A inserção é feita de forma que se l está ordenada então a lista resultante também está ordenada. Podemos enunciar esta propriedade como um lema:

Lema 2.1. *Se l é uma lista ordenada e x é um inteiro, então $(insert\ x\ l)$ é uma lista ordenada.*

Exercício 2.2. *Prove o Lema 2.1.*

Na aula de *[2024-03-18 seg]*, iniciamos a prova do Lema 2.1 utilizando indução na estrutura da lista l. A base da indução corresponde ao caso em que a lista l é vazia. Neste caso, temos que provar que $insert\ x\ nil$ é uma lista ordenada, assumindo que a lista vazia está ordenada. Mas isto é trivial porque, pela definição de *insert*, $insert\ x\ nil$ é a lista unitária $x :: nil$ que está ordenada por definição.
O passo indutivo ocorre quando l tem a forma $h :: tl$. Neste caso, temos que provar que $insert\ x\ (h :: tl)$ é uma lista ordenada, assumindo que a lista $h :: tl$ está ordenada. A definição de *insert* nos dá dois subcasos:

1. $x \leq h$: Neste caso, $insert\ x(h :: tl)$ retorna $x :: h :: tl$ que está ordenada porque o elemento inserido x é menor ou igual ao primeiro elemento de uma lista ordenada.
2. $x > h$: Na aula de *[2024-03-20 qua]*, iniciamos com a escrita da hipótese de indução:

   > Se tl é uma lista ordenada e x é um inteiro, então $(insert\ x\ tl)$ é uma lista ordenada.

Temos como hipótese (do problema) que $h :: tl$ é uma lista ordenada, e queremos mostrar que a lista $insert\ x\ (h :: tl)$ é uma lista ordenada. Pela definição da função *insert*, temos que $insert\ x\ (h :: tl) = h :: (insert\ x\ tl)$. Agora observe que, por hipótese de indução, a lista $insert\ x\ tl$ está ordenada. Além disto, temos que $h < x$ e h é menor ou igual a todo elemento da lista tl já que $h :: tl$ está ordenada. Então h é menor ou igual a todo elemento de $(insert\ x\ tl)$, e portanto a lista $h :: (insert\ x\ tl)$ está ordenada. □

Agora vamos refazer esta prova mecanicamente utilizando o assistente de provas Coq. Primeiramente, vamos definir o predicado que caracteriza uma lista ordenada.

```
Inductive ordenada : list nat -> Prop :=
  | nil_ord : ordenada nil
  | one_ord : forall n:nat, ordenada (n::nil)
  | cons_ord : forall (n m:nat) (l:list nat), n <= m ->
               ordenada (m::l) -> ordenada (n::m::l).
```

A função *insert* pode ser definida em Coq como segue:

```
Fixpoint insert (x:nat) (l:list nat) : list nat :=
  match l with
  | nil => x::nil
  | h::tl => if x <=? h then x::l else h::(insert x tl)
  end.
```

Agora podemos enunciar e provar o lema 2.1 em Coq:

```
Lemma insert_ord : forall (x:nat) (l:list nat), ordenada l -> ordenada (insert x l).
```

Exercício 2.3. *Prove o lema* `insert_ord` *no Coq.*

A resolução do exercício anterior nos dá uma boa ideia do que consiste o processo de formalização (ou mecanização) de um algoritmo, ainda que bastante simples. A função principal do algoritmo de ordenação por inserção é dada a seguir:

$$insertion\_sort\ l := \begin{cases} l, & \text{se } l = nil \\ insert\ x\ (insertion\_sort\ tl), & \text{se } l = h :: tl \end{cases}$$