Name :____

Place number:______________________________

Permanent code:____________________________________

Directives:

- Write you name, place number and permanent code.
- Make visible your student identification card.
- Read all questions and **write your answers directly on the questionnaire**.
- Only use a pen or pencil. **No documentation, calculator, cell phone, computer, or other objects allowed**.
- This exam has 5 questions for 160 points in total.
- The scale was established to about 1 point per minute.
- This exam contains 22 pages, including 7 detachable sheets at the end for your draft.
- For developing questions, **write clearly and detail your answers**.
- You have 160 minutes to complete this exam.

GOOD LUCK AND HAVE A NICE SUMMER!

1	/ 20
2	/ 20
3	/ 50
4	/ 40
5	/ 30
Total	/ 160

1. (20). The following algorithm takes an array as input and returns the array with all duplicate elements removed. For example, if the input array is {1,3,3,2,4,2}, the algorithm returns {1,3,2,4}. What is its time performance if the set S is implemented with:

```
def setisize( E ):
    S = set()
    A = []
    for x in E:
        if not x in S:
            S.add( x )
            A.append( x )
    return A
```

a) (10) An AVL tree?

b) (10) A hashing table?

2. (20) Assume the following hash table, implemented with linear probing and the identity hash function, `h(x) = x`.

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

a) (10) In which order were the elements added to the table? WARNING: There are several good answers, and you have to give them all. Suppose that the size of the table has never been modified and no element has been deleted.

A	9, 14, 4, 18, 12, 3, 21
B	12, 3, 14, 18, 4, 9, 21
C	12, 14, 3, 9, 4, 18, 21
D	9, 12, 14, 3, 4, 21, 18
E	12, 9, 18, 3, 14, 21, 4

b) (10) Knowing that when removing an element from the table one can either release the entry that was used by that element or insert an availability value (e.g. `AVAIL`). Redraw the above table after removing 3 and explain your answer.

c) (10) If we want a hash table that stores a set of strings (String), a possible hash function is the length of the string, `h(x)=len(x)`. Is this a good hash function? Explain.

3. (50) Draw the resulting trees (eight in total) after inserting the keys {8, 16, 4, 2, 1, 64, 0, 32}, in that order, into an initially empty tree of type:

   a) (6) Heap

b) (6) Binary search tree

c) (6) AVL tree

d) (6) Splay tree

e) (8) 2-4 tree

f) (8) Red-Black tree

g) (10) B-tree of order 7 (a B-tree of order *d* is a (*a,b*) tree, where *a* = *d*/2 and *b* = *d*)

4. (40) A bidirectional `Map` is a `Map` that supports bidirectional search: from a key, *k*, you can find the corresponding value, *v*, and given a value, *v*, you can find the corresponding key, *k*. In a bidirectional `Map`, there is always a one-to-one relationship between keys and values. In other words, each key has exactly one value, and each value is found under exactly one key. The bidirectional `Map` supports the usual `Map` operations (see the table below). We can implement a bidirectional `Map` using two `Map`, each implemented by a red-black tree, for example.

   - *forward* is a `Map` of the keys to values. In the example below, after the three first operations it contains the entries {1:2 et 3:4}.
   - *back* is a `Map` of the values to the keys. In the example below, when *forward* contains the entries {1:2 et 3:4}, *back* contains the entries {4:3 et 2:1}.

   The invariant is that the two maps always contain the same data: *forward* contains the mapping k → v, if and only if *back* contains the mapping v → k. One can implement the "lookup", M [k], ie the search for key *k* in *forward* and "rlookup", the reverse search, as the search for the key *v* in *back*.
   Your task is to implement the remaining operations, ***insert*** and ***delete***, with execution time in O(log *n*). WARNING: the algorithm is more complicated than it seems (eg operation # 5 in the table below). Be sure to keep the invariant of the data structure. It is also a good idea to test your solution on the example. Give the pseudo code for each operation. You do not need to write Python code, but be precise - a competent programmer should be able to take your description and implement it easily.
   You can freely use the standard data structures and course algorithms in your solution, including inserting, searching, and deleting in a `Map`, without explaining how they are implemented.

	Opération	**Résultat**
1	m = BidirectionalMap()	{}
2	m[1] = 2	{1:2}
3	m[3] = 4	{1:2, 3:4}
4	m[1]	2
5	m[4] = 2	{3:4, 4:2} Notez que 1:2 est remplacé par 4:2.
6	m.rlookup(2)	4
7	del M[4]	{3:4}

```
def __setitem__( self, k, v ):
```

```
def __delitem__( self, k ):
```

5. (30) A labyrinth is ***correctly built*** if there is a path from the start to the end, all the labyrinth is accessible from the start, and there is no loop. A labyrinth is represented by a n × n boolean matrix. Consider the following two examples, where the start is at the top left and the end at the bottom right (in bold and underlined), and the 1 are connected by a path. The 1 in bold in the correct labyrinth is a path from the start to the end.

```
correct = [[1,1,1,1,0],          incorrect = [[1,1,0,0,0],
           [0,1,0,1,0],                       [0,1,0,0,1],
           [1,0,0,1,1],                       [1,0,0,1,1],
           [0,1,0,0,0],                       [0,0,0,0,0],
           [1,0,1,1,1]]                       [1,0,1,0,1]]
```

a) (20) How can we determine if a labyrinth is correctly built?

b) (10) What is the runtime of your algorithm?

Draft

Draft

Draft

Draft

Draft

Draft

Draft