IFT2015 : Structures de données A14

Name:_____

Permanent code: _____

Place number: _____

Directives pédagogiques :

- 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 4 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 16 pages, including 3 pages at the end for draft.
- For each question, **write legibly and detail your answers**.
- You have 100 minutes to complete this exam.

GOOD LUCK !

1	/ 30
2	/ 25
3	/ 20
4	/ 35
Total	/100

Q1. (30) Suppose we want to find the kth smaller element of a data collection, A, which is not sorted. For example, the 3rd smallest in the collection A = [18, 72, 88, 13] is 72. A trivial algorithm is to first sort the collection. Then, the kth element is in A[k-1], A sorted = [13, 18, 72, 88], then A[2] = 72.

a) (5) What is the complexity of the trivial algorithm on average?

b) (15) Propose an algorithm in O(n) on average inspired by the median sort. The Python code of the median sort is in **Appendix A**.

c) (5) What is the complexity in worst case of the algorithm you proposed in (b)?

d) (5) Is there an algorithm in O(n) in worst case (which we saw in the course)? Which one?

Q2. (25) Consider the ADT Queue (see **Appendice B**).

a) (15) Give an implementation of the operations enqueue, dequeue and first using two stacks as instance variables and so that each operation executes inO(1) in amortized time.

```
class TwoStackQueue:

    def enqueue( self, element ):

    def dequeue( self ):

    def first( self ):
```

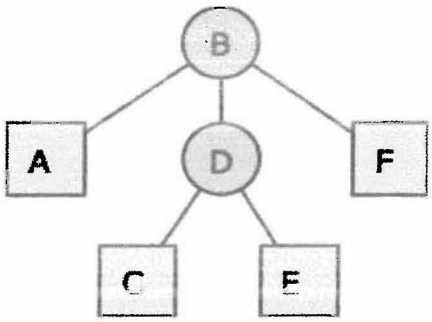
b) (10) Give a formal proof that each operation is in O(1) in amortized time.

Q3. (20) Fill the following table indicating in each line the progress made by the heapsort algorithm on the initial collection shown in the first row. Use the next page to show the initial and successive heaps as the algorithm progresses. (*NB. the number of rows in the table does not necessarily match the necessary number of collection states.*)

12	3	2	17	14	11	5	16	8

Your heaps here:

Q4. (35) Consider the following general tree:



a) (10) Define the class Node to store the information in each node of a general tree, knowing that we want access to the children and parent of a node in O(1).

```
class Node:

    Node parent;

    List<Node> children
```

b) (10) Draw the internal structure of this tree using your class **Node**.

?

What it means by internal structure?

c) (5) In which order the nodes will be visited in a preorder traversal?

preorder-traversal?

d) (5) In which order the nodes will be visited in a postorder traversal?

post order-traversal?

e) (5) In which order the nodes will be visited in a breadth-first traversal?.

Appendix A : Median sort

```
import random

def swap( A, i, j ):
    tmp = A[i]
    A[i] = A[j]
    A[j] = tmp

def partition( A, g, d, iPivot ):
    pivot = A[iPivot]
    swap( A, iPivot, d )

    iPivot = g
    for i in range( g, d ):
        if A[i] <= pivot:
            swap( A, iPivot, i )
            iPivot += 1

    swap( A, iPivot, d )
    return iPivot

def select( A, k, g, d ):
    i = random.randint( g, d )
    iPivot = partition( A, g, d, i )

    if ( g + k - 1 ) == iPivot:
        return iPivot
    if ( g + k - 1) < iPivot:
        return select( A, k, g, iPivot-1 )
    else:
        return select( A, k - ( iPivot-g+1 ), iPivot + 1, d )

def triMediane( A, g, d ):
    if d <= g:
        return

    milieu = (d - g + 1 ) // 2
    mediane = select( A, milieu, g, d )

    triMediane( A, g, mediane - 1 )
    triMediane( A, mediane + 1, d )
```

Appendix B : ADT Queue

```
class Queue:

    def __init__( self ):
        pass

    def __len__( self ):
        pass

    def __str__( self ):
        pass

    def is_empty( self ):
        pass

    #add an element to the Queue
    def enqueue( self, element ):
        pass

    #remove an element to the Queue
    def dequeue( self ):
        pass

    #return the first element in the Queue
    #sans le retirer
    def first( self ):
        pass
```