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 3 des exercices de 1 à 6 (maximum 3 D ou E) et vous devez obtenir D ou plus sur au moins 5 des exercices de 1 à 6 (maximum 1 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.

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

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

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

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

```
Algo_c :
    c <- 1
    Tant que c < n:
        c <- c + 2

Algo_d :
    c <- n
    Tant que c > 0:
        c <- c / 2

Algo_e :
    Pour i allant de 1 à n:
        Si i*i > n:
            Retourner i
```

```
Algo_f :
    c <- 0
    Pour i allant de 1 à n:
        Pour j allant de 1 à n:
            Pour k allant de 1 à n:
                Pour l 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)**.**
Problème : tester si un entier n est un carré parfait, c'est-à-dire est-ce qu'il existe k tel quel $k \times k = n$. Par exemple : 0, 1, 4, 9 et 16 sont des carrés parfait mais pas 2 et 5.

Donner **deux** algorithmes qui répondent au problème. Les deux doivent avoir une complexité **sous-linéaire**, c'est-à-dire inférieure stricte à $O(n)$. Cependant l'un des deux sera beaucoup plus efficace que l'autre.

Remarque : si n n'est pas un carré parfait, il existera k tel que $k \times k < n$ et $(k+1) \times (k+1) > n$.

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 sous-linéaire et l'un des deux est optimal.
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.
C (11)	Le principe général d'au moins un des algorithmes est bon et donne une complexité sous linéaire.
D (8)	Les deux algorithmes sont faux ou de complexité linéaire. 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 de taille n.
(2) Je cherche une structure telle que je puisse ajouter un élément ou supprimer n'importe quel élément en $O(1)$, que dois-je choisir ?
(3) J'ai codé une liste de valeurs par une liste chaînée, je dois rajouter de nombreux éléments. Pour que la complexité soit optimale, dois-je les ajouter en début ou fin de liste ?

Grille d'évaluation indicative

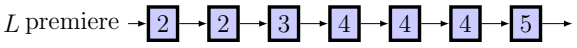
A (20)	C (11)	D (8)	E (1)
3 bonnes réponses	2 bonnes réponses	1 bonne réponse	0 bonne réponse

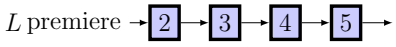
Exercice 4 (Listes chaînées)**.**
On utilise la structure de donnée suivante :

```
Structure Cellule :
    Entier valeur
    Cellule suivante

Structure File :
    Cellule premiere
```

Donner un algorithme qui prend en entrée une liste chaînée dont on suppose les valeurs ordonnées et qui supprime les occurrences multiples.

Par exemple, si la liste est 

Le résultat après le passage de l'algorithme 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:
        Retourner 1
    Retourner 2*MonCalcul(n-1)
```

(1) Calculer la valeur retournée pour les entrées : 1, 2, 3.
(2) Sur quelles valeurs d'entrée cet algorithme termine-t-il et que calcule-t-il de façon générale ?
(3) Combien d'appels récursifs sont effectués pour la valeur d'entrée 10.
(4) Exprimer le nombre d'appels récursives sous forme d'une fonction mathématiques récursives.
(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 produit $a \times b$ de deux entiers donnés en utilisant **uniquement** des additions ou soustractions. Votre algorithme doit donner un résultat correct pour tous les entiers positifs ou négatifs.
(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)**.**
Le *shuffle* entre deux mots u et v est la liste des mots obtenus en mélangeant entre elles les lettres de u et v tout en conservant l'ordre des lettres des mots de départ. Par exemple, les mots du shuffle de `"bob"` et `"cat"` sont exactement ceux de la liste suivante

bobcat bocbat bocabt bocatb bcobat bcoabt bcoatb bcaobt bcaotb bcatob cbobat cboabt cboatb cbaobt cbaotb cbatob cabobt cabotb cabtob catbob.

Remarquez que les lettres de "bob" et "cat" sont toujours dans le même ordre. Ainsi, le mot "tbobca" n'appartient pas au shuffle car les lettres de "cat" ne sont plus dans le bon ordre.

Voici quelques propriétés du shuffle :

— Si l'un des deux mots est vide, le shuffle ne contient qu'un seul élément : l'autre mot. (Si les deux sont vides, il ne contient que le mot vide "").

— Sinon, un mot du shuffle commence soit par la première lettre de u, soit par la première lettre de v, la suite est donnée par le shuffle des lettres restantes.

Implanter un algorithme récursif qui prend en paramètres deux mots `u` et `v` et retourne la liste des mots de leur shuffle.

On pourra utiliser la syntaxe python pour les listes et les chaînes de caractères.

Liste vide : `L <- [ ]`

Ajout : `L.append(valeur)`

Première lettre d'un mot : `u[0]`

Concaténation de deux chaînes de caractère : `u + v`

Le chaîne de caractère obtenue en copiant les lettres de `u` à partir de l'indice 1 : `u[1 :]`

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.