Examen d'algorithmique 2h

1. Algorithme à tester :

Q1) Vérifiez en prenant l'exemple donnée si le déroulement de l'algorithme de la procédure Tri( ) permet d'obtenir le résultat attendu. Pour cela vous présenterez vos variables sous la forme d'un tableau (voir page suivant) et vous expliquerez votre raisonnement.
(Il n'est peut être pas nécessaire de détailler toutes les lignes à chaque fois !!!).

```
1 :  PROCEDURE  Tri(ES tableau T( ) : réel, E NbreCase : entier)

     VARIABLES
2 :      i : entier
3 :      compteur, memoire : réel
4 :      FinTrie : booléen

5 :      pour i ← 2 à NbreCase pas 1

6 :          memoire ← T( i )
7 :          compteur ← (i – 1)

8 :          Répéter

9 :              FinTrie ← vrai

10:              Si  T(compteur) > memoire  alors
11:                  T(compteur + 1) ← T(compteur)
12:                  compteur ← compteur-1
13:                  FinTrie ← faux
14:              FSi

15:              Si  compteur < 1  alors
16:                  FinTrie ← vrai
17:              FSi

18:          Jusqu'à FinTrie = vrai

19:          T(compteur + 1) ← memoire

20:      FPour

21:  FinProcédure
```

Q2) Déterminez la complexité de cet algorithme : $O(n^2)$

2. Multiplication égyptienne :

2.1. Méthode :

Les égyptiens de l'antiquité savaient :

– additionner deux entiers strictement positifs,
– soustraire 1 à un entier strictement positif,
– multiplier par 1 et 2 tout entier strictement positif,
– diviser par 2 un entier strictement positif pair.

Voici un exemple qui multiplie 14 par 13 en utilisant uniquement ces opérations :

14 × 13	= 14 + 14 × (13 - 1)	= 14 + 14 × 12
	= 14 + (14 × 2) × (12 / 2)	= 14 + 28 × 6
	= 14 + (28 × 2) × (6 / 2)	= 14 + 56 × 3
	= 14 + 56 + 56 × (3 - 1)	= 70 + 56 × 2
	= 70 + (56 × 2) × (2 / 2)	= 70 + 112 × 1
	= 70 + 112	= 182

2.2. Algorithme :

Ecrire l'algorithme récursif qui permet la multiplication de 2 naturels suivant cette méthode :

Q3) Déterminer le ou les cas terminaux (cas ou la récursivité s'arrête).

Q4) Déterminer le ou les cas généraux (appel à la récursivité).

Q5) Donner le corps de la fonction suivante en utilisant un algorithme récursif :

Fonction MultiplicationEgyptienne(E a : entier, E b : entier) : entier

Q6) Question bonus : Déterminé la complexité de votre algorithme.

Cas terminal :
Si b = 1 alors a * b = a

Cas général, Si b > 1
– Si b est pair alors a * b = 2a * b/2
– Si b est impair alors a * b = a + a * (b − 1)

```
Fonction MultiplicationEgyptienne(E a : entier, E b : entier) : entier

        si b=1 alors
                retourner a
        sinon
                si  b mod 2 = 0  alors
                        retourner MultiplicationEgyptienne(2*a, b div 2)
                sinon
                        retourner a + MultiplicationEgyptienne(a, b-1)
                finsi
        finsi
FinFonction
```

Nom : _____ Prénom : _____

N° ligne	i	memoire	compteur	FinTrie	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]
1	?	?	?	?	1	3	0	2.2	-7	-6
					1	3	3	2.2	-7	-6
					1	1	3	2.2	-7	-6
					0	1	3	2.2	-7	-6
					0	1	3	3	-7	-6
					0	1	2.2	3	-7	-6
					0	1	2.2	3	3	-6
					0	1	2.2	2.2	3	-6
					0	1	1	2.2	3	-6
					0	0	1	2.2	3	-6
					-7	0	1	2.2	3	-6
					-7	0	1	2.2	3	3
					-7	0	1	2.2	2.2	3
					-7	0	1	1	2.2	3
					-7	0	0	1	2.2	3
					-7	-6	0	1	2.2	3

Correction Ex3:

Q1) A voir en fonction des explications

Q2) structure => 2 champs :

- un entier (nbre d'occurrences)
- un tableau d'entier (pour l'emplacement de l'occurrence)

Pour la procédure : ajouter un paramètre supplémentaire en ES (la structure)
Pour la procédure : renvoyer la structure

Modifier le code :

```
si q=m alors
        e c r i r e (" l e mot i f apparaît en position "+( i−m) )
        q ← pi ( q )
fin si
```

Par :

```
si q=m alors
        NomStructure.NbOcurrence ← NomStructure.NbOcurrence +1
        NomStructure.Position(NomStructure.NbOcurrence ) ←  i+m
         q ← pi ( q )
fin si
```

Ne pas oublier d'initialiser NomStructure.NbOcurrence ← 0 au départ