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 <- n
    Tant que c > 1:
        c <- c/2 (division entière)

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

Algo_c :
    c <- 0
    Pour i allant de 1 à n:
        Pour j allant de 1 à n:
                c <- c+1

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

Algo_f :
    c <- 1
    Pour i allant de 1 à n:
        c <- c + 1
    Pour i allant de 1 à n:
        c <- c + 1
        Pour i allant de 1 à n:
            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 a codé un dictionnaire à l'aide d'un tableau D contenant des mots rangés par ordre alphabétique.

(1) **Écrire une fonction qui renvoie l'indice d'un mot donné dans le dictionnaire (ou -1 si le mot est absent) de complexité optimale et donner sa complexité.** La fonction prend en paramètre le tableau D de taille n ainsi que le mot v. On a accès à une fonction de comparaison `compare(v1,v2)` de complexité $O(1)$ qui prend en paramètre 2 mots et renvoie -1 si $v1$ est avant dans l'ordre alphabétique, 0 si les deux mots sont égaux et 1 et $v1$ est après dans l'ordre alphabétique.

(2) La taille n du dictionnaire correspond au nombre de mots présents. Pour pouvoir rajouter des mots, le dictionnaire a aussi une capacité $c > n$. Lorsqu'on rajoute un mot et que la capacité est atteinte, elle doit être augmentée. Décrire une stratégie d'augmentation de la capacité qui permette d'obtenir une complexité optimale.

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. La complexité donnée est juste.
C (11)	Le principe général du premier algorithme 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 au début d'un tableau ?

(2) Je cherche une structure telle que je puisse ajouter et supprimer des éléments proches du début de la liste avec complexité faible, que dois-je choisir ?

(3) Je dois stocker un ensemble d'objets sans répétition, sans que l'ordre des objets soit important. Quelle structure dois-je utiliser pour que je puisse ajouter / supprimer des objets avec complexité optimale ?

Grille d'évaluation indicative

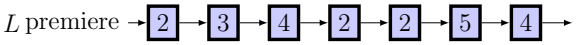
A (20)	C (11)	D (8)	E (1)
3 bonnes réponses	2 bonnes réponses	1 bonne réponse	0 bonne réponse

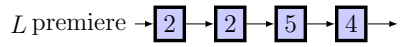
Exercice 4 (Listes chaînées)**.**
On utilise la structure de donnée suivante :

```
Structure Cellule :
    Entier valeur
    Cellule suivante

Structure File :
    Cellule premiere
```

Donner un algorithme qui prend en entrée une liste chaînée et un entier k et supprime les k premiers éléments de la liste.

Par exemple, si la liste est 

Le résultat après le passage de l'algorithme avec $k = 3$ 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 :
    - a, b, 2 entiers
Procédé :
    Si a = 0 ou si b = 0 :
        Renvoyer 0
    Si a < 0 :
        Renvoyer MonCalcul(-a,b)
    Si a > b :
        Renvoyer MonCalcul(b,a)
    Si a%2 = 0 :
        c <- MonCalcul(a/2,b)
        Renvoyer c+c
    Sinon :
        c <- MonCalcul((a-1)/2,b)
        Renvoyer b + c + c
```

(1) Calculer la valeur retournée pour les entrées : $a = 8$,$b = 10$
(2) Que calcule cet algorithme de façon générale ?
(3) 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).
(4) Combien d'appels récursifs sont effectués pour les valeurs $a = 8$,$b = 10$
(5) Exprimer le nombre d'appels récursifs sous forme d'une fonction mathématique récursive.
(6) 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)**.**

La suite de Syracuse d'un nombre k est donnée par l'algorithme suivant :

$$U_1 = k$$
$$U_{n+1} = \begin{cases} \frac{U_n}{2}, \text{ si } n \text{ est pair} \\ 3U_n + 1, \text{ si } n \text{ est impair.} \end{cases}$$

Par exemple, pour $k = 3$, on obtient $3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots$. La conjecture de Syracuse prétend que la suite atteint toujours la valeur 1. Ainsi, on définit la *taille de Syracuse* n d'un entier k par le plus petit n tel que $U_n = 1$. Pour $k = 3$, la taille est donc 8 car la 8ème valeur de la suite est 1. Pour 7, on obtient la suite : $7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$ soit une taille de 17.

Écrire une fonction récursive qui calcule la taille de Syracuse d'un nombre donné.

Grille d'évaluation indicative

A (20)	L'algorithme est récursif et répond au problème donné.
B (16)	La partie récursive de l'algorithme est correcte mais il y des erreurs sur les cas d'arrêts ou cas particuliers.
C (11)	L'appel récursif est correct mais le résultat est faux.
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 $n!$ permutations possible. Par exemple, si on donne en entrée : `[1,2,3]` , l'algorithme renverra `[ [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]` .

Aide : pour un tableau vide, on renvoie le tableau contenant l'unique permutation vide donc `[ [] ]` . Pour un tableau de taille 1, comme [1], on renvoie l'unique permutation qui est le tableau lui même donc `[ [1] ]` . Quand le tableau est plus grand, une permutation est faite d'un élément du tableau puis d'une permutation des éléments restant.

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

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.

Exercice 8 (Structures de données)**.**

(1) Quelle est la complexité de l'opération suivante : ajouter un élément au début d'un tableau ?

(2) Je cherche une structure telle que je puisse ajouter et supprimer des éléments proches du début de la liste avec complexité faible, que dois-je choisir ?

(3) Je dois stocker un ensemble d'objets sans répétition, sans que l'ordre des objets soit important. Quelle structure dois-je utiliser pour que je puisse ajouter / supprimer des objets avec complexité optimale ?

Grille d'évaluation indicative

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