```
theory Problem-1
  imports Main
begin
```

Jacek has n cards, labelled with consecutive numbers $1, \ldots, n$. He places them in a row on the table, in any order he chooses. He will be taking the cards off the table in the ordering given by the cards' labels he will first take card number 1, then number 2, and so on. Before Jacek begins taking the cards off the table, Placek will color each card red, blue or yellow. Prove that Placek may color the cards in such a manner, that while Jacek is taking them off the table, at all times the following property is maintained: between any two cards of the same color, there is at least one card of a different color.

```
datatype color = Red | Blue | Yellow

definition property :: color list ⇒ bool where
  property clrs ⟷ (∀ a < length clrs. ∀ b < length clrs.
    a < b ∧ clrs ! a = clrs ! b ⟶ (∃ m ∈ {a..b}. clrs ! m ≠ clrs ! a))

lemma successively-conv-nth: successively R xs ⟷ (∀ i<length xs − 1. R (xs ! i) (xs ! Suc i))
  by (induction xs rule: induct-list012; force simp: nth-Cons split: nat.splits)

lemma propertyD: property xs ⟹ i < j ⟹ j < length xs ⟹ xs ! i = xs ! j ⟹ ∃ m ∈ {i..j}.
xs ! m ≠ xs ! i
  unfolding property-def by auto


lemma property-altdef[simp]: property xs ⟷ distinct-adj xs
proof (intro iffI)
  assume p: property xs
  show distinct-adj xs
    unfolding distinct-adj-def successively-conv-nth
  proof (intro allI impI)
    fix i assume i: i < length xs − 1
    show xs ! i ≠ xs ! Suc i
    proof
      assume ∗: xs ! i = xs ! Suc i
      with p i have ∃ m∈{i..Suc i}. xs ! m ≠ xs ! i
        by (intro propertyD) auto
      thus False using ∗ by (auto simp: le-Suc-eq)
    qed
  qed
next
  assume distinct-adj xs
  show property xs unfolding property-def
  proof safe
    fix i j assume ij: i < j j < length xs xs ! i = xs ! j
    from ‹distinct-adj xs› have xs ! i ≠ xs ! Suc i
      using ij by (auto simp: successively-conv-nth distinct-adj-def)
    thus ∃ m∈{i..j}. xs ! m ≠ xs ! i
      using ij by (intro bexI[of - Suc i]) auto
  qed
qed

lemma property-insert:
  assumes property (l @ r)
  obtains clr where property (l @ clr # r)
proof −
  obtain clr where clr: clr ≠ last l  clr ≠ hd r
    by (metis color.distinct(1,3,5))
  let ?list = l @ [clr] @ r
  have distinct-adj l  distinct-adj r
    using assms by auto
  hence property ?list
```

    unfolding *property-altdef distinct-adj-append-iff*
    using *clr* **by** *auto*
  thus *?thesis* **using** *that* **by** *simp*
qed

definition *remove-smallest* :: *nat list* ⇒ *nat list* **where**
  remove-smallest xs = *remove1* (*Min* (*set xs*)) *xs*

lemma *remove1-split*:
  assumes $a \in$ *set xs*
  shows ∃ *l r*. *xs* = *l* @ *a* # *r* ∧ *remove1 a xs* = *l* @ *r*
using *assms* **proof** (*induction xs*)
  case (*Cons x xs*)
  show *?case*
  proof *cases*
    assume $x = a$
    show *?thesis*
      apply (*rule exI*[*of* - []])
      using ‹$x = a$› **by** *simp*
  next
    assume $x \neq a$
    then have $a \in$ *set xs*
      using ‹$a \in$ *set* (*x* # *xs*)›
      by *simp*
    then obtain *l r* **where** ∗: *xs* = *l* @ *a* # *r* ∧ *remove1 a xs* = *l* @ *r*
      using *Cons.IH* **by** *auto*
    show *?thesis*
      apply (*rule exI*[*of* - *x* # *l*])
      apply (*rule exI*[*of* - *r*])
      using ‹$x \neq a$› ∗ **by** *auto*
  qed
qed *simp*

lemma *remove-smallest-distinct*:
  distinct xs ⟹ *distinct* (*remove-smallest xs*)
  unfolding *remove-smallest-def* **by** *simp*

lemma *remove-smallest-subset*:
  set (*remove-smallest xs*) ⊆ *set xs*
  unfolding *remove-smallest-def* **by** (*rule set-remove1-subset*)

lemma *remove-smallest-length*[*simp*]: *xs* ≠ [] ⟹ *length* (*remove-smallest xs*) < *length xs*
  by (*simp add*: *remove-smallest-def length-remove1*)

lemma *remove-smallest-map*:
  assumes *map f xs* = *map g xs*
  shows *map f* (*remove-smallest xs*) = *map g* (*remove-smallest xs*)
proof *cases*
  assume *xs* = []
  then show *?thesis* **by** (*simp add*: *remove-smallest-def*)
next
  assume *xs* ≠ []
  then have *Min* (*set xs*) ∈ *set xs*
    by *auto*
  then obtain *l r* **where** ∗: *xs* = *l* @ *Min* (*set xs*) # *r* **and** ∗∗: *remove-smallest xs* = *l* @ *r*
    unfolding *remove-smallest-def*
    using *remove1-split* **by** *fast*
  from *assms* **have** ∀ *x* ∈ *set xs*. *f x* = *g x* **by** *auto*
  hence (∀ *x* ∈ *set l*. *f x* = *g x*) ∧ (∀ *x* ∈ *set r*. *f x* = *g x*)
    apply (*subst* (*asm*) ∗)
    by *auto*
  then show *?thesis*
    unfolding ∗∗

```
    by simp
qed

lemma removal-induction:
  assumes P []
  assumes ⋀xs. xs ≠ [] ⟹ P (remove-smallest xs) ⟹ P xs
  shows P ys
  using assms
  apply induction-schema
    apply auto[1]
  by lexicographic-order

theorem problem1:
  fixes order :: nat list
  assumes distinct order
  shows ∃ colors :: nat ⇒ color. ∀ n. property (map colors ((remove-smallest^^n) order))
    (is ∃ colors. ∀ n. ?P colors n order)
using assms proof (induction rule: removal-induction)
  case 1
  have remove-smallest [] = []
    by (simp add: remove-smallest-def)
  hence (remove-smallest^^n) [] = [] for n
    by (induction n) auto
  then show ?case by simp
next
  case (2 order)
  then have distinct: distinct (remove-smallest order)
    using remove-smallest-distinct by auto
  note 2.IH[OF distinct]
  then obtain colors where IH′: ?P colors n (remove-smallest order) for n
    by auto
  let ?m = Min (set order)
  obtain xs ys where xsmys: order = xs @ ?m # ys and xsys: remove-smallest order = xs @
ys
    unfolding remove-smallest-def
    using ‹order ≠ []› by atomize-elim (auto intro: remove1-split)
  let ?l = map colors xs and ?r = map colors ys
  have property (?l @ ?r)
    using xsys IH′[of 0] by simp
  then obtain clr where property-clr: property (?l @ clr # ?r)
    by (auto intro: property-insert)
  let ?colors′ = colors(?m := clr)
  have ?m ∉ set xs  ?m ∉ set ys
    by (metis ‹distinct order› xsmys distinct-append not-distinct-conv-prefix)
       (metis ‹distinct order› xsmys distinct.simps(2) distinct-append)
  hence property0: property (map ?colors′ order)
    using property-clr by (subst (2) xsmys) simp
  have ?m ∉ set (remove-smallest order)
    unfolding remove-smallest-def
    using ‹distinct order› by simp
  hence ?m ∉ set ((remove-smallest^^n) (remove-smallest order)) (is ?m ∉ set (?xs n))
    for n
    by (induction n) (use remove-smallest-subset in auto)
  hence map-colors: map ?colors′ (?xs n) = map colors (?xs n) for n
    by simp
  show ?case
    apply (intro exI[of - ?colors′] allI)
    subgoal for n
      apply (cases n)
       apply (simp only: funpow-0, rule property0)
      by (auto simp only: funpow-Suc-right o-apply map-fun-upd map-colors IH′)
    done
qed
```

end