# Algoritmi per inserzione

- Ordinano gli elementi inserendoli nella posizione corretta
- Metodo semplice e intuitivo
- Utile per elenchi quasi ordinati
- Basato sul confronto e spostamento dei valori

# Principio di funzionamento

- Inizia dal secondo elemento
- Lo confronta con quelli precedenti
- Lo inserisce nella posizione corretta
- Ripete per tutti gli elementi

# Esempio passo-passo

- Lista iniziale: `[5, 3, 8, 2]`
- Prende `3`, confronta con `5`, inserisce prima
- Lista: `[3, 5, 8, 2]`
- Prende `8` → già ordinato
- Prende `2` → inserisce in prima posizione

# Pseudocodice

```
per i da 1 a n-1:
    chiave = lista[i]
    j = i-1
    mentre j >= 0 e lista[j] > chiave:
        lista[j+1] = lista[j]
        j = j-1
    lista[j+1] = chiave
```

- Evidenzia inserimento e spostamento
- Metodo iterativo e didattico

# Complessità

- Tempo: O(n^2) nel caso peggiore
- Memoria: O(1)
- Più efficiente del selection sort su piccoli insiemi quasi ordinati
- Adatto per didattica

# Vantaggi

- Semplice da capire e implementare
- Buon comportamento su piccoli dataset
- Ordinamento stabile (mantiene ordine relativo)
- Adatto a dati quasi ordinati

# Svantaggi

- Inefficiente su grandi dataset casuali
- Numero di confronti elevato
- Non adatto per applicazioni professionali su grandi quantità di dati
- Meglio algoritmi avanzati come quicksort o mergesort