Data Structures and Algorithms

Computer Science Degree

Final Exam. September 6^{th}, 2018

1. (3 points) We say an array is a *quasi Fibonacci sequence* if all the elements of the array meet the following property: $v[i] = v[i-1] + v[i-2]$ or $v[i] = v[i-1]$.

Specify, design an code an iterative algorithm that receives an array v of size n ($2 \leq n \leq 1000$) and determines whether it is a *quasi Fibonacci sequence* and the sum of all its elements is not greater than a certain value $k \geq 0$.

Input							Output
n	k	v					
5	5	1	1	1	1	1	SI
5	4	1	1	1	1	1	NO
5	5	1	1	1	2	3	NO
5	5	0	0	0	0	0	SI
5	5	0	0	0	0	1	NO
0							

2. (2 points) Code a function that receives 3 integer numbers, *ini*, n ($2 \leq n \leq 100$) and k ($k \geq 0$) and generates all the arrays of size n that form a *quasi Fibonacci sequence* where $v[0] = ini$ and the sum of its elements is not greater than k (i.e. all the arrays with $v[0] = ini$ for which the function of exercise 1 would return `SI` using the same values of n and k).

Clue: You don't need to use the previous function to solve this exercise

Input			Output
ini	n	k	
1	5	4	
1	5	5	1 1 1 1 1
1	5	6	1 1 1 1 1
			1 1 1 1 2
1	5	7	1 1 1 1 1
			1 1 1 1 2
			1 1 1 2 2
0	5	10	0 0 0 0 0
0	0	0	

3. (2 points) We say a node in a binary tree of integers is *curious* when its value is the result of adding its level and the number of nodes in its left subtree. Code a function

```
int numCurious(const Arbin<int>& a)
```

that returns the number of *curious* nodes there are in the binary tree `a`. Explain carefully the complexity of your solution.

4. **(3 points)** We have to build a system to manage the waiting lists in an online concert ticket selling system.

   When clients sign up in the system, they are assigned a unique client ID code (of type string). The system allows the administrators to add concerts, identified by a unique concert ID (of type string), and to manage clients waiting lists, who intend to buy tickets for the concerts available in the system.

   In order to code this system, we need a `SellingSystem` ADT with the following operations:

   - `create()`: constructor that creates a new, empty selling system.
   - `addClient(clientID)`: adds a new client to the system, with code `clientID`. If a client with that code already exists, the operations throws aa `ExistingClient` exception.
   - `addConcert(concertID)`: adds a new concert to the system, with code `concertID`. If a concert with that code already exists, the operation throws an `ExistingConcert` exception.
   - `addToWaitingList(clientID,concertID)`: adds the client with code `clientID` to a waiting list for the concert with code `concertID`. The system only allows for a client to be waiting in a single waiting list. The operation throws a `WaitingNotAllowed` exception if the `clientID` or `concertID` do not exist or if the client is already in a waiting list.
   - `clientsWaiting(concertID)`→`boolean`: returns *true* if there are clients waiting to buy tickets for the concert with code `concertID`, and *false* otherwise. The operation throws an `InexistentConcert` exception if the `concertID` is not in the system.
   - `nextClient(concertID)`→`clientID`: returns the `clientID` of the first client in the waiting list for the concert with code `concertID`. The operation throws an `InexistentConcert` exception if the `concertID` is not in the system. If the concert exists but there are not clients waiting to buy tickets, the operation throws an `EmptyWaitingList` exception.
   - `sell(concertID)`: sells a ticket for the concert with code `concertID`. As a result, the operation removes the first client in the concert waiting list, so that the client can now be added to another waiting list. The operation throws an `InexistentConcert` exception if the `concertID` is not in the system. If the concert exists but there are not clients waiting to buy tickets, the operation throws an `EmptyWaitingList` exception.

   Provided that this is a critical system, the operations must be as efficient as possible. Therefore, you have to choose an appropriate representation for the ADT, code the operations and justify their complexity.