```fsharp
//week3 sorting
module SortAndPBT

open FsCheck

let rec merge lst1 lst2 =
    match (lst1, lst2) with
    | (l1head :: _, l2head :: l2tails) when l1head >= l2head -> l2head :: merge lst1 l2tails
    | (l1head :: l1tails, l2head :: _) when l1head < l2head -> l1head :: merge l1tails lst2
    | ([], []) -> []
    | ([], _) -> lst2
    | (_, []) -> lst1


let split array =
    let rec split_help array (a1, a2) =
        match array with
        | [] -> (List.rev a1, List.rev a2)
        | array_head :: [] -> (List.rev (array_head :: a1), List.rev a2)
        | index1 :: index2 :: array_tail -> split_help array_tail (index1 :: a1, index2 :: a2)

    split_help array ([], [])


let sort array_ori =
    let rec sort_help array =
        let (array1, array2) = split array

        match array with
        | head :: [] -> [ head ]
        | [] -> []
        | _ :: _ -> merge (sort_help array1) (sort_help array2)

    sort_help array_ori

let rec ordered xs =
    match xs with
    | head1 :: head2 :: tails ->
        if head1 <= head2 then
            true && ordered (head2 :: tails)
        else
            false
    | head1 :: [] -> true
    | [] -> true

(*printfn "%A" (merge [1;4;9;12] [2;3;4;5;10;13])
```

```
47  printfn "%A" (split [1; 2; 3; 4; 4; 5; 9; 10; 12; 13])*)
48  (*printfn "%A" (sort [2;6;7;8;3;5;1;2;4;23;6;4;55])*)
49
50  printfn
51      "%b"
52      (ordered (
53          sort [ 2
54                 6
55                 7
56                 8
57                 3
58                 5
59                 1
60                 2
61                 4
62                 23
63                 6
64                 4
65                 55 ]
66      ))
67
68
69
70  let orderedSort (xs: int list) = ordered (sort xs)
71
72  let increment (x, cnt) =
73      let rec inc (x, cnt) =
74          match cnt with
75          | (key, value) :: tails ->
76              if key = x then
77                  value + 1
78              else
79                  inc (x, tails)
80          | [] -> 1
81
82      inc (x, cnt)
83
84  let toCounting xs =
85      let rec count xs temp =
86          match xs with
87          | head :: tails when (increment (head, temp) = 1) -> count tails
              ((head, 1) :: temp)
88          | head :: tails ->
89              count
90                  tails
91                  (List.map
92                      (fun (key, value) ->
93                          if key = head then
94                              (key, value + 1)
```

```
 95                             else
 96                                 (key, value))
 97                         temp)
 98             | [] -> temp
 99
100         count xs []
101
102 let p = [ 1; 2; 1; 1; 1; 6 ]
103 toCounting p
104
105
```