be-OI 2017 **Finale - SENIOR** samedi 11 mars 2017	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	300 **Réservé**

Olympiade belge d'Informatique (durée : 2h maximum)

Ce document est le questionnaire de la finale de l'Olympiade belge d'Informatique 2017. Il comporte 11 questions qui doivent être résolues en **2h au maximum**.

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

1. Vérifiez que vous avez bien reçu la bonne série de questions (mentionnée ci-dessus dans l'en-tête):
   - Pour les élèves jusqu'en deuxième année du secondaire: catégorie **cadets**.
   - Pour les élèves en troisième ou quatrième année du secondaire: catégorie **junior**.
   - Pour les élèves de cinquième année du secondaire et plus: catégorie **senior**.
2. N'indiquez votre nom, prénom et école **que sur la première page**.
3. Indiquez vos réponses sur les pages prévues à cet effet, à la fin du formulaire.
4. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille (au verso si nécessaire).
5. Écrivez de façon **bien lisible** à l'aide d'un **stylo ou stylo à bille** bleu ou noir.
6. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, ... sont **interdits**.
7. Vous pouvez toujours demander des feuilles de brouillon supplémentaires, au surveillant ou à votre enseignant.
8. Quand vous avez terminé, remettez la première page (avec votre nom) et les pages avec les réponses. Vous pouvez conserver les autres.
9. Tous les extraits de code de l'énoncé sont en **pseudo-code**. Vous trouverez, sur la page suivante, une **description** du pseudo-code que nous utilisons.
10. Si vous devez répondre en code, vous **devez** répondre en **pseudo-code** ou dans un **langages de programmation courant** (Java, C, C++, Pascal, Python, ...). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation.

Bonne chance !

L'Olympiade Belge d'Informatique est possible grâce au soutien de nos sponsors et de nos membres:

exellys IT talent integrators

docbyte Value your content

Aide-mémoire pseudo-code

Les données sont stockées dans des variables. On change la valeur d'une variable à l'aide de $\leftarrow$. Dans une variable, nous pouvons stocker des nombres entiers, des nombres réels, ou des tableaux (voir plus loin), ainsi que des valeurs booléennes (logiques): vrai/juste (**`true`**) ou faux/erroné (**`false`**). Il est possible d'effectuer des opérations arithmétiques sur des variables. En plus des quatre opérateurs classiques ($+$, $-$, $\times$ et $/$), vous pouvez également utiliser $\%$: si a et b sont des nombres entiers, alors a/b et $a\%b$ désignent respectivement le quotient et le reste de la division entière. Par exemple, si $a = 14$ et $b = 3$, alors: $a/b = 4$ et $a\%b = 2$. Par exemple, dans le code suivant, la variable age reçoit 20.

```
annee_naissance ← 1997
age ← 2017 − annee_naissance
```

Pour exécuter du code uniquement si une certaine condition est vraie, on utilise l'instruction **`if`** et éventuellement l'instruction **`else`** pour exécuter un autre code si la condition est fausse. L'exemple suivant vérifie si une personne est majeure et stocke le prix de son ticket de cinéma dans la variable $prix$. Notez les commentaires dans le code.

```
if (age ≥ 18)
{
    prix ← 8 // Ceci est un commentaire.
}
else
{
    prix ← 6 // moins cher !
}
```

Pour manipuler plusieurs éléments avec une seule variable, on utilise un tableau. Les éléments individuels d'un tableau sont indiqués par un index (que l'on écrit entre crochets après le nom du tableau). Le premier élément d'un tableau tab est d'indice 0 et est noté $tab[0]$. Le second est celui d'indice 1 et le dernier est celui d'indice $N - 1$ si le tableau contient N éléments. Par exemple, si le tableau tab contient les 3 nombres 5, 9 et 12 (dans cet ordre), alors $tab[0]= 5$, $tab[1]= 9$, $tab[2]= 12$. La longueur est 3, mais l'indice le plus élevé est 2.

Pour répéter du code, par exemple pour parcourir les éléments d'un tableau, on peut utiliser une boucle **`for`**. La notation **`for (`**$i \leftarrow a$ **`to`** b **`step`** k**`)`** représente une boucle qui sera répétée tant que $i \leq b$, dans laquelle i commence à la valeur a, et est augmentée de k à la fin de chaque étape. L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.

```
sum ← 0
for (i ← 0 to N − 1 step 1)
{
    sum ← sum + tab[i]
}
```

On peut également écrire une boucle à l'aide de l'instruction **`while`** qui répète du code tant que sa condition est vraie. Dans l'exemple suivant, on va diviser un nombre entier positif N par 2, puis par 3, ensuite par 4 ... jusqu'à ce qu'il ne soit plus composé que d'un seul chiffre (c'est-à-dire qu'il est < 10).

```
d ← 2
while (N ≥ 10)
{
    N ← N/d
    d ← d + 1
}
```

Question 1 – Échauffement

On considère le code suivant, dont le comportement dépend de la valeur $N \geq 1$.

Dans ce code, $j\%2$ est la valeur du reste de la division de j par 2.

```
Input : une valeur N ≥ 1

j ← 1
while (j < N)
{
  j ← j * 2
}

if (j%2 = 0)
{
  Afficher ``A''
}

if (j ≤ 100)
{
  if (j ≥ 1024)
  {
    Afficher ``B''
  }
  else
  {
    Afficher ``C''
  }
}
else
{
  if (j/2 > 511 and j/2 ≤ 512)
  {
    Afficher ``D''
  }
  else
  {
    Afficher ``E''
  }
}
```

Q1(a) [2 pts]	**Qu'est-ce que le programme affiche quand $N = 1$?**
Q1(b) [2 pts]	**Qu'est-ce que le programme affiche quand $N = 50$?**
Q1(c) [2 pts]	**Qu'est-ce que le programme affiche quand $N = 1024$?**
Q1(d) [2 pts]	**Qu'est-ce que le programme affiche quand $N = 1026$?**
Q1(e) [2 pts]	**Quelle est la valeur minimale de N pour laquelle A est toujours affichée ?**
Q1(f) [2 pts]	**Y a-t-il une lettre qui n'est affichée pour aucune valeur de N ? Si oui, laquelle ?**

Question 2 – En rang par deux !

Un professeur voudrait compter le nombre d'étudiants qui sont présents dans sa classe (il sait néanmoins que ce nombre est pair et une puissance de 2). Il propose trois techniques différentes pour les compter.

Pour la première technique, le professeur procède comme suit:

- Il initialise un compteur à 0;
- Il passe en revue *chaque étudiant*, et, à chaque étudiant, *ajoute* 1 *au compteur*;
- Quand il a terminé de passer en revue tous les étudiants, le compteur contient le nombre d'étudiants.

Pour la seconde technique, le professeur procède comme suit:

- Il initialise un compteur à 0;
- Il passe en revue tous les étudiants par *paire*, et, à chaque paire d'étudiants, *ajoute* 2 au compteur;
- Quand il a terminé de passer en revue tous les étudiants, le compteur contient le nombre d'étudiants.

Pour la troisième technique, le professeur ne compte plus lui-même mais met ses étudiants au travail:

- Il demande d'abord à tous ses étudiants de se lever, et leur demande à tous d'exécuter en même temps la procédure suivante:
  1. Chaque étudiant retient un compteur qui vaut initialement 1, ensuite:
  2. Tous les étudiants debout s'associent par paire;
  3. Dans chaque paire, le plus grand des deux étudiants ajoute à son propre compteur la valeur du compteur du plus petit étudiant. Le plus petit étudiant s'assied.
  4. On recommence au point 2 avec tous les étudiants restant debout.
- La procédure se termine quand il ne reste qu'un seul étudiant debout. Le professeur prétend que le compteur de ce dernier étudiant debout vaut le nombre d'étudiants présents.

Q2(a) [2 pts]	**Combien de sommes auront été effectuées par le professeur pour la première technique, s'il y a 512 étudiants ?**

Q2(b) [2 pts]	**Combien de sommes auront été effectuées par le professeur pour la deuxième technique, s'il y a 512 étudiants ?**

Q2(c) [3 pts]	**Combien de sommes auront été effectuées par le dernier étudiant debout pour la troisième technique, s'il y a 512 étudiants ?**

Q2(d) [2 pts]	**Combien de sommes auront été effectuées par le dernier étudiant debout pour la troisième technique, s'il y a n étudiants, en supposant que n est une puissance de 2 ?**

Question 3 – Le dictionnaire

On souhaite rechercher une définition dans le Robert (dictionnaire dont les entrées sont triées par ordre alphabétique, comme tout dictionnaire). On considère deux algorithmes différents pour trouver la définition associée à un mot.

Algorithme 1:

```
Input : une chaîne de caractères mot, le mot à chercher

// ouvrir le dictionnaire à la première page
while (pas à la fin du dictionnaire)
{
    if (mot est sur la page) {
        // lire la définition
        return
    }
    else {
        // tourner la page
    }
}
```

Algorithme 2:

```
Input :  une chaîne de caractères mot, le mot à chercher

// ouvrir le dictionnaire à la page du milieu
if (mot est sur la page) {
    // lire la définition
    return
}
else {
    // déchirer le dictionnaire en deux à la page où il est ouvert
    if (mot est avant la page du milieu dans le dictionnaire) {
        recommencer l'algorithme avec la première moitié
    }
    else if (mot est après la page du milieu dans le dictionnaire) {
        recommencer l'algorithme avec la seconde moitié
    }
    else { // Question (d)
        return
    }
}
```

On souhaite mieux comprendre le comportement de ces deux algorithmes, notamment quels sont le pire et le meilleur cas pour chacun d'eux (en terme de rapidité d'exécution).

Q3(a) [2 pts]	**Parmi les cas suivants, quel est le *meilleur* cas pour le premier algorithme ?**
☐	le mot commence par A et est sur la première page.
☐	le mot commence par A et est sur la dernière page.
☐	le mot est au milieu du dictionnaire.
☐	le mot commence par Z et est sur la première page.
☐	le mot commence par Z et est sur la dernière page.
☐	le mot n'est pas dans le dictionnaire.

Q3(b) [2 pts]	**Parmi les cas suivants, quel est le *meilleur* cas pour le second algorithme ?**
☐	le mot commence par A et est sur la première page.
☐	le mot commence par A et est sur la dernière page.
☐	le mot est au milieu du dictionnaire.
☐	le mot commence par Z et est sur la première page.
☐	le mot commence par Z et est sur la dernière page.
☐	le mot n'est pas dans le dictionnaire.

Q3(c) [2 pts]	**Parmi les cas suivants, quel est le *pire* cas pour le second algorithme ?**
☐	le mot commence par A et est sur la première page.
☐	le mot n'est pas dans le dictionnaire.
☐	le mot commence par A et est sur la dernière page.
☐	le mot est au milieu du dictionnaire.

Q3(d) [2 pts]	**Dans quel cas va-t-on entrer dans le dernier `else` du second algorithme ?**
☐	Quand le mot est dans la première moitié du dictionnaire.
☐	Quand le mot est dans la seconde moitié du dictionnaire.
☐	Quand le mot n'est pas dans le dictionnaire.
☐	Quand le mot commence par A.

Question 4 – La bibliothèque

Chaque livre possède un unique *International Standard Book Number*, ou ISBN, qui se compose de 10 chiffres, et dont le dernier chiffre peut aussi être la lettre 'X'. On ne choisit pas librement le dernier chiffre, mais il est calculé en utilisant le code suivant.

```
Input  : p: un tableau de 9 chiffres
Output : un chiffre ou bien le caractère 'X'.

a ← 0
j ← 10
for (i ← 0 to 8 step 1)
{
    a ← a + p[i] * j
    j ← j − 1
}
résultat ← 11 − (a%11)
if (résultat = 10)
{
    return 'X'
}
else
{
    return résultat%11
}
```

Le code suivant calcule la même valeur, mais d'une façon différente ! Complétez ce code de façon à ce qu'il calcule la même valeur que le précédent. Les entrées et les sorties sont les mêmes.

```
a ← 0
j ← 0
for (i ← 10 to 2 step −1)
{
    [...]                  //(a)
    [...]                  //(b)
}
résultat ← 11 − (a%11)
if ( [...] )               //(c)
{
    return résultat%11
}
else
{
    return 'X'
}
```

Q4(a) [2 pts]	**Quelle est l'instruction** `(a)` **?**

Q4(b) [2 pts]	**Quelle est l'instruction** `(b)` **?**

Q4(c) [2 pts]	**Quelle est la condition** `(c)` **?**

Ce code permet toujours de retrouver le nombre d'origine si un des chiffres est perdu. Essayons cela:

Q4(d) [4 pts]	**Vous recevez l'ISBN** 00070?0305**. Quel est le chiffre qui manque à la place du point d'interrogation ?**

Question 5 – Séries à regarder

John possède, sur son smartphone, une application qui lui permet d'encoder les séries télévisées qu'il regarde. Voici les informations que John doit encoder pour une série :

Nom : le nom de la série (par exemple "Game of Thrones").

Nombre de saisons : le nombre de saisons qui existent actuellement (par exemple 6 saisons).

Nombre d'épisodes par saison : le nombre d'épisodes dans une saison[1] (par exemple 10 épisodes par saison).

Durée d'un épisode : la durée en heures et minutes d'un épisode[2] (par exemple : 1h10).

Série finie : pour préciser si la série est finie ou si elle est toujours en cours de diffusion. Autrement dit, est-ce que la dernière saison a été diffusée ou non?

Voici les séries que John a encodées dans cette application :

Nom	Nbr de saisons	Episodes par saison	Durée épisode	Finie?
Game of Thrones	6	10	1h09	Non
Lost	6	20	0h42	Oui
Breaking Bad	5	12	0h50	Oui
The Walking Dead	7	13	1h06	Non
Pretty Little Liars	7	21	0h44	Non

Le but de cette application est de permettre à John de se souvenir du moment où il est arrivé en regardant ses diverses séries. Pour cela, lorsque John regarde une série, il doit encoder le numéro de la saison qu'il regarde[3], le numéro de l'épisode qu'il regarde[4] et le temps où il s'est arrêté de regarder.

Par exemple, pour la série "Game of Thrones", John en est à la saison 5, à l'épisode 8 et à une durée de 35 minutes dans cet épisode. De cette manière, John sait que la prochaine fois il pourra reprendre cet épisode à la 35ème minute afin de continuer à regarder sa série.

Et voici, pour chacune de ces séries, le moment où John s'est arrêté de regarder:

Nom	Saison en cours	Episode en cours	Timing
Game of Thrones	5	8	0h35
Lost	6	20	0h42
Breaking Bad	4	2	0h35
The Walking Dead	3	3	0h37
Pretty Little Liars	6	17	0h29

[1] On considère qu'il y a le même nombre d'épisodes dans chaque saison d'une même série.
[2] On considère que les épisodes d'une même série ont tous la même durée.
[3] On commence à compter à la saison 1 pour chaque série
[4] On commence à compter à l'épisode 1 pour chaque saison

Q5(a) [2 pts]	**Combien de minutes (au total) John doit-il encore regarder pour arriver à la fin de la dernière saison existante de *Game of Thrones* ?**

Q5(b) [2 pts]	**Combien de minutes (au total) John doit-il encore regarder pour arriver à la fin de la dernière saison existante de *Lost* ?**

L'application de John trie les séries encodées selon le temps qu'il reste à regarder pour arriver à la fin de la dernière saison existante. Moins il reste de temps à regarder et plus la série est prioritaire dans le tri. Si jamais deux séries se retrouvent à égalité de durée restante, alors une série qui est finie (voir tableau ci-dessus) est prioritaire par rapport à une série non finie. Et si deux séries finies sont à égalité, celles-ci sont triées selon leurs noms en ordre alphabétique. Les séries qui ont été entièrement regardées se retrouvent en fin de tri et sont triées selon leur nom dans l'ordre alphabétique.

Q5(c) [4 pts]	**Quel est l'ordre des séries une fois que celles-ci sont triées (séries les plus prioritaires en premier) ?**

John se met maintenant à regarder les *Simpson*, dont les caractéristiques sont les suivantes:

Nom	Nbr de saisons	Episodes par saison	Durée épisode	Finie?
Les Simpson	28	20	0h45	Non

Q5(d) [3 pts]	**À quel moment John devra-t-il s'arrêter *au plus tôt* de regarder les Simpson pour que cette série se retrouve en tête de liste ?**

Question 6 – Les châteaux de sable

Vous souhaitez préparer vos prochaines vacances à Blankenberge en étudiant de près la structure des tas de sable, afin de faire les plus beaux châteaux sur la plage. Votre ami Abel vous suggère de réaliser des simulations à l'aide d'un ordinateur:

Je considère une surface carrée sur laquelle je vais verser des seaux de sable, et je découpe cette surface en 5×5 *cellules carrées aussi. Je peux donc représenter ma surface par un tableau dont chaque case contient un certain nombre de seaux de sable. Par exemple, la matrice* M*:*

		2	2	
		3		

contient 3 seaux de sable dans la case $M[2][2]$ *et 2 seaux dans les cases* $M[1][2]$ *et* $M[1][3]$ *(pour rappel,* $M[i][j]$ *est la case à la ligne* i*, colonne* j*, et les lignes et colonnes sont numérotées à partir de* 0*).*

On peut ajouter des seaux de sable dans toutes les cases. Mais, si le nombre de seaux de sable dans une case atteint 4 (ou plus), ce tas de sable va s'écrouler: la case va se vider, et les 4 cases immédiatement adjacente recevront chacune un seau de sable. En d'autres mots, si $M[i][j]$ *contient* 4 *on met* $M[i][j]$ *à* 0 *et on ajoute* 1 *dans* $M[i-1][j]$*,* $M[i+1][j]$*,* $M[i][j-1]$*,* $M[i][j+1]$*, pour peu que ces cases existent (sinon les seaux tombent du plateau). Par exemple, si on ajoute un seau dans* $M[2][2]$ *dans l'exemple ci-dessus, on a maintenant 4 seaux, et on obtient:*

		3	2	
	1		1	
		1		

Évidemment, ces seaux de sable qui tombent dans d'autres cases pourraient les faire dépasser 4*, auquel cas il faut recommencer…jusqu'à ce qu'il n'y ait plus aucune case qui contienne* 4 *ou plus. Aussi vrai que je m'appelle Abel, je te garantis que cela arrivera toujours au bout d'un nombre fini d'étapes.*

Afin de bien comprendre comment fonctionne ce système vous essayez quelques exemples:

Q6(a) [4 pts] **Quelle matrice obtient-on si on dépose un seau dans la case** $M[1][4]$ **de la matrice suivante:**

			1	**3**
			2	**2**
	1			

?

Q6(b) [4 pts]

Quelle matrice obtient-on si on dépose un seau dans la case $M[2][2]$ de la matrice suivante:

			1	
	2	**3**	**3**	
			1	

?

Q6(c) [5 pts]

Quelle matrice obtient-on si on dépose un seau dans la case $M[2][2]$ de la matrice suivante:

1	**1**	**1**	**1**	**1**
1	**3**	**3**	**3**	**1**
1	**3**	**3**	**3**	**1**
1	**3**	**3**	**3**	**1**
1	**1**	**1**	**1**	**1**

?

Question 7 – Pliages

Soit une feuille de papier de x mm d'épaisseur. Si on plie cette feuille en deux, on obtient une épaisseur de $2x$ mm. Si on effectue un nouveau pli, l'épaisseur sera $4x$ mm... et ainsi de suite. L'algorithme suivant prend deux paramètres: la valeur de x et une épaisseur souhaitée g. Il calcule le nombre minimum de plis qu'il faut effectuer pour obtenir au moins g mm d'épaisseur.

```
Input  : x > 0, l'épaisseur du papier en mm.
         g > 0, l'épaisseur minimale souhaitée en mm.
Output : le nombre de plis à faire, au minimum, pour avoir au moins
         g mm d'épaisseur.

n ← 0

while (x < g)
{
   [...] // Instructions manquantes
}

return n
```

Q7(a) [6 pts]	**Quelles sont les (maximum 2) instructions manquantes ?**

Question 8 – La plus longue séquence

L'algorithme donné ci-dessous a été conçu pour afficher, étant donné un tableau tab contenant n entiers, la longueur de la plus longue portion contiguë dont le contenu des cases forme une suite strictement croissante d'entiers. Par exemple, pour le tableau suivant:

1	4	2	5	7	2	2

on constate l'existence de 10 séquences strictement croissantes, à savoir, classées par longueur,

- $[]$;
- $[1]$, $[4]$, $[2]$, $[5]$, $[7]$ (remarquez que la séquence $[2]$ apparaît trois fois);
- $[1, 4]$, $[2, 5]$, $[5, 7]$;
- $[2, 5, 7]$.

Pour cet exemple, l'algorithme va donc renvoyer la valeur 3 (la longueur de $[2, 5, 7]$). On vous demande de compléter l'algorithme ci-dessous afin qu'il calcule le bon résultat quelle que soit la taille du tableau tab.

Note : La fonction `max(a,b)` calcule le maximum entre a et b, c'est-à-dire la valeur la plus grande entre les deux.

```
Input  : tab, un tableau d'entiers.
         n, un entier positif (n ≥ 0), la taille du tableau tab.
Output : la longueur de la plus longue séquence strictement croissante est renvoyée.
         le tableau tab n'a pas été modifié.

length ← [...]                                  // (a)
m ← [...]                                       // (b)

for (i ← [...] to [...] step 1)                 // (c,d)
{
    if ([...])                                  // (e)
    {
        length ← 1
    }
    else
    {
        length ← length + 1
    }
    m ← max(m, length)
}
return m
```

Q8(a) [2 pts]	**Quelle est l'expression** `(a)` **?**

Q8(b) [2 pts]	**Quelle est l'expression** `(b)` **?**

Q8(c) [2 pts]	**Quelle est l'expression** `(c)` **?**

Q8(d) [2 pts]	**Quelle est l'expression** `(d)` **?**

Q8(e) [4 pts]	**Quelle est l'expression** `(e)` **?**

Question 9 – Croissance de population

Bart étudie l'évolution de plusieurs populations. Il mesure le nombre de personnes habitant dans un lieu donné au cours du temps, et voudrait utiliser ces valeurs pour prédire la population future. Bart fait appel à son amie Lisa, une informaticienne, car il voudrait un programme qui lui permettra de prédire la taille d'une population dans le futur (sur base des mesures qu'il a faites). Bart fournit à Lisa ses observations. Lisa analyse ces données et en déduit la façon dont la population évolue dans le temps. Elle écrit ensuite le programme qui calcule ces valeurs.

Les programmes de Lisa viennent d'arriver et Bart aimerait être certain qu'ils fournissent des résultats corrects. Il vérifie que les programmes calculent au moins les bonnes valeurs sur les observations qu'il a faites.

Chaque programme se présente sous la forme d'une fonction, qui a un paramètre t. Ce paramètre est un entier qui représente l'instant auquel on souhaite connaître la taille de la population (en particulier, quand $t = 0$ on a la taille initiale de la population). La fonction renvoie alors la taille de la population à l'instant t. Voici la première fonction:

```
function population1(t)
{
    if (t = 0)
    {
        return 0
    }
    else
    {
        return population1(t − 1) + 3
    }
}
```

Q9(a) [3 pts]	**Quelles sont les valeurs calculées par la fonction** `population1` **pour les valeurs** $t = 0$**,** $t = 1$**, etc jusqu'à** $t = 5$ **?**

La seconde fonction fournie par Lisa est la suivante:

```
function population2(t)
{
    if (t = 0)
    {
        return 1
    }
    else
    {
        return population2(t − 1) × 3
    }
}
```

Q9(b) [3 pts]	**Quelles sont les valeurs calculées par la fonction** `population2` **pour les valeurs** $t = 0$**,** $t = 1$**, etc jusqu'à** $t = 5$ **?**

Bart envoie maintenant une série de nouvelles observations à Lisa. Lisa doit donc écrire une fonction qui concorde avec ces valeurs. Voici les observations faites par Bart (la valeur p_0 est l'observation faite au temps $t = 0$, p_1 est l'observation faite au temps $t = 1$, etc):

$p_0 = 0$	$p_1 = 1$	$p_2 = 4$	$p_3 = 9$	$p_4 = 16$	$p_5 = 25$

On vous demande de compléter l'algorithme ci-dessous pour qu'il calcule les valeurs données par Bart:

```
function population3(t)
{
    if ([...])              // (a)
    {
        return [...]        // (b)
    }
    else
    {
        return [...]        // (c)
    }
}
```

Q9(c) [2 pts]	**Quelles sont les expressions** `(a)`, `(b)` **et** `(c)` **qu'il faut utiliser pour que les valeurs correspondent à l'algorithme ?**
☐	`(a)`: $t = 0$; `(b)`: 0 ; `(c)`: `population3(`$t-1$`) + `$t \times t$
☐	`(a)`: $t \leq 1$; `(b)`: t ; `(c)`: `population3(`$t-1$`)×2`
☐	`(a)`: $t = 0$; `(b)`: 0 ; `(c)`: `population3(`$t-1$`) + (2×`t`) - 1`
☐	`(a)`: $t < 1$; `(b)`: t ; `(c)`: `population3(`$t-1$`)×population3(`$t-1$`)`

Vous trouverez ci-dessous un nouvel algorithme incomplet proposé par Lisa. On vous demande à nouveau de le compléter de manière à ce que le résultat de l'algorithme concorde avec les mesures de Bart.

$p_0 = 0$	$p_1 = 1$	$p_2 = 1$	$p_3 = 2$	$p_4 = 3$	$p_5 = 5$	$p_6 = 8$

```
function population4(t)
{
  if (t < 2)
  {
    return [...]            // (d)
  }
  else
  {
    return [...]            // (e)
  }
}
```

Q9(d) [3 pts]	**Quelle est l'expression** `(d)` **?**

Q9(e) [3 pts]	**Quelle est l'expression** `(e)` **?**

Question 10 – Mettons de l'ordre

On considère l'algorithme *« Quick Sort »* qui a pour but de trier un tableau par ordre croissant et fonctionne comme suit. Pour commencer, on choisit une valeur du tableau (typiquement la dernière) qu'on appelle le *pivot*. Ensuite, on réalise une *partition* du tableau en trois parties: dans une certaine case j (qui doit être bien choisie), on place le *pivot*; du début du tableau à la position $j-1$, on place toutes les valeurs plus petites que le *pivot*; et de la position $j+1$ à la fin du tableau, on place toutes les valeurs plus grande que le *pivot*. Une fois que cette partition est faite, on ré-applique ce même algorithme à chacune des parties à gauche et à droite de j, jusqu'à ce qu'on ne doive appliquer l'algorithme que sur une seule case.

L'algorithme est donc le suivant :

```
Input  : tab, un tableau d'entiers.
         debut et fin, des indices du tableau, tels que debut ≤ 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)
    {
        indexpivot ← partition (tab, debut, fin)
        quicksort (tab, debut, indexpivot − 1)
        quicksort (tab, indexpivot + 1, fin)
    }
}
```

Pour découper le tableau en trois parties, et pour choisir la valeur de j, l'algorithme fait appel à la fonction `partition` décrite ci-dessous. Cette fonction fait appel à la fonction `swap`: `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 tableau tels que debut ≤ fin.
Output : La fonction réalise la partition du tableau entre les indices debut et fin
            comme décrit ci-dessus. Le reste du tableau n'est pas modifié.
         L'indice j du pivot est renvoyé.

function partition (tab, debut, fin)
{
    j ← debut
    for (i ← debut to fin step 1)
    {
        if (tab[i] < tab[fin]) // on utilise le dernier élément comme pivot
        {
            swap (tab, i, [...])                                    // (a)
            [...]                                                   // (b)
        }
    }
    [...]                                                           // (c)

    return j
}
```

On vous demande de compléter cet algorithme.

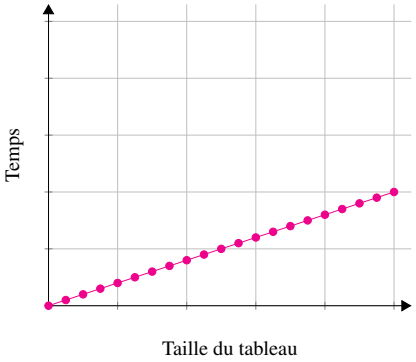
Q10(a) [4 pts]	**Quelle est l'expression** `(a)` **?**

Q10(b) [4 pts]	**Quelle est l'instruction** `(b)` **? Vous pouvez utiliser la fonction** `swap` **si nécessaire.**

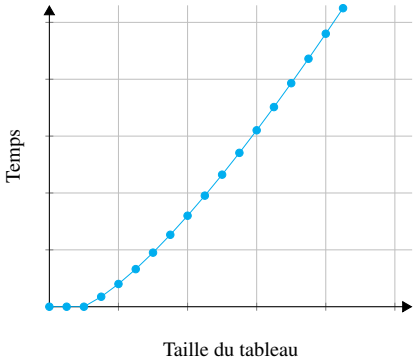
Q10(c) [4 pts]	**Quelle est l'instruction** `(c)` **? Vous pouvez utiliser la fonction** `swap` **si nécessaire.**

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 correspond à l'évolution du coût de « *Quick Sort* », en moyenne, en fonction de la taille du tableau à trier.

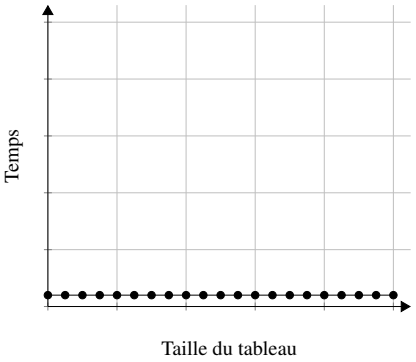
A:



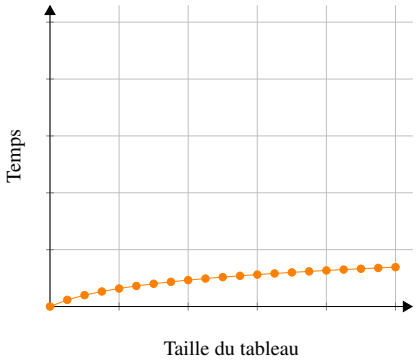
B:



C:



D:



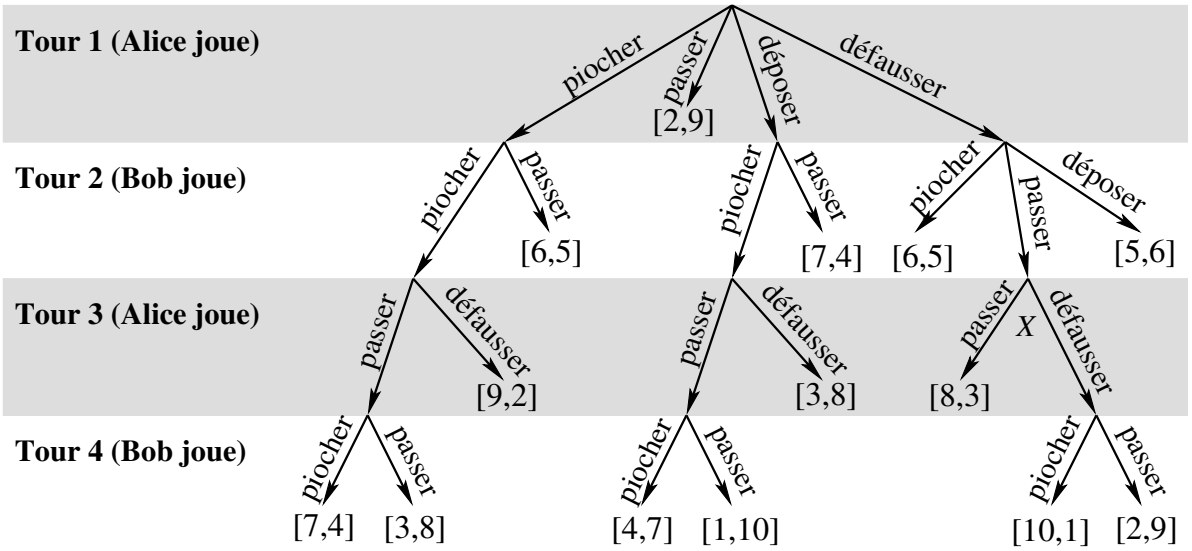
Q10(d) [4 pts]	**Lequel de ces quatre graphiques correspond à l'évolution du coût de « *Quick Sort* », en moyenne ?**
☐	A
☐	B
☐	C
☐	D

Question 11 – Jeu de stratégie

Alice et Bob jouent à un jeu de stratégie complexe dans lequel ils doivent construire une civilisation plus puissante que celle de leur adversaire. Une civilisation est déterminée par un ensemble de cartes posées par un joueur.

A la fin d'une partie, le score de chacun est déterminé en partageant 11 points de victoire entre les deux joueurs selon différents critères (2 points pour la puissance militaire, 3 points pour l'avancement technologique, etc.). Le joueur avec le score le plus élevé est déclaré vainqueur. Par exemple, une partie endiablée peut se terminer par la victoire sur le fil d'Alice sur un score de 6 points contre 5 points pour Bob (par convention, un tel résultat de fin de partie est noté $[6, 5]$).

Le tirage au sort désigne Alice pour commencer la partie. Elle décide de ne rien laisser au hasard et dessine toutes les situations possibles du jeu (voir ci-dessous). Pour son premier tour de jeu, Alice dispose de 4 coups possibles (*piocher*, *passer*, *déposer* et *défausser*). Ensuite, c'est au tour de Bob de jouer et ainsi de suite jusqu'à ce que le jeu soit terminé, ce qui est représenté par le score de fin de partie. Le nombre de coups possibles dépend de la situation du jeu. Vous pouvez ainsi remarquer que certaines parties peuvent se terminer assez rapidement, par exemple par une défaite d'Alice si celle-ci passe au premier tour.



Q11(a) [3 pts]	**Si la stratégie de Bob consiste à toujours *passer* invariablement et qu'Alice le sait, par quel coup Alice doit-elle commencer pour maximiser son score et quel sera celui-ci?**

Lors d'une autre partie, Alice et Bob jouent leur stratégie optimale et ils le savent tous les deux. Cela signifie qu'ils jouent du mieux possible, chacun tentant de maximiser son propre score. Par exemple, dans la situation du Tour 3 notée X ci-dessus, Alice choisit de *passer* (pour un score de 8) car si elle choisit de *défausser* (en espérant obtenir 10), Bob décidera de *passer* (et Alice n'obtiendra qu'un score de 2).

Q11(b) [3 pts]	**Si le deux joueurs jouent une stratégie optimale, par quel coup Alice doit-elle commencer pour maximiser son score et quel sera celui-ci?**

Donnez vos réponses ici !

Q1(a)	**Une ou plusieurs lettres**	/2
Q1(b)	**Une ou plusieurs lettres**	/2
Q1(c)	**Une ou plusieurs lettres**	/2
Q1(d)	**Une ou plusieurs lettres**	/2
Q1(e)	**Un entier**	/2
Q1(f)	**Une lettre ou bien « aucune »**	/2
Q2(a)	**Un entier**	/2
Q2(b)	**Un entier**	/2
Q2(c)	**Un entier**	/3
Q2(d)	**Une expression**	/2

Q3(a)		**Ne cochez qu'UNE seule réponse !**	/2
	☐	le mot commence par A et est sur la première page.	
	☐	le mot commence par A et est sur la dernière page.	
	☐	le mot est au milieu du dictionnaire.	
	☐	le mot commence par Z et est sur la première page.	
	☐	le mot commence par Z et est sur la dernière page.	
	☐	le mot n'est pas dans le dictionnaire.	

Q3(b)		**Ne cochez qu'UNE seule réponse !**	/2
	☐	le mot commence par A et est sur la première page.	
	☐	le mot commence par A et est sur la dernière page.	
	☐	le mot est au milieu du dictionnaire.	
	☐	le mot commence par Z et est sur la première page.	
	☐	le mot commence par Z et est sur la dernière page.	
	☐	le mot n'est pas dans le dictionnaire.	

Q3(c)		**Ne cochez qu'UNE seule réponse !**	/2
	☐	le mot commence par A et est sur la première page.	
	☐	le mot n'est pas dans le dictionnaire.	
	☐	le mot commence par A et est sur la dernière page.	
	☐	le mot est au milieu du dictionnaire.	

Q3(d)		**Ne cochez qu'UNE seule réponse !**	/2
	☐	Quand le mot est dans la première moitié du dictionnaire.	
	☐	Quand le mot est dans la seconde moitié du dictionnaire.	
	☐	Quand le mot n'est pas dans le dictionnaire.	
	☐	Quand le mot commence par A.	

Q4(a)	**Une instruction**	/2
. . . . .		

Q4(b)	**Une instruction**	/2
. . . . .		

Q4(c)	**Une condition**	/2
. . . . .		

Q4(d)	**Un chiffre**	/4
. . . . .		

Q5(a)	**Un nombre de minutes**	/2
. . . . .		

Q5(b)	**Un nombre de minutes**	/2
. . . . .		

Q5(c) **Une liste triée de noms de série.**	/4
Q5(d) **Un numéro de saison, un numéro d'épisode et un temps (saison, épisode et temps).**	/3
Q6(a) **Dessinez la matrice résultat**	/4
Q6(b) **Dessinez la matrice résultat**	/4
Q6(c) **Dessinez la matrice résultat**	/5
Q7(a) **Une ou deux instructions**	/6

Q8(a) **Une expression**	/2
Q8(b) **Une expression**	/2
Q8(c) **Une expression**	/2
Q8(d) **Une expression**	/2
Q8(e) **Une expression**	/4
Q9(a) **Une série de 6 valeurs**	/3
Q9(b) **6 valeurs**	/3

Q9(c)		**Ne cochez qu'UNE seule réponse !**	/2
	☐	(a) : $t = 0$; (b) : 0 ; (c) : `population3`$(t-1)$ + $t \times t$	
	☐	(a) : $t \leq 1$; (b) : t ; (c) : `population3`$(t-1) \times 2$	
	☐	(a) : $t = 0$; (b) : 0 ; (c) : `population3`$(t-1)$ + $(2 \times t)$ - 1	
	☐	(a) : $t < 1$; (b) : t ; (c) : `population3`$(t-1) \times$`population3`$(t-1)$	

Q9(d) **Une expression**	/3
Q9(e) **Une expression**	/3
Q10(a) **Une expression**	/4
Q10(b) **Une instruction**	/4
Q10(c) **Une instruction**	/4

Q10(d)		**Ne cochez qu'UNE seule réponse !**	/4
	☐	A	
	☐	B	
	☐	C	
	☐	D	

Q11(a) **Un coup et un nombre**	/3
. . . . .	

Q11(b) **Un coup et un nombre**	/3
. . . . .	