Algorithmics
Final Exam #1 (P1)

Undergraduate 1^{st} year S1
EPITA

9 Jan. 2018 - 10 : 00

Instructions (read it) :

- You must answer on **the answer sheets provided.**
  - No other sheet will be picked up. Keep your rough drafts.
  - Answer within the provided space. **Answers outside will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- **Code:**
  - All code must be written in the language `Python` (no C, CAML, ALGO or anything else).
  - **Any `Python` code not indented will not be marked.**
  - All that you need (types, routines) is indicated in the **appendix** (last page)!
- Duration : 2h

Exercise 1 (Stack or queue? – *2 points*)

Values A, B, C, D, E and F are inserted, in this order, into an empty linear data structure. Indicate, for each output order given on the answer sheets, whether the structure in question may be: a stack, a queue (it can be both), or neither (neither a stack nor a queue).

Exercise 2 (Binary Search – *3 points*)

Here we use a version of the binary search algorithm that stops when bounds intersect or become equal.

1. Complete the decision tree learning of a binary search on a 16-element list. Each node represents a range of search (left and right bounds) and the medium rank.
2. (a) Let a list containing 32768 elements be sorted in increasing order. How many element comparisons will be done, in worst case, in case of a negative search (integer answer)?
   (b) Let k be the answer to the previous question. Which length, at most, can the list have in order to cause $k + 2$ comparisons in case of a negative search?

Exercise 3 (ALGO → Python – *3 points*)

Let the function `test`, that uses operations of abstract type *Iterative list*, be defined as follows:

```
function test(List L) : boolean
variables
    integer  i
    boolean  b
begin
    b ← true
    i ← 1
    while i < length(L) do
        if nth(L, i) > nth(L, i+1) then
            b ← false
        end if
        i ← i + 1
    end while
    return b
end
```

1. What does the function `test` do?
2. Write a Python version of the function `test` that is possibly more optimized than the ALGO version shown above.

Exercise 4 (Minimaxi – *3 points*)

Write a function that searches for the minimum and the maximum values in an integer list.
It returns the positions in the list of the searched values.

Application examples:

```
>>> posMiniMaxi([1, 8, -2, 9, 12, -5, 0, 25, 12])
(5, 7)
>>> posMinimax([8, 5, 8, 5, 8])
(1, 0)
>>> posMinimax([])
...
Exception: empty list
```

Exercise 5 (Merge sort (Tri fusion) – *2,5 + 5 + 2,5 points*)

1. Write the function `partition` that splits a list into two (new) lists of almost identical lengths: one half in each list.

   Application examples:

```
>>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])
>>> partition([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Write the function `merge` that merges two lists, sorted in increasing order, into one new sorted list.

   Application example:

```
>>> merge([1,5,8], [2,3,4,8])
[1, 2, 3, 4, 5, 8, 8]
```

3. To sort a list `L`, we proceed (recursively) as follows:

   ▷ A list of length < 2 is sorted.

   ▷ A list of length ≥ 2:
   - the list is split into two lists `L1` and `L2` of almost identical lengths;
   - the two lists `L1` and `L2` are sorted recursively;
   - finally, the two lists `L1` and `L2` are merged into one sorted list.

   Use the two previous functions (written or not) to write the function `mergesort` that sorts a list in increasing order (not "in place": the function builds and returns a new list.)

   Application example:

```
>>> mergesort([5,3,2,8,7,1,5,4,0,6,1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

Appendix: Authorised functions and methods

You can use the method `append` and the function `len` on lists:

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.
```

You can also use the function `range` and `raise` to raise exceptions. Reminder:

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end=' ')
5 6 7 8 9

>>> raise Exception("blabla")
...
Exception: blabla
```