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 8 questions pour 110 points au total.
- Faites attention au temps car le barème est établi à 1 point par minute.
- Cet examen contient 17 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	/ 10
2	/ 10
3	/ 10
4	/ 10
5	/ 15
6	/ 15
7	/ 20
8	/ 20
Total	/ 110

1. (10) En assumant qu'il est possible de trier n nombres en temps $O(n \log n)$. Montrez qu'il est possible et donnez un algorithme en $O(n \log n)$ pour déterminer que trois ensembles, A, B et C ne possèdent pas d'élément commun aux 3 ensembles, c'est-à-dire qu'il n'existe pas d'élément x tel que x se retrouve à la fois dans A, B et C. Notez que x peut cependant se retrouver dans 2 des 3 ensembles.

2. (10) Écrivez une fonction récursive (en Python ou en pseudo-code), `minmax`, qui trouve les valeurs minimum et maximum d'une séquence sans utiliser de boucle, e.g. `minmax( [2,1,3,4,6,5] )` retourne `(1,6)`.

3. (10) Utilisez les structures de contrôle pour calculer la somme des nombres d'un ensemble n x n représenté par une liste de listes, e.g. `sum`([[1,2,3],[4,5,6], [7,8,9]]) = 45.

```
def sum( L ):
```

4. (10) Modifier le code de `ArrayStack` (Appendice A) pour que la capacité de la pile soit limitée à `maxlen` éléments, où `maxlen` est un argument optionnel du constructeur (défaut = `None`). Si push est appelée quand la capacité `maxlen` est atteinte, jetez une exception `Full`. Complétez les codes de `__init__` et `push` ci-dessous.

```
class ArrayStack( ):

def __init__( self,                                  ):
```

```
def __len__( self ):
    return len( self._A )

def is_empty( self ):
    return len( self._A ) == 0

def push( self, obj ):
```

```
def pop( self ):
    try:
        return self._A.pop()
    except IndexError:
        return None

def top( self ):
    return self._A.get( len( self._A ) - 1 )
```

5. (15) Considérez un arbre binaire dont les parcours dans l'ordre (in-order) et postfixé (post-order) visiteraient les clés dans les ordres EHEABOLZEDBPRBOX et EEHBLOAEBDRXOBPZ, respectivement.

   a) (5) Dessinez cet arbre.

   b) (5) Dans quel ordre seraient visitées les clés lors d'un parcours préfixé (pre-order)?

   c) (5) Dans quel ordre seraient visitées les clés lors d'un parcours en largeur (breadth-first)?

6. (15) Montrez comment on peut implanter l'ADT Stack (Appendice B) en utilisant une file avec priorités (Appendice C) et une variable entière additionnelle. Complétez le code ci-dessous.

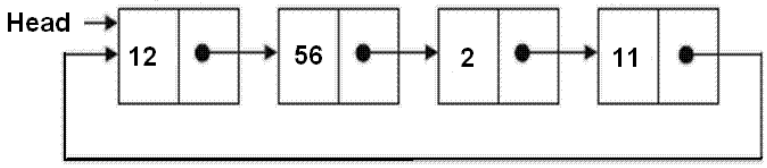
```
#ADT Stack "interface"
class PQStack:

def __init__( self ):

def __len__( self ):

def is_empty( self ):
```

```
    def push( self, element ):
```

```
    def pop( self ):
```

```
    def top( self ):
```

7. (20) 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 méthode (Python ou pseudo-code), `fusionner`, de la classe LSCC qui ajoute les noeuds d'une LSCC, M, à celle de L (e.g. `L.fusionner( M )`).

```
def fusionner( self, M ):
```

b) (10) Supposez des références à des noeuds, `x` et y, de LSCC qui ne sont pas nécessairement dans les mêmes listes. Écrivez une fonction (Python ou pseudo-code), `memeliste`, qui retourne **`True`** si `x` et `y` réfèrent à des noeuds appartenant à la même LSCC et **`False`** autrement.

```
def memeliste( x, y ):
```

8. (20) On veut construire un monceau-min en lisant en input les valeurs suivantes : 12, 15, 5, 11, 8, 2, 13, 19, 21, 6, 7, 9, 23, 16 et 4.

   a) (10) Montrez les étapes de construction et le monceau résultant de la procédure de construction vue en classe (`BuildHeap`). Quelle est la complexité en temps de cet méthode de construction ?

b) (10) Montrez les étapes de construction et le monceau résultant de l'insertion une à une des valeurs dans un monceau initialement vide. Quelle est la complexité en temps de cette méthode ?

Appendice A : ArrayStack

```
from DynamicArrayWithPop import DynamicArray

class ArrayStack( ):

    def __init__( self ):
        self._A = DynamicArray()

    def __len__( self ):
        return len( self._A )

    def is_empty( self ):
        return len( self._A ) == 0

    def push( self, obj ):
        self._A.append( obj )

    def pop( self ):
        try:
            return self._A.pop()
        except IndexError:
            return None

    def top( self ):
        return self._A.get( len( self._A ) - 1 )
```

Appendice B : Stack

```
#ADT Stack "interface"
class Stack:

    def __init__( self ):
        pass

    #return the number of elements in Stack
    def __len__( self ):
        pass

    def __str__( self ):
        pass

    def is_empty( self ):
        pass

    def push( self, element ):
        pass

    def pop( self ):
        pass

    def top( self ):
        pass
```

Appendice C : PriorityQueue

```
#ADT PriorityQueue
class PriorityQueue:

    #Nested class for the items
    class _Item:
        #efficient composite to store items
        __slots__ = '_key', '_value'

        def __init__( self, k, v ):
            self._key = k
            self._value = v

        def __lt__( self, other ):
            return self._key < other._key

        def __gt__( self, other ):
            return self._key > other._key

        def __str__( self ):
            pass

    #ADT and basic methods

    def __init__( self ):
        pass

    def __len__( self ):
        pass

    def is_empty( self ):
        return len( self ) == 0

    def min( self ):
        pass

    def add( self, k, x ):
        pass

    def remove_min( self ):
        pass
```

Brouilon 1

Brouilon 2