Problem (1):

We want to prove : power n x = x^n

The principle inductive natural number is □n, P(n) if P(0) and P(n') → P(n'+1)

The inductive proof is:

- ✓ The base case is P(0) is Power 0 x = 1 (by the definition of Power)
- ✓ The inductive hypothesis is power n' x = $x^{n'}$
- ✓ The inductive case is power (n'+1) x = $x^{(n'+1)}$

power (n'+1) x	
= x * power ((n'+1)-1) x	(by the definition of Power)
= x^1 * power n' x	(by the properties of addition and subtraction)
= (x) * $x^{n'}$	(by inductive hypothesis)
= $x^{n'+1}$	(by the properties of power)

--

Problem (2):

We want to prove : power (n) x = $x^{toInt(n)}$

The principle inductive natural number is □n, P(n) if P(0) and P(n') → P(Succ n')

- ✓ The base case is P(0) is Power 0 x= $x^{toInt(0)}$ = 1

Power 0 x	
= 1	(by the definition of Power)
= $x^{toInt(0)} = x^0$ = 1	(by the definition of toInt)

- ✓ Given : □n ϵ nat. power (n') x = $x^{toInt(n')}$
- ✓ The inductive case is :

power (Succ n') x = $x^{toInt(Succ\ n')}$	
= x * power (n') x	(by the definition of Power)
= x * $x^{toInt(n')}$	(by inductive hypothesis)
= $x^{toInt(n')+1}$	(by the properties of power)

= $x^{\text{toInt(Succ n')}}$ (by the definition of toInt)

problem (3):

We want to prove : ∀ l,r ϵ ‘a list. length (l @ r) = length l + length r

The principle inductive is ∀ l, P(l) if P ([ ]) and P(l’) =⇒ P(v :: l’))

The inductive proof is: The base case is P ([]) is length ([] @ r) = length r

∀ r. r ϵ a’ list (

length ([] @ r)

= length r (by the properties of list which is [] @ r = r)

= 0 + length r (identity of addition natural number)

= length ([]) + length r (by the definition of length)

✓ The inductive hypothesis is

∀ l2 ϵ ‘a list. length (l1 @ l2) = length l1 + length l2

✓ The inductive case is length (x :: xs @ r) = length (x :: xs) + length (r)

length (x :: (xs @ r)) (by the properties of list)

= 1 + length (xs @ r) (by the definition of length)

= 1 + length (xs) + length (r) (by inductive hypothesis)

= length (x :: xs) + length (r) (by the definition of length)

--

Problem (4):

We want to prove : ∀ r, r ϵ ‘a list. length (reverse r) = length r

The principle inductive is ∀ l, P(l) if P ([ ]) and P(l’) =⇒ P(v :: l’))

The inductive proof is:

✓ The base case is P ([]) is length (reverse ([]) = 0

∀ r. r ϵ a’ list (

length (reverse[])

= length ([]) (by the definition of reverse)

= 0 (by the definition of length)

✓ The inductive hypothesis is

∀ l. l ϵ a' list. length (reverse l) = length l

✓ The inductive case is : length (reverse (x :: xs)) = length (x :: xs)

length (reverse (x :: xs))

= length (reverse (xs @ [x])) (by the definition of reverse)

= length (reverse (l)) (assume l = xs @ [x])

= length (l) (by inductive hypothesis)

= length (xs @ [x]) (return the value of assumption)

= length ((x :: xs)) (by the properties of list)

--

Problem (5):

We want to prove :

∀ l1 and l2, l1 and l2 ϵ 'a list. reverse (append l1 l2) = append (reverse l2) (reverse l1)

The principle inductive is ∀ l, P(l) if P ([ ]) and P(l') =⇒ P(v :: l'))

The inductive proof is:

✓ The base case P ([]) is reverse (append [] l2) = reverse l2

reverse (append [] l2)

= reverse (l2) (by the definition of append)

✓ The inductive hypothesis is

∀ l1 and l2, l1 and l2 ϵ 'a list. reverse (append l1 l2) = append (reverse l2) (reverse l1)

✓ The inductive case is reverse (append (x :: xs) l2) = append (reverse l2) (reverse (x::xs))

reverse (append (x :: xs) l2)

= reverse (x :: append xs l2) (by the definition of append)

= reverse (append xs l2) @ [x] (by the definition of reverse)

= append (reverse l2) (reverse xs) @ [x] (by inductive hypothesis)

= append (reverse l2) (reverse (xs @ [x])) (by the properties of list)

= append (reverse l2) (reverse (x :: xs)) (by the definition of reverse)

--

Problem (6):

We want to prove : ∀ l, l ϵ 'a list. sorted l => sorted (place e l)

The principle inductive is ∀ l, P(l) if P ([ ]) and P(v::l') =⇒ P(v1 ::v2:: l'))

The inductive proof is:

- ✓ The base case P ([]) is

  sorted ([]) => sorted (place e [])

  the right side:

  sorted ([]) (by the definition of sorted)

  = true

  the left side:

  sorted (place e [])

  = sorted([e]) (by the definition of place)

  = true (by the definition of sorted)

- ✓ The inductive hypothesis is

  ∀ (v::xs), (v::xs) ϵ 'a list. sorted ((v::xs)) => sorted (place e (v::xs))

- ✓ The inductive case is sorted ((v1::v2::xs)) => sorted (place e (v1::v2::xs))

  the left side is sorted (place e (v1::v2::xs))

  <u>if e < v1</u>

  sorted (e::v1::v2::xs) (by the definition of place)

  sorted (e::(v1::v2::xs)) (by the properties of list)

  sorted (v1::(v2::xs)) => sorted (e::(v1::v2::xs)) is true (by inductive hypothesis)

  sorted (v1::v2::xs) => sorted (e::v1::v2::xs)

  <u>otherwise when e >= x</u>

  sorted (v1 :: place e (v2::xs)) (by the definition of place)

  sorted (v1 :: (place e (v2::xs))) (by the properties of list)

  sorted (v1::(v2::xs)) => sorted (v1 :: (place e (v2::xs))) is true (by inductive hypothesis)

  sorted (v1::v2::xs) => sorted (v1 :: place e v2::xs)

--

Problem (7):

First of all the function place will add the element e in the right place. So when we start searching about this element, of course we will find that (e = x) in one of the recursive call of (is_elem e l). And as we know : True ||(any other boolean) is true =⇒ in the is_elem function we will see : e = x || . . . is true.

No, it is not because when we add the element e in the list l before the first element which is bigger than e. So when is_elem start search about the e, it will find it by returning true of e = x condition and this condition of course will happen because we search about it after we add it immediately.