Name :_____

Code permanent : _____

Place number: _____

Directives :

- Write your name, first name, code permanent and place number.
- Read carefully all questions and **answer directly on the questionnaire**.
- You can only use a pen or pencil, **no documentation or other object is allowed**.
- This exam contains 8 questions for 110 points.
- Be careful with time as approximately 1 point = 1 minute.
- This exam contains 17 pages, including 2 draft pages at the end.
- **Write visibly and detail your answers when necessary**.
- You have 110 minutes to complete this exam.

GOOD LUCK !

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

1. (10) Assuming it is possible to sort n numbers in O($n \log n$) time, show that it is possible and give an algorithm to solve the three-way set disjointness problem in O($n \log n$) time. Three sets are three-way disjoint if there is no element common to all three sets, i.e, there exists no x such that x is in A, B, and C. Note that x can however be found in two of the three sets.

2. (10) Write a recursive function (in Python or pseudo-code), `minmax`, which finds the minimum and maximum values of a sequence without using any loop, e.g. `minmax( [2,1,3,4,6,5] )` returns `(1,6)`.

3. (10) Use standard control structures to copute the su of all numbers in an n x n data set represented as a list of lists, e.g. `sum( [[1,2,3],[4,5,6],[7,8,9]] ) = 45`.

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

4. (10) Modify `ArrayStack` (Appendix A) so that the stack's capacity is limited to `maxlen` elements, where `maxlen` is an optional argument of the constructor (défaut = `None`). If `push` is called when the stack is at full capacity, throw a `Full` exception. Complete the codes of `__init__` and `push` below.

```
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) Consider a binary tree for which in-order and post-order traversals would visit the keys EHEABOLZEDBPRBOX and EEHBLOAEBDRXOBPZ, respectively.

a) (5) Draw this tree.

b) (5) In which order the keys would be visited in pre-order traversal?

c) (5) In which order the keys would be visited in breadth-first traversal?

6. (15) Show how we can implement the Stack ADT (Appendix B) using a priority queue (Appendix C) and one additional integer variable. Complete the code below.

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

def __init__( self ):
```

```
def __len__( self ):
```

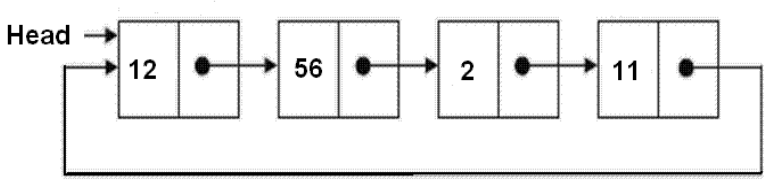
```
def is_empty( self ):
```

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

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

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

7. (20) A circular singly linked list (CSLL) is a single linked list for which the next element (`next`) of the last node refers to the first element of the list (`Head`).



a) (10) Write a method (Python or pseudo-code), `fusionner`, of the CSLL class that concatenate the nodes of a CSLL, M, to the one of L (e.g. `L.fusionner( M )`).

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

b) (10) Suppose references to CSLL nodes, `x` and y, which are not necessarily in the same lists. Write a function (Python ou pseudo-code), `memeliste`, which returns **`True`** if `x` and `y` refer to nodes of the same CSLL and **`False`** otherwise.

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

8. (20) We want to build a min-heap while reading from the input the following values: 12, 15, 5, 11, 8, 2, 13, 19, 21, 6, 7, 9, 23, 16, and 4.

   a) (10) Show the construction steps and the resulting heap when using the construction procedure seen in the course (`BuildHeap`). What is the complexity in time of this method?

b) (10) Show the construction steps and the resulting heap when inserting the values in a heap initially empty. What is the complexity in time of this method?

Appendix 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 )
```

Appendix 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
```

Appendix 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
```

Draft 1

Draft 2