be-OI 2011 **Finale** 30 mars 2011	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

Olympiades belges d'Informatique (durée : 1h40 maximum)

Ce document est le questionnaire de la finale des Olympiades belges d'Informatique 2011 pour **la catégorie secondaire**. Il comporte sept questions qui doivent être résolues en **1h40 au maximum**. Chaque question est accompagnée d'un temps de résolution, donné à titre purement indicatif.

Notes générales (à lire attentivement avant de répondre aux questions)

1. N'indiquez votre nom, prénom et école **que sur la première page**. Sur toutes les autres pages, vous ne pouvez écrire que dans les **cadres prévus** pour votre réponse. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille, sans quoi votre réponse ne pourra être corrigée.

2. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM... sont **interdits**.

3. Vos réponses doivent être écrites **au stylo ou au bic** bleu ou noir. Pas de réponses laissées au crayon. Si vous désirez des feuilles de brouillon, demandez-en auprès d'un surveillant.

4. Vous **devez** répondre aux questions ouvertes en **pseudo-code** ou dans l'un des **langages de programmation autorisés** (Java, C, C++, Pascal, Python et PHP). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation. Vous ne pouvez utiliser aucune fonction prédéfinie à l'exception de celles présentées à la page suivante.

5. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants ou les organisateurs. Toute question portant sur la compréhension de la question ou liée à des problèmes techniques ne peut être posée qu'aux organisateurs. Toute question logistique peut être posée aux surveillants.

6. Les participants **ne peuvent en aucun cas quitter leur place** pendant l'épreuve, par exemple pour aller aux toilettes ou pour fumer une cigarette. Selon le lieu où vous présentez l'épreuve, il peut vous être interdit de manger ou boire durant cette dernière.

7. Vous avez **exactement une heure et quarante minutes** pour répondre à ce questionnaire. Un **aide-mémoire** sur le pseudo-code et les conventions prises se trouve à la page suivante.

Bonne chance !

001

Questionnaire finale secondaire

Aide-mémoire du pseudo-code

Dans ce questionnaire, il y a des questions pour lesquelles vous devez compléter une partie de code. On vous demande de les compléter avec soit des *expressions*, soit des *instructions*. Les expressions ont une valeur, les instructions effectuent une action.

Les expressions possèdent une valeur et sont utilisées dans les calculs, pour modifier les valeurs des variable et dans les conditions. Certaines expressions ont comme valeur un nombre (entier ou réel) et d'autres un booléen (**`true`** ou **`false`**). Les instructions représentent des actions à exécuter. Un algorithme est une suite d'instructions.

Les expressions les plus basiques sont les entiers (comme 3, -12...) et les variables (comme x, sum...). On peut également construire des expressions avec des opérateurs comme le montre le tableau suivant.

opération	**notation**	**exemples**	
égalité	$=$	$3 = 2$ vaut **`false`**	$1 = 1$ vaut **`true`**
différence	$\neq$	$1 \neq 4$ vaut **`true`**	$3 \neq 3$ vaut **`false`**
comparaison	$>, \geq, <, \leq$	$1 > 3$ vaut **`false`**	$5 \leq 5$ vaut **`true`**
addition et soustraction	$+, -$	$3 + 2$ vaut 5	$1 - 9$ vaut -8
multiplication	$\times$	2×7 vaut 14	-2×3 vaut -6
division entière	$/$	$10/3$ vaut 3	$9/3$ vaut 3
reste de la division entière	$\%$	$10\%3$ vaut 1	$9\%3$ vaut 0
« non » logique	**`not`**	**`not`** $3 = 2$ vaut **`true`**	**`not`** $1 = 1$ vaut **`false`**
« et » logique	**`and`**	$3 = 3$ **`and`** $5 \leq 9$ vaut **`true`**	$1 \geq 3$ **`and`** $2 = 2$ vaut **`false`**
« ou » logique	**`or`**	$3 = 3$ **`or`** $12 \leq 9$ vaut **`true`**	$1 \geq 3$ **`or`** $2 \neq 2$ vaut **`false`**
accès à un élément de tableau	`[]`	soit tab valant `[1,2,3]`, alors tab`[1]` vaut 2	
appel à une fonction retournant une valeur		`size (`$list$`)`	`min (12, 5)`

Les fonctions que vous pouvez utiliser sont, en plus de celles définies dans l'énoncé et celles définies sur les tableaux ci-dessous, les fonctions `max (`a`, `b`)`, `min (`a`, `b`)` et `pow (`a`, `b`)` qui permettent respectivement de calculer le maximum entre deux nombres, le minimum entre deux nombres et la puissance (a^b).

Le tableau suivant reprend les instructions de base :

opération	**notation**	**exemple**
affectation	$\leftarrow$	$x \leftarrow 20 + 11 \times 2$
renvoi	**`return`**	**`return`** 42
appel à une fonction (qui ne renvoie rien)	nom de la fonction	`sort (`$list$`)`

En plus, il y a également des *instructions de contrôle* qui sont au nombre de trois : **`if`**–**`else`**, **`while`** et **`for`**. Ces instructions vont faire en sorte d'exécuter un groupe d'instructions ou non, en fonction de conditions. La notation **`for`** `(`$i \leftarrow a$ **`to`** b **`step`** k`) { [...] }` indique une boucle qui se répète tant que $i \leq b$ pour i initialisé à a et incrémenté de k à la fin de chaque itération.

Dans un algorithme, outre les variables permettant de stocker une valeur, on peut également utiliser des *tableaux* pour stocker plusieurs valeurs. Un tableau tab de taille n est indicé de 0 à $n - 1$. La notation tab`[`i`]` permet d'accéder au $(i + 1)^{\text{e}}$ élément du tableau. Ainsi, le premier élément du tableau est tab`[0]`. On peut créer un nouveau tableau tab de taille n, dont les éléments sont initialisés à 0, avec la notation $tab \leftarrow$ `newArray (`n`)`. On peut faire une copie d'un tableau tab de taille n vers un nouveau tableau $newTab$ avec la notation $newTab \leftarrow tab$.

À propos du cout des opérations

Les questions 3 et 5 vous demandent d'écrire des algorithmes efficaces afin d'obtenir le score maximum.

Lorsqu'on s'intéresse à la complexité temporelle d'un algorithme (une estimation du temps qu'il va prendre pour s'exécuter), une manière de procéder consiste à compter le nombre d'opérations élémentaires qu'il va effectuer. Toutes les instructions de base coutent une unité de temps. La création d'un tableau de taille n, et également la copie d'un tel tableau, coutent n unités de temps. Pour les instructions **`if`**–**`else`**, **`while`** et **`for`**, il faut compter une unité de temps pour le calcul de la condition. Puis, bien sûr compter les instructions qui seront exécutées.

Question 1 – Parenthésage (5 min)

Nous cherchons à écrire un algorithme qui permet de vérifier qu'une expression mathématique, écrite sous forme d'une chaine de caractères, est bien parenthésée. Pour cela, il est nécessaire qu'à toute parenthèse ouvrante « (» corresponde une parenthèse fermante «) » située plus loin dans l'expression. De plus, une parenthèse fermante doit également correspondre à une parenthèse ouvrante précédemment rencontrée.

Par exemple, la chaine de caractères « $(a + 2 + (c/4) - (1 + e))$ » est bien parenthésée, alors que les deux suivantes ne le sont pas : « $)a + (2 - x)/(1$ » et « $a + (2 - 3)/((3 + a) + 3$ ». On vous demande de compléter l'algorithme suivant.

```
Input  : • s, un tableau de caractères.
         • n, entier positif, la longueur du tableau de caractères.
Output : • renvoie true (vrai) si l'expression mathématique représentée par
           le tableau s est bien parenthésée (voir énoncé) et false (faux) sinon.

num ← 0

for (i ← 0 to n step 1)
{
    if (s[i] = '(')
    {
        [...]                                                    // Q1(a)
    }
    if (s[i] = ')')
    {
        [...]                                                    // Q1(b)
    }
}
return (num = 0) // renvoie true si num est égal à zéro et false sinon
```

Q1(a) **(de 1 à 5 instructions)**

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Q1(b) **(de 1 à 5 instructions)**

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Question 2 – Mon beau sapin... (10 min)

L'algorithme donné ci-dessous est sensé dessiner des sapins à l'écran. Un sapin est dessiné à l'écran à l'aide d'un empilement de triangles isocèles, eux-mêmes représentés en affichant des caractères `X`, comme ceci :

```
   X
  XXX
   X
  XXX
 XXXXX
   X
  XXX
 XXXXX
XXXXXXX
   X
```

La taille du sapin affiché dépend du paramètre n de l'algorithme, comme suit. Partant de bas en haut, le sapin est d'abord constitué d'un triangle dont la base est de longueur n. Le triangle venant ensuite aura une base de longueur $n-2$. On continue ainsi jusqu'au dernier triangle dont la longueur de la base vaut 3. Sur l'exemple donné ci-dessus, le paramètre n vaut 7, et le sapin est donc composé de trois triangles, à savoir, de bas en haut : un triangle de base 7, un de base 5 et un de base 3. Le sapin est complété par un tronc, comme sur l'illustration.

On vous demande de compléter cet algorithme pour qu'il soit correct par rapport à ce qui a été décrit ci-dessus. Pour ce faire, l'algorithme exploite une fonction `draw` qui reçoit deux paramètres x et y, et affiche à l'écran une ligne composée de x espaces suivis de y caractères `X` (et d'un retour à la ligne). L'affichage se fait bien entendu de haut en bas.

```
Input  : • n, entier strictement positif et impair, représentant la longueur de la base
           du triangle inférieur du sapin.
Output : • un sapin de taille n (voir énoncé) est dessiné à l'écran.

function sapin (n)
{
    base ← 3
    distance ← n/2 − 1
    while (base ≤ n)
    {
        numcar ← 1
        while ([...])                                          // Q2 (a)
        {
            espaces ← distance + [...]                         // Q2 (b)
            draw (espaces, numcar)
            numcar ← numcar + 2
        }
        base ← base + 2
        [...]                                                  // Q2 (c)
    }
    draw ([...], 1)                                            // Q2 (d)
}
```

Q2(a) **(une expression)**

. . . . . .

Q2(b) **(une expression)**

. . . . . .

Q2(c) **(une instruction)**

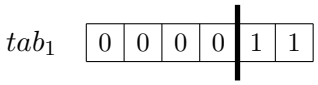
. . . . . .

Q2(d) **(une expression)**

. . . . . .

Question 3 – Faites la somme... (15 min)

Soit un tableau constitué de 0 et de 1 qui peut être découpé en deux parties, celle de gauche ne contenant que des 0 et celle de droite que des 1. Voici un exemple d'un tel tableau, avec l'endroit de découpe indiqué par une ligne épaisse :



On vous demande d'écrire un algorithme qui permet de calculer la somme des éléments du tableau. De plus, votre algorithme doit être **le plus efficace possible**.

Input : • tab, un tableau contenant une succession de 0 suivie d'une succession de 1.
• n, un entier positif, la taille du tableau tab.
Output : • la somme des éléments du tableau est renvoyée.
• le tableau tab n'a pas été modifié.

Seront pris en compte lors de la correction de cette question, outre le fait que l'algorithme calcule bien ce qui est demandé :

- le nombre d'opérations d'accès qui seront effectuées **dans le pire des cas** ;
- la simplicité de votre algorithme, qui devrait idéalement ne contenir qu'une seule boucle **`while`** et une seule instruction **`if`**–**`else`**, en plus d'instructions d'affectation de variables et d'une instruction **`return`**.

Q3

Question 4 – Faut que je range ma pile de livres... (15 min)

J'ai à ma disposition deux collections incomplètes de magazines, qui sont empilés et triés, le numéro le plus récent se trouvant au sommet de la pile. Plusieurs opérations sont possibles sur une pile. Soit la pile p, on peut faire :

- `top(`p`)` permet d'obtenir l'élément se trouvant au sommet de la pile p ;
- `pop(`p`)` permet d'obtenir l'élément se trouvant au sommet de la pile p et retire cet élément de la pile ;
- `push(`p`,`e`)` permet d'empiler l'élément e au sommet de la pile p ;
- `isEmpty(`p`)` permet de savoir si la pile p est vide ou non.

Voici un exemple de deux piles $p1$ et $p2$:

$p1$
1
3
5
8
8
12

$p2$
2
3
5
10

On vous demande d'écrire un algorithme qui, étant donné deux piles $p1$ et $p2$, va construire deux nouvelles piles $collection$ et $doublons$ contenant les mêmes magazines mais classés de façon différente. La première pile va contenir une collection, aussi complète que possible, de magazines. Elle ne pourra contenir qu'un seul exemplaire de chaque magazine et sera triée de sorte que le magazine le plus récent se trouve tout en bas et le plus ancien tout en haut. La seconde pile contiendra les doublons et est triée de la même manière que la première. Voici le résultat attendu pour l'exemple présenté ci-dessus.

$collection$
12
10
8
5
3
2
1

$doublons$
8
5
3

On vous demande de compléter l'algorithme suivant pour qu'il calcule ce qui est demandé.

```
Input  : • p1, p2 deux piles dont les éléments sont triés de manière croissante,
           le plus petit se trouvant au sommet de la pile.
Output : • une paire de piles (collection, doublons) contenant la collection et les
           doublons comme défini dans l'énoncé est renvoyée.

collection ← une nouvelle pile vide
doublons ← une nouvelle pile vide

while ([...])                                                    // Q4(a)
{
    if ([...])                                                   // Q4(b)
    {
        min ← pop (p1)
    }
    else
    {
        min ← pop (p2)
    }

    if ([...])                                                   // Q4(c)
    {
        push (collection, min)
    }
    else
    {
        push (doublons, min)
    }
}

return (collection, doublons)
```

Q4(a) **(une expression)**

.....

Q4(b) **(une expression)**

.....

Q4(c) **(une expression)**

.....

Question 5 – Multiplions (15 min)

Complétez l'algorithme suivant pour qu'il réalise la multiplication de deux nombres positifs représentés sous la forme de tableaux de chiffres et renvoie le résultat comme un tableau de chiffres.

Par exemple, si a et b valent respectivement :

a | 3 | 1 | 0 | b | 1 | 8 |

La solution sera :

r | 0 | 5 | 5 | 8 | 0 |

```
Input  : • a et b, deux tableaux non-vides de chiffres.
         • n et m, entiers positifs correspondant respectivement aux tailles de a et b.
Output : • Un tableau représentant le résultat de la multiplication des nombres
           représentés par les tableaux a et b est retourné. Les tableaux a et b ne
           sont pas modifiés.

function mult (a, b, n, m)
{
    r ← un nouveau tableau d'entiers de taille (n + m), initialisé avec des 0

    for (j ← m − 1 to 0 step −1)
    {
        for (i ← n − 1 to 0 step −1)
        {
            x ← a[i] × b[j]
            r[...] ←[...]                                          // Q5(a)
            r[...] ←[...]                                          // Q5(b)
        }
    }
    return r
}
```

Complétez cet algorithme.

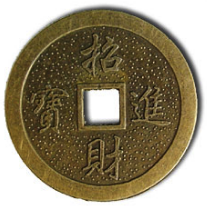
Q5(a) **(une affectation)**

. . . . . .

Q5(b) **(une affectation)**

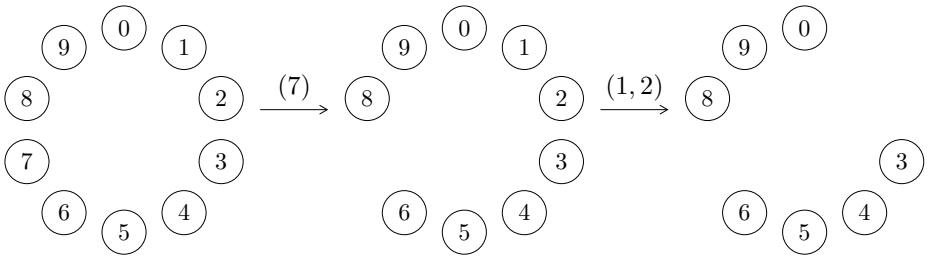
. . . . . .

Question 6 – Qui aura la dernière ? (15 min)



Dans le jeu suivant, une série de n (n étant pair) pièces chinoises sont placées en cercle sur la table. On va les numéroter de 0 à $n-1$, en les parcourant dans le sens horloger. Ce jeu se joue à deux joueurs. À chaque tour, le joueur choisit de retirer une ou deux pièces, sachant que s'il en prend deux, elles doivent être adjacentes. Le gagnant est le joueur qui est le dernier à jouer (c'est-à-dire qu'il ne reste plus aucune pièce après son tour).

Voici un exemple de début de partie :



Pour ce jeu, le second joueur (joueur 2) pourra toujours gagner, peu importe les pièces retirées par le premier joueur (joueur 1). L'algorithme suivant représente la stratégie adoptée par le second joueur. On vous demande de le compléter afin de lui assurer la victoire à tous les coups à partir du moment où il suit cette stratégie dès le début de la partie (n pièces sur la table).

```
Input  : • n, un entier pair positif > 3, le nombre de pièces.
         • (last_i, last_b), avec last_i le numéro de la pièce retirée par le joueur 1
           au tour précédent et last_b le nombre de pièces prises (1 ou 2). Dans le cas
           où 2 pièces sont retirées, last_i représente la première des deux (dans le
           sens horloger). Le jeu est dans une configuration dans laquelle c'est au
           joueur 2 de jouer et qu'il a appliqué la même stratégie depuis le début
           de la partie.
Output : • Un couple (i, b) avec i le numéro de la pièce qui doit être retirée et b le
           nombre de pièces qu'il faut prendre (1 ou 2). Dans le cas où 2 pièces sont
           retirées, i représente la première des deux (dans le sens horloger).

return ([...], [...])                                              // Q6(a,b)
```

Q6(a) **(une expression)**

. . . . . .

Q6(b) **(une expression)**

. . . . . .

Question 7 – Mettons de l'ordre (25 min)

Classer par ordre croissant les éléments d'un tableau d'entiers est un problème très classique d'algorithmique; de son efficacité dépend l'efficacité de nombreux autres problèmes très courants. Parmi ces algorithmes de tri, l'un des plus utilisés, car souvent le plus efficace, est l'algorithme *« Quick Sort »* qui fonctionne comme suit.

Pour commencer, on choisit une valeur du tableau (typiquement la dernière) qu'on appelle le *pivot*. Ensuite, on classe les valeurs du tableau en deux ensembles: le premier contenant toutes les valeurs strictement inférieures au pivot et le second contenant toutes les valeurs égales ou supérieures au pivot. Ce classement vise à obtenir un tableau qui contient, de gauche à droite, tous les éléments du premier ensemble, le pivot, et enfin tous les éléments du second ensemble. Finalement, on ré-applique ce même algorithme à chacun des deux sous-ensembles jusqu'à ce qu'on obtienne des ensembles ne contenant qu'un seul élément. Cette technique est aussi appelée *« diviser pour mieux régner »*.

L'algorithme est le suivant :

Input : • tab, un tableau d'entiers.
• $debut$ et fin, des indices du tableau, tels que $debut \leq fin$.
Output : • Les entiers entre les indices $debut$ et fin ($debut$ et fin compris) sont triés par ordre croissant.
• L'algorithme ne renvoie rien.

```
function quicksort (tab, debut, fin)
{
    if (fin − debut > 0)
    {
        pivot ← partition (tab, debut, fin)
        quicksort (tab, debut, pivot − 1)
        quicksort (tab, pivot + 1, fin)
    }
}
```

Cette algorithme fait appel à la fonction *partition* décrite ci-dessous. Pour des raisons d'efficacité, aucun tableau intermédiaire n'est utilisé dans *partition*. Le classement est effectué grâce à des échanges (`swap`). La fonction $swap(tab, i, j)$ permet d'intervertir les valeurs aux indices i et j du tableau tab.

```
Input  : • tab, un tableau d'entiers.
         • debut et fin, indices du tableaux tels que debut ≤ fin.
Output : • Les valeurs de tab entre debut et fin (inclus) ont été réordonnées de manière
           à ce qu'il existe une position j, debut ≤ j ≤ fin, telle que tab[j] contiennent
           le pivot (c'est-à-dire la valeur qui se trouvait initialement en tab[fin])
           et que toutes les cases i, debut ≤ i < j, contiennent des valeurs inférieures
           au pivot et que toutes les cases i, j < i ≤ fin, contiennent des valeurs égales
           ou plus grandes que le pivot.
         • tab est inchangé avant l'indice debut et après l'indice fin.
         • L'indice final du pivot est renvoyé.

function partition (tab, debut, fin)
{
    pivot ← fin
    x ← debut
    for (i ← debut to fin step 1)
    {
        if (s[i] < s[pivot])
        {
            swap (tab, i, [...])                                    // Q7(a)
            [...]                                                   // Q7(b)
        }
    }
    [...]                                                           // Q7(c)

    return x
}
```

Complétez cet algorithme (vous pouvez utiliser la fonction `swap` si cela vous semble nécessaire).

Q7(a) **(une expression)**

. . . . . .

Q7(b) **(une instruction)**

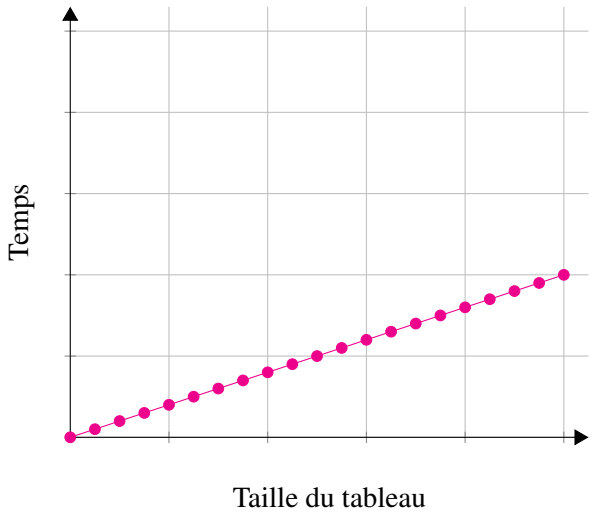
. . . . . .

Q7(c) **(une instruction)**

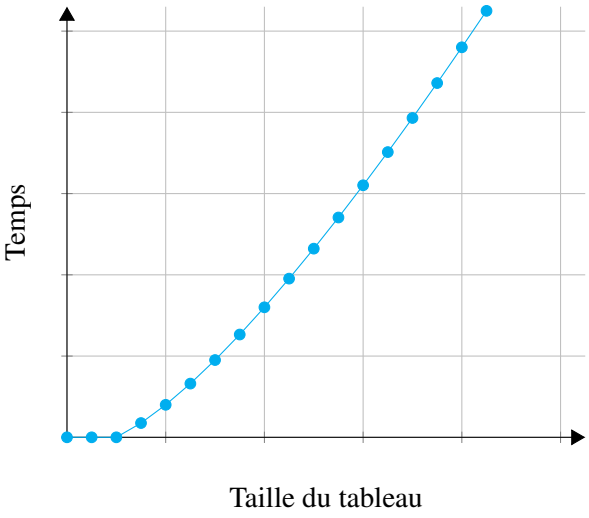
. . . . . .

Question 7 (suite)

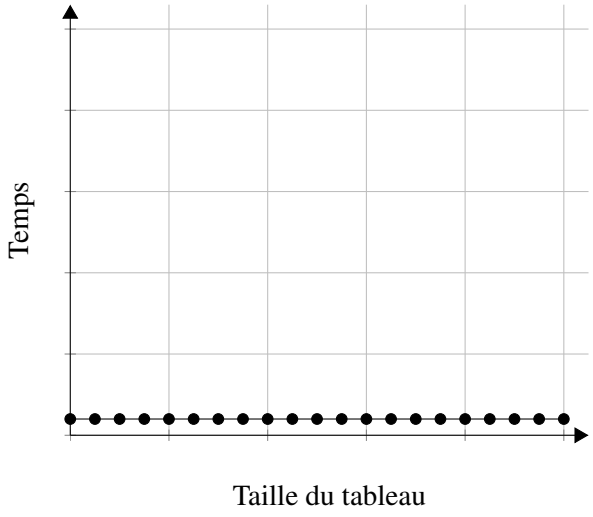
On aimerait maintenant connaître l'évolution du temps d'exécution de « *Quick Sort* » lorsque les tableaux traités sont de plus en plus grands. Pour cela, considérons que toutes les instructions (test de la condition d'un `if`, test de la condition d'un `while`, affectation...) ont un coût similaire et unitaire en temps. Choisissez parmi les quatre graphiques suivants celui qui pourrait correspondre à l'évolution du coût de « *Quick Sort* », en moyenne, en fonction de la taille du tableau à trier. Il n'y a pas de pénalité en cas de mauvaise réponse.



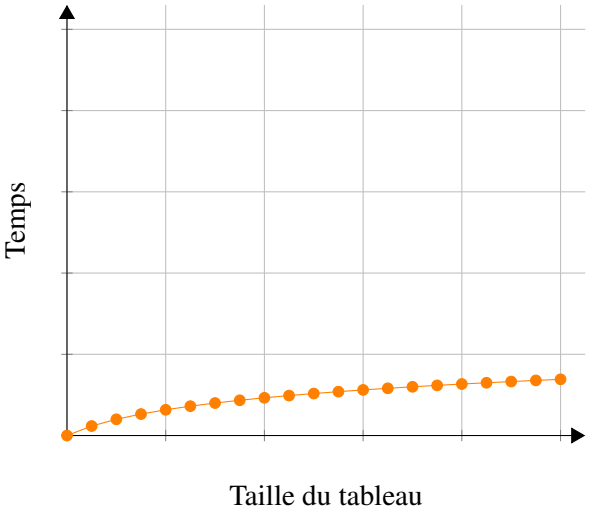
☐ **A**



☐ **B**



☐ **C**



☐ **D**