Exercise 4
The following module provides queues of natural numbers (solution of Exercise 3):

```plaintext
module Queue;
   import Bool, true, false from Bool;
   Nat, zero from Nat;
export all;
sort Queue;
operations
   newqueue : -> Queue;
   enqueue : Queue x Nat -> Queue;
   isempty : Queue -> Bool;
   dequeue : Queue -> Queue;
   head : Queue -> Nat;
   tail : Queue -> Nat;
declare q : Queue; n,m : Nat;
axioms
   isempty(newqueue) == true;
   isempty(enqueue(q,n)) == false;
   dequeue(newqueue) == newqueue;
   dequeue(enqueue(newqueue,n)) == newqueue;
   dequeue(enqueue(enqueue(q,m),n)) ==
      enqueue(dequeue(enqueue(q,m)),n);
   tail(newqueue) == zero;
   tail(enqueue(q,n)) == n;
   head(newqueue) == zero;
   head(enqueue(newqueue,n)) == n;
   head(enqueue(enqueue(q,m),n)) == head(enqueue(q,m));
end module Queue;
```

a) Decompose the set of operations into constructors and non-constructor operations.
b) For the axioms, check all constructivity constraints.
c) For the queue \( q = 1\ 2\ 3\ 4\ 5 \), illustrate its representation as a tree for the respective term.
d) For \( q \), elaborate the execution of \( \text{head}(q) \) by application of term rewrite rules.
e) Based on c) and d), discuss the efficiency of this prototypical implementation.

Exercise 5

a) Using a module \texttt{ArrayNatNat}, provide an abstract implementation of the module \texttt{Queue} shown above. (The module \texttt{ArrayNatNat} works analogously to \texttt{ArrayNat}, but support two natural numbers – used for the lower and upper bound of the array – instead of one.)
b) Discuss the efficiency of this implementation, assuming a built-in efficient implementation of arrays and natural numbers.