partie récursive de l'algorithme est correcte mais il y a des erreurs sur les cas d'arrêts ou cas particuliers ET sur la complexité.
D (8)	Erreur sur la partie récursive de l'algorithme ou cas d'arrêt complètement absents.
E (1)	Algorithme non récursif ou récursif sans cas d'arrêts et faux.

Exercice 7 (Bonus)**.**
Donner un algorithme qui prend en paramètre un tableau de taille n contenant des valeurs distinctes et renvoie les 2^n sous ensemble possible. Par exemple, si on donne en entrée : `[1,2,3]` , l'algorithme renverra `[ [1,2,3], [1,2], [1,3], [1], [2,3], [2], [3], [] ]` . Dans chaque tableau, les nombres seront dans l'ordre initial, par contre, l'ordre des sous-ensemble dans le résultat n'a pas d'importance.

Remarque : pour un tableau vide, on renvoie le tableau contenant l'unique sous-ensemble vide donc `[ [] ]` .

(Vous pouvez utiliser la syntaxe que vous voulez pour tronquer / concaténer des tableaux).

Grille d'évaluation indicative

A (20)	L'algorithme est récursif et répond au problème donné.
B (16)	Le principe récursif est en partie correct mais il y a des erreurs dans son implantation et / ou dans les cas particuliers et conditions d'arrêt.
C (11)	Un algorithme récursif est proposé qui tente de répondre à la question posée même s'il ne répond pas aux spécifications du problème.