Terminaison d'un algorithme : Décrire un variant de boucle qui prouve ou non la terminaison des algorithmes suivants.

QCM1 : La fonction 'moy (lst) :' renvoie la moyenne de la liste 'lst'.

```
lst =[12,16,8,14,9,11,15,3,6,12,18]
def moy (lst):
    som = 0
    n = 0
    for i in range (len(lst)):
        som = som + lst[i]
        n = n + 1
    return round(som/n,2)
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : le variant de boucle n tend vers len(lst) et prouve que moy() se termine
- ☐ B : le variant de boucle 'len(lst)-i' tend vers 0 et prouve que moy() se termine
- ☐ C : le variant de boucle 'i' tend vers 0 et prouve que moy() se termine.
- ☐ D : le variant de boucle 'return moy' prouve que moy() se termine.

QCM2 : La fonction 'moy (note) :' renvoie la moyenne de la liste 'note'.

```
note = [12,16,8,14,9,11,15,3,6,12,18]
def moy (note):
    tot = 0
    n = len(note)
    i = 0
    while i < n :
        tot = tot + note[i]
        i = i + 1
    return round(tot/n,2)
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : le variant de boucle 'n' prouve que moy() se termine
- ☐ B : le variant de boucle 'i' prouve que moy() se termine
- ☐ C : le variant de boucle 'n-i' prouve que moy() ne se termine pas
- ☐ D : le variant de boucle 'n-i' prouve que moy() se termine.

QCM3 : La fonction 'test(compteur) :' renvoie l'état du compteur.

```
def test(compteur):
    compteur=50;
    while compteur>=50:
        print ("en cours de décomptage")
        test+=1;
    print ("décomptage terminé")

test(compteur)
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : le variant de boucle 'compteur>=50' prouve que test() se termine
- ☐ B : le variant de boucle 'test+=1' prouve que test() se termine
- ☐ C : le variant de boucle 'compteur>=50' prouve que test() ne se termine pas
- ☐ D : le variant de boucle 'test+=1' prouve que test() ne se termine pas

QCM4 : La fonction 'rech_seq(lst,x)' réalise une recherche séquentielle de l'élément x dans la liste lst.

```
lst = [12,16,8,14,9,11,15,3,6,12,18]
def rech_seq(lst,x):
    '''recherche séquentielle de x dans la liste lst'''
    c = 0
    t = len(lst)-1
    while c < t:
        c = c - 1
        if lst[c] == x :
            return True
    return False
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : le variant de boucle 't' tend vers 0 prouve que rech_seq() se termine.
- ☐ B : le variant de boucle 'c' tend à devenir nul ou négatif prouve que rech_seq() ne se termine pas
- ☐ C : le variant de boucle 'c' tend vers -∞ prouve que rech_seq() ne se termine pas
- ☐ D : le variant de boucle 'return False' prouve que rech_seq() se termine au pire des cas.

QCM5 : La fonction 'cher(lst,x)' réalise une recherche par dichotomie dans la liste 'lst'.

```
lst = [2, 27, 28, 32, 39, 40, 51, 52, 53, 54, 64, 66, 66, 74, 75, 82, 85, 86, 90, 92]
def rech (lst,x):
    c=0
    d=0
    f=len(lst)-1
    while d<=f:
        c=c+1
        m=(d+f)//2
        if lst[m]==x:
            return "l'element ",x," est présent à l'indice ",m," de la liste, trouvé en ",c," coups"
        elif x<lst[m]:
            f=m-1
        else :
            d=m+1
    return "l'element ",x," n'est pas présent dans la liste, trouve en ",c,"coups"
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : le variant de boucle 'd+f' tend vers +∞ prouve que cher() se termine
- ☐ B : le variant de boucle 'd-f' tend à devenir nul ou négatif prouve que cher() se termine
- ☐ C : le variant de boucle 'd-f' tend à devenir nul ou négatif prouve que cher() ne se termine pas
- ☐ D : le variant de boucle 'd+f' tend vers +∞ prouve que cher() ne se termine pas

Correction d'un algorithme : Décrire un invariant de boucle qui prouve ou non la correction des algorithmes suivants.

QCM6 : Soit la fonction mini() donnée ci-dessous

```
lst = [2, 15, 38, 44, 100, 50, 69, 91, 2, 100, 33]
def mini(lst) :
    ''' prend lst (list) en entrée et renvoie un tuple de la valeur minimum
        de lst et sa dernière position
    '''
    n = len(lst)
    i_m =0
    m = lst[0]
    for j in range (1,n):
        if lst[j] <= m:
            m = lst[j]
            i_m = j
    return (m,i_m)
```

Parmi les quatre propositions suivantes, laquelle est un invariant de boucle qui prouve que la fonction renvoie bien le tuple décrit dans la doc string .

- ☐ A : return (m,i_m)
- ☐ B : m = lst[j]
- ☐ C : lst[j] <= m
- ☐ D : lst[j] < m

QCM7 : Soit la fonction maxi() donnée ci-dessous

```
lst=[2, 15, 38, 44, 100, 50, 69, 91, 2, 100, 33]
def maxi(lst) :
    ''' prend lst (list) en entrée et renvoie un tuple de la valeur maximum
        de lst et sa première position
    '''
    n = len(lst)
    i_m = 0
    m = lst[0]
    for j in range (1,n):
        if lst[j] > m:
            m = lst[j]
            i_m = j
    return (m,i_m)
```

Parmi les quatre propositions suivantes, laquelle est un invariant de boucle qui prouve que la fonction renvoie bien le tuple décrit dans la doc string .

- ☐ A : return (m,i_m)
- ☐ B : m = lst[j]
- ☐ C : lst[j] > m
- ☐ D : i_m = j

QCM8 : La fonction 'tri_up' de tri par ordre croissant prend comme paramètre d'entrée la liste 'liste' et renvoie cette même liste triée par ordre croissant.

```
liste=[5,27,8,32,3,14,6,17,12,23,19]

def tri_up(liste):
    for i in range(len(liste)):
        for j in range(i+1,len(liste)):
            if liste[i]>liste[j]:
                liste[i],liste[j]=liste[j],liste[i];
    return (liste)
```

Parmi les quatre propositions suivantes, laquelle est un invariant de boucle qui prouve que la fonction réalise bien un tri par ordre croissant.

- ☐ A : return (liste)
- ☐ B : for j in range (i+1,len(liste)) ;
- ☐ C : liste[i] > liste[j]
- ☐ D : liste[i] = liste[j]

QCM9 : On souhaite modifier la fonction 'tri_up' et la renommer 'tri_dw' pour qu'elle réalise un tri par ordre décroissant.
Parmi les quatre propositions suivantes, laquelle modifie correctement l'invariant de boucle qui prouve que la fonction réalise bien un tri par ordre décroissant.

- ☐ A : liste[i] < liste[j]
- ☐ B : liste[i] > liste[j]
- ☐ C : for j in range (i-1,len(liste))
- ☐ D : return (liste-)

QCM10 : La fonction 'tri_inser2 (lst) :' trie et renvoie la liste 'lst'.

```
lst=[2, 15, 50, 2, 100, 33]
def tri_insert2 (lst) :
  for i in range (1,len(lst)):
    j = i
    x = lst[i]
    while j > 0 and lst[j-1] > x:
      lst[j] = lst[j-1]
      j = j - 1
      lst[j] = x
  return (lst)
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : l'invariant 'return lst' prouve que la fonction trie par ordre croissant
- ☐ B : l'invariant lst[j - 1] > x prouve que la fonction trie par ordre croissant
- ☐ C : l'invariant j = j - 1 prouve que la fonction trie par ordre décroissant
- ☐ D : l'invariant lst[j] = x prouve que la fonction trie par ordre décroissant

QCM11 : La fonction 'tri_inser3 (liste) :' trie et renvoie la liste 'liste'.

```
liste=[5,27,8,32,3,14,6,17,12,23,19]

def tri_inser3 (liste):
    for i in range (1,len(liste),1):
        j=i-1;
        while j>=0 and liste[j+1]>liste[j]:
            liste[j],liste[j+1]=liste[j+1],liste[j];
            j-=1;
    return (liste);
```

Parmi les quatre propositions suivantes, laquelle est vraie.

- ☐ A : l'invariant 'j = i-1' prouve que la fonction trie par ordre croissant
- ☐ B : l'invariant 'liste[j+1] > liste[j]' prouve que la fonction trie par ordre croissant
- ☐ C : l'invariant 'liste[j+1] > liste[j]' prouve que la fonction trie par ordre décroissant
- ☐ D : l'invariant 'return (liste)' prouve que la fonction trie par ordre décroissant