# A Correção do Algoritmo de Ordenação por Inserção

Alan dos Santos Dias - 232007830
Bruno Henrique Duarte - 221022239
João Marcos Rodrigo Cardoso - 232027411

5 de dezembro de 2025

## 1 Introdução

Este trabalho apresenta uma prova formal da correção do algoritmo de ordenação por inserção . A formalização foi desenvolvida no assistente de provas Coq. O assistente de provas Coq utiliza o sistema de Dedução Natural, o que o torna adequado para o desenvolvimento de atividades computacionais na disciplina de Lógica Computacional 1. Conforme discutido em aula, provas matemáticas realizadas apenas em papel estão suscetíveis a erros humanos, ambiguidades e saltos lógicos. O uso de uma ferramenta formal como o Coq é fundamental para mitigar esses riscos, garantindo o rigor e a correção mecânica de cada passo da demonstração.

Observação sobre o ambiente de prova: A formalização deste projeto foi realizada utilizando a plataforma online jsCoq (disponível em $https://jscoq.github.io/scratchpad.html$). As versões específicas utilizadas foram o jsCoq 0.12.3, executando sobre o núcleo do Coq versão 8.12.2 (build 81200).

## 2 Definição dos Algoritmos

### 2.1 Função Auxiliar de Inserção

A função auxiliar $insert$ recebe um número natural $x$ e uma lista $l$ (assumida como ordenada). O algoritmo é definido por análise da estrutura da lista:

- Lista Vazia: Retorna a lista unitária $[x]$.
- Lista Não-Vazia ($h :: tl$): Compara-se $x$ com a cabeça $h$:
  - Se $x \leq h$: $x$ é inserido na posição atual, tornando-se a nova cabeça.
  - Se $x > h$: Mantém-se $h$ na cabeça e insere-se $x$ recursivamente na cauda $tl$.

### 2.2 Algoritmo de Ordenação

A função principal $insertion\_sort$ percorre a lista de entrada recursivamente para construir a lista ordenada final:

- Caso Base: Para uma lista vazia, retorna-se uma lista vazia.
- Passo Recursivo: Para uma lista composta por cabeça $h$ e cauda $tl$, o algoritmo primeiro ordena recursivamente a cauda $tl$ e, em seguida, utiliza a função $insert$ para posicionar o elemento $h$ no local correto dentro da cauda já ordenada.

# 3 Propriedades Auxiliares

Para provar a correção total do algoritmo, precisamos estabelecer duas propriedades fundamentais sobre a função de inserção:

- 1. Ela preserva a ordenação dos elementos.
- 2. Ela preserva o conjunto de elementos (é uma permutação).

## 3.1 Preservação da Ordenação

O lema a seguir garante que a operação de inserção mantém a integridade da ordem.

Lema: Para todo elemento $x$ e lista $l$, se $l$ já está ordenada ($Sorted$), então a lista resultante de $insert\ x\ l$ também estará ordenada.

Demonstração:

A prova é realizada por indução na evidência de que a lista $l$ já está ordenada ($H : Sorted\ le\ l$).

- Caso Base (Lista Vazia): A inserção de $x$ em uma lista vazia resulta na lista unitária $[x]$, que é trivialmente ordenada.
- Passo Indutivo: Supondo uma lista da forma $a :: tl$ onde a cauda $tl$ também é ordenada, temos dois cenários baseados na comparação entre $x$ e a cabeça $a$:
  - Se $x \leq a$: O elemento $x$ torna-se a nova cabeça da lista ($x :: a :: tl$). Como $x \leq a$ e o restante da lista já estava ordenado, a propriedade é preservada.
  - Se $x > a$: O algoritmo insere $x$ recursivamente na cauda $tl$. Pela hipótese de indução, sabemos que essa inserção recursiva gera uma lista ordenada. Resta apenas provar que a cabeça original $a$ preserva a ordem em relação à nova lista gerada, o que é garantido pois $a < x$ e $a$ já era menor ou igual a todos os elementos de $tl$.

## 3.2 Preservação da Permutação

Além da ordenação, é fundamental garantir que a operação de inserção não altere o conjunto de dados.

Lema: Para todo elemento $x$ e lista $l$, a lista resultante da inserção $insert\ x\ l$ é uma permutação da lista original acrescida de $x$ (ou seja, $x :: l$).

Demonstração:

A demonstração é conduzida por indução estrutural na lista $l$:

- Caso Base: Para uma lista vazia, a inserção retorna a lista unitária $[x]$. Como a lista original adicionada de $x$ também é $[x]$, a permutação é trivial (reflexiva).
- Passo Indutivo: Considerando uma lista $a :: l'$, comparamos $x$ com a cabeça $a$:
  - Se $x \leq a$: O elemento é inserido na cabeça. A lista resultante é idêntica à lista de entrada com $x$ prefixado.
  - Se $x > a$: O elemento deve ser inserido recursivamente na cauda. A prova exige um passo crucial de transitividade: 1. Primeiro, estabelecemos que $x :: a :: l'$ é uma permutação de $a :: x :: l'$ (troca de posições ou "swap"). 2. Em seguida, focamos na cauda (ignorando a cabeça $a$ que é comum a ambos) e aplicamos a Hipótese de Indução, que garante que a inserção de $x$ em $l'$ mantém a propriedade de permutação.

# 4 Teorema Principal

A correção total de um algoritmo de ordenação é estabelecida verificando-se duas propriedades fundamentais na lista de saída:

- Ordenação: Os elementos devem estar dispostos em ordem não decrescente (propriedade verificada pelo predicado *Sorted*).
- Permutação: A lista resultante deve conter exatamente os mesmos elementos da lista de entrada, preservando suas multiplicidades (propriedade verificada pelo predicado *Permutation*).

Formalizamos essa especificação no teorema a seguir:
Teorema: Para qualquer lista $l$, a lista gerada por *insertion_sort l* é ordenada e é uma permutação de $l$.

## 4.1 Demonstração

A prova da correção total combina as duas propriedades verificadas nos lemas auxiliares (preservação da ordenação e da permutação). A demonstração procede por indução estrutural na lista de entrada $l$:

- Caso Base: A lista vazia é trivialmente ordenada e é uma permutação de si mesma.
- Passo Indutivo: Assumindo que a chamada recursiva para a cauda da lista já produz um resultado correto (Hipótese de Indução), dividimos o objetivo em duas partes:
  - Ordenação: Aplicamos o lema *insertPreservesSorted* sobre o resultado da chamada recursiva. Como a hipótese garante que a cauda ordenada permanece ordenada, a inserção mantém essa propriedade.
  - Permutação: Utilizamos a transitividade e o lema *insertPreservesPerm*. Sabemos que inserir a cabeça na cauda ordenada gera uma permutação da lista original, completando a prova.

# 5 Conclusão

A formalização apresentada demonstrou mecanicamente que o algoritmo Insertion Sort satisfaz as propriedades de ordenação e permutação. O uso da indução estrutural e a decomposição em casos (baseada na comparação entre elementos) permitiram cobrir exaustivamente todos os cenários de execução. A verificação pelo Coq elimina a possibilidade de erros lógicos comuns em provas manuais, garantindo que a lista de saída é, inequivocamente, uma versão ordenada da lista de entrada.

# Referências


[1] THE COQ DEVELOPMENT TEAM. *The Coq Proof Assistant Reference Manual*. INRIA. Disponível em: https://rocq-prover.org/docs. Acesso em: nov. 2025.

[2] DE MOURA, Flávio L. C. *Notas de aula da disciplina Logica Computacional 1*. Universidade de Brasília (UnB). Disponível em: https://flaviomoura.info. Acesso em: nov. 2025.