```
// Example of implementing a binary search algorithm.
//
// Copyright 2001 Escher Technologies Ltd. All rights reserved.

final class Table of X
require X has total operator ~~(arg) end
^=
abstract
        var members: seq of X;

        // Invariant states that the members are in a non-decreasing order
        invariant members.isndec;

interface
        // Find the index of the first element that ranks with or below the
        parameter
        function search(x: X): nat
                satisfy result <= #members,
                        forall z::0..<result :- x > members[z],
                        forall z::result..<#members :- ~(x > members[z])
                via
                        // Implement with a binary search
                        var i: nat != 0;
                        loop
                                var k: int != #members;
                                change i
                                keep
                                        0 <= i',
                                        i' <= k',
                                        k' <= #members,
                                        forall z::0..<i' :- x > members[z],
                                        forall z::k'..<#members :- ~(x >
                                        members[z])
                                until i' = k'
                                decrease k' - i';
                                // Start of loop body...
                                // Set up the partition value
                                let p ^= (i + k)/2;
                                assert i <= p < k;
                                if
                                [x > members[p]]:
                                        i! = >p;
                                        // If x ranks above members[p] then
                                        // move i to >p

                                []:
                                        k! = p;
                                        // If x ranks with or below members[p]
                                        // then move k to p

                                fi
                        end;
                        value i;
                end;

        // Count operator
        operator #: nat
                ^= #members;

        // Index operator to access the members of the list
        operator [](index: nat): X
                pre index < #self
                ^= members[index];

        // Build from a set
        build{mem: set of X}
                post members! = mem.permndec;

end;
```