Name:_____

Permanent code:_____

Place number:_____

Directives:

- Write your name, permanent code, and place number.
- Read all the questions and **answer directly on the questionnaire**.
- Use only a pen or pencil. **No documentation, calculator, phone, computer, or any other object allowed**.
- The exam contains 8 questions for 110 points. 10 are bonus points.
- Be careful with time. The exam is conceived so that 1 minute corresponds to 1 point approximatively.
- This exam contains 17 pages, including 3 pages at the end for draft.
- **Write legibly and detail your answers**.
- You have 100 minutes to complete this exam.

BONNE CHANCE !

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

1. (20) Consider the functions `echelonner` and `echelonner2` shown in Appendix A. The numerical type in Python is immutable and the use of the operation `*=` in this context creates a new instance, i.e. it does not mutate the existing instance.

   a) (5) What is the value of `x` after the execution of the two following instructions:

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

   b) (5) If `x` is not equal to `[1,2,3]`, how is this possible that this implementation of `echelonner` changes the parameter sent by the function call ?

c) (5) What is the value of `x` after the execution of the two following instructions:

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

d) (5) Explain why `echelonner2` modifies or not the value of `x`:

2. (10) The numbers of operations executed by algorithms A and B are $52n^2$ and $2n^3$, respectively. Determine n_0 such that A is better than B for all $n \geq n_0$.

3. (15) Describe an algorithm to find the minimum and maximum values of n numbers that uses less than 3n/2 comparisons. Hint : Try first to create a group of minimum candidates and a group of maximum candidates.

4. (15) Write a recursive function that generates all subsets of a set of *n* elements `T` (without any repeat of any).

5. (10) Consider the `DynamicArray` class saw in class and partially given in Appendix B. Modify the function `__getitem__` so that it implements the Python semantic of negative indexing.

6. (15) Show how to use a stack `S` and a queue `Q` to generate all the possible subsets of a set of *n* elements `T` in a non recursive manner.

7. (10) Describe an fast algorithm, in O(n), to reverse a singly linked list (see the Interface of a `List` in Appendix C).

8. (15) Fill the array below by showing in each line the state changes of a corresponding array when we sort it using heapsort. Start with the initial heap. Draw on the next page the initial heap and subsequent alterations. (*NB. the number of lines in the array below does not necessarily represent the exact number needed to sort the values of its initial state*).

	15	7	1	16	4	15	6	21	8
Initial heap =>									

Draw your heaps here :

Appendix A : functions `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
```

Appendix 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]
```

(and so on…)

Appendix 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
```

Draft 1

Draft 2

Draft 3