Nom :_____

Code permanent : _____

Numéro de place : _____

Directives pédagogiques :

- Inscrivez votre nom, prénom, code permanent et le numéro de votre place.
- Lisez attentivement toutes les questions et **répondez directement sur le questionnaire**.
- Seule l'utilisation d'un crayon ou stylo est permise, **aucune documentation, calculatrice, téléphone cellulaire, ordinateur, ou autre objet permis**.
- Cet examen contient 7 questions pour 110 points au total.
- Faites attention au temps car le barème est établi à 1 point par minute.
- Cet examen contient 10 pages, incluant 2 pages à la fin pour vos brouillons.
- **Écrivez lisiblement et détaillez vos réponses**.
- Vous avez 110 minutes pour compléter cet examen.

BONNE CHANCE !

1	/ 15
2	/ 5
3	/ 20
4	/ 15
5	/ 15
6	/ 25
7	/ 15
Total	/ 110

1. (15) Écrivez une fonction en Python (ou en pseudo-code), `renverse`, qui renverse les éléments d'une séquence (i.e. liste Python), tel que les éléments sont retournés dans une nouvelle séquence dans l'ordre inverse à la séquence initiale, e.g. `renverse( [1,2,3,4] )` retourne `[4,3,2,1]`. NB. vous ne pouvez pas utiliser la fonction Python `reverse` dans votre implantation, ni utiliser une pile.

```
def renverse( x ):
    tmp = list()
    for y in range( len( x ) - 1, -1, -1 ):
        tmp.append( x[y] )
    return tmp
```

2. (5) Considérez les instructions Python suivantes et donnez le résultat attendu des deux instructions `print`.

```
x = [1,2,3]
y = x
x.append( 4 )
y.append( 5 )
print( x )
print( y )
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

3. (20) Écrivez une fonction Python (ou pseudo-code), `somme_impairs`, qui calcule la somme des éléments impairs d'une séquence `S` passée en argument.

```
def somme_impairs( S ):
  #retourne la somme des éléments impairs de la séquence S
  n = len( S )
  somme = 0
  for j in range( 0, n ):
    if not( S[j] % 2 == 0 ):
      somme += S[j]
  return somme
```

a) (5) Donnez des expressions mathématiques exactes exprimant le nombre d'instructions élémentaires exécutées en meilleur cas, en moyenne et en pire cas.

ligne 3: n = len(S), assignation et longeur exécutées 1 fois chacune (2).
ligne 4: somme = 0, assignation exécutée 1 fois (1).
ligne 5: for j in range(0, n), assignation exécutée n fois (n).
ligne 6: if not(S[j] % 2 == 0), test, accès au tableau et modulo n fois (3n).
ligne 7: somme += S[j], accès, assignation et somme 0..n fois selon test:
0 fois en meilleur cas;
n/2 fois en moyenne;
n fois en pire cas.
ligne 8: return somme, sortie de fonction, 1 fois (1)

b) (5) Donnez l'ordre (grand-O) pour estimer la croissance en temps d'exécution en meilleur cas, en moyenne et en pire cas, en termes de `n`, soit le nombre d'éléments dans `S`.

Dans les 3 cas, O(n).

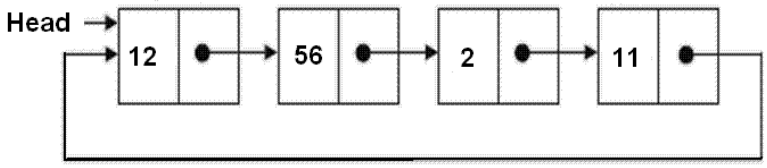
4. (15) Écrivez une fonction Python récursive (ou pseudo-code récursif), `multrecursif`, pour multiplier deux nombres, `n` et `m`, en utilisant uniquement les opérations d'addition et de soustraction.

```
def multrecursif2( n, m ):
    if m == 0:
        return n
    else:
        return n + multrecursif2( n, m-1 )

def multrecursif( n, m ):
    if m == 0 or n == 0:
      return 0
    if m < 0:
        n = -n
        m = -m
   return multrecursif2( n, m-1 )
```

5. (15) Écrivez une fonction en Python (ou en pseudo-code), `renverser`, qui renverse les éléments d'une séquence (i.e. liste Python), tel que les éléments sont retournés dans une nouvelle séquence dans l'ordre inverse à la séquence initiale, e.g. `renverser( [1,2,3,4] )` retourne `[4,3,2,1]`. Cette fois, utilisez une pile !

```
def renverser( S ):
    #ici on utilise la liste T comme une pile
    #insert simule l'opération push
    #pile explicite aussi acceptée

    T = list()
    for x in S:
        T.insert( 0, x )
    return T
```

6. (25) Une liste simplement chaînée circulaire (LSCC) est une liste simplement chaînée dont le prochain élément (`next`) du dernier noeud réfère au premier élément de la liste (`Head`).



a) (10) Écrivez une fonction Python (ou pseudo-code) pour compter le nombre d'éléments dans une LSCC.

```
def __len__( self ):
    if self._head == None:
        return 0
    count = 1
    tmp = self._head
    while not( tmp.next == self._head ):
        tmp = tmp.next
        count += 1
    return count
```

b) (15) Supposez des références `x` et `y` à des noeuds dans deux LSCC qui ne sont pas nécessairement les mêmes listes. Écrivez une fonction Python (ou pseudo-code) pour dire si `x` et `y` réfèrent à des noeuds appartenant à la même LSCC.

```
def dans_meme_liste( self, x, y ):
    if x == y:
        return True
    tmp = x.next
    while not( tmp == x or tmp == y ):
        tmp = tmp.next
    return tmp == y
```

7. (15) Remplissez le tableau ci-dessous en montrant à chaque ligne les changements d'états du tableau lorsqu'on le trie par la méthode du monceau. Commencez avec le monceau obtenu initialement. Dessinez sur la page suivante le monceau initial et ses altérations subséquentes. La dernière ligne contiendra les valeurs dans l'ordre croissant. (*NB. le nombre de lignes dans le tableau ne représente pas nécessairement le nombre exact d'états du tableau*.)

	16	3	1	15	18	19	6	21	8
Monceau initial =>									

Dessinez vos monceaux ici :

Brouilon 1

Brouilon 2