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. Il y a donc 10 points bonis.
- Faites attention au temps car le barème est établi à 1 point par minute.
- Cet examen contient 17 pages, incluant 3 pages à la fin pour vos brouillons.
- **Écrivez lisiblement et détaillez vos réponses**.
- Vous avez 100 minutes pour compléter cet examen.

BONNE CHANCE !

1	/ 20
2	/ 10
3	/ 15
4	/ 15
5	/ 10
6	/ 15
7	/ 15
8	/ 10
Total	/ 110

1. (20) Considérez les fonction `echelonner` et `echelonner2` en Appendice A. Le type numérique en python est immutable et l'utilisation de l'opération `*=` dans ce contexte crée une nouvelle instance et non pas la mutation de l'instance existante.

   a) (5) Quelle est la valeur de `x` après l'exécution des deux instructions suivantes:

```
x = [1,2,3]
echelonner( x, 5 )
```

   b) (5) Si `x` n'est plus égal à `[1,2,3]`, comment cela est-il possible que cette implantation de `echelonner` change le paramètre envoyé à l'appel ?

c) (5) Quelle est la valeur de `x` après l'exécution des deux instructions suivantes:

```
x = [1,2,3]
echelonner2( x, 5 )
```

d) (5) Expliquez pourquoi `echelonner2` modifie ou non la valeur de `x`:

2. (10) Les nombres d'opérations effectuées par l'algorithme A et B sont respectivement de $52n^2$ et $2n^3$. Déterminez n_0 tel que A est meilleur que B pour tout $n \geq n_0$.

3. (15) Décrivez un algorithme pour trouver le minimum et le maximum de n nombres en utilisant moins que 3n/2 comparaisons. Hint : En premier, construisez un groupe de candidats minimum et un groupe de candidats maximum.

4. (15) Écrivez une fonction récursive qui va générer tous les sous-ensembles d'un ensemble de *n* éléments `T` (sans répétition d'aucun sous-ensemble).

5. (10) Soit la classe `DynamicArray` vu en cours et partiellement donnée en Appendice B. Modifiez la fonction `__getitem__` de manière à implanter la sémantique d'une liste Python qui supporte l'indexage négatif.

6. (15) Montrez comment utiliser une pile `S` et une queue `Q` pour générer tous les sous-ensembles possibles d'un ensemble de *n* éléments `T` de manière non récursive.

7. (10) Décrivez un algorithme rapide, en O(*n*), pour renverser une liste simplement chaînée (voir l'interface `List` en Appendice C).

8. (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. (*NB. le nombre de lignes dans le tableau ne représente pas nécessairement le nombre exact d'états du tableau*.)

	15	7	1	16	4	15	6	21	8
Monceau initial =>									

Dessinez vos monceaux ici :

Appendice A : fonctions `echelonner` et `echelonner2`

```
def echelonner( donnees, facteur ):
    for j in range( len( donnees ) ):
        donnees[j] *= facteur
```

```
def echelonner2( donnees, facteur ):
    for val in data:
        val *= facteur
```

Appendice B : DynamicArray

```
import ctypes

class DynamicArray:

    #return a pointer to a memory area
    #that can store c contiguous python objects
    def _makeArray( self, c ):
        return( c * ctypes.py_object )()

    #create an array of 1 element
    def __init__( self ):
        self._n = 0
        self._capacity = 1
        self._A = self._makeArray( self._capacity )


    #returns the number of elements in the array
    def __len__( self ):
        return self._n

    #returns the capacity of the array
    def capacity( self ):
        return self._capacity

    #return the element at index k
    def __getitem__( self, k ):
        if not 0 <= k < self._n:
            raise IndexError( 'index out of bounds' )
        return self._A[k]
```

(et ainsi de suite…)

Appendice C : `List`

```
#ADT List "interface"
class List:

    def __init__( self ):
        pass

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

    # convert a List into a string:
    # elements listed between brackets
    # separated by commas
    # size and capacity of the data structure
    # indicated when relevant
    def __str__( self ):
        pass

    #add element at the end of list
    def append( self, element ):
        pass

    #remove the kth element
    def remove( self, k ):
        pass

    #find and return the rank of
    #element if in list, False otherwise
    def find( self, element ):
        pass
```

Brouilon 1

Brouilon 2

Brouilon 3