Resumo – Algoritmos de Ordenação

Algoritmos $O(n^2)$ – simples, mas lentos em listas grandes

- **Bubble Sort**
  - Ideia: troca vizinhos até ordenar.
  - Complexidade: $O(n^2)$.
  - Estável ✓.
  - Uso: aprendizado, listas muito pequenas.

- **Selection Sort**
  - Ideia: seleciona o menor elemento e coloca na posição correta.
  - Complexidade: $O(n^2)$.
  - Instável ✗.
  - Uso: poucas trocas, mas ineficiente em listas grandes.

- **Insertion Sort**
  - Ideia: insere cada elemento na parte já ordenada (como cartas de baralho).
  - Complexidade: $O(n^2)$, mas $O(n)$ em listas quase ordenadas.
  - Estável ✓.
  - Uso: listas pequenas ou quase ordenadas; parte de algoritmos híbridos (ex.: TimSort).

Algoritmos O(nlogn) – eficientes para grandes volumes

- **Merge Sort**
  - Ideia: divide e conquista, combina sublistas ordenadas.
  - Complexidade: O(nlogn).
  - Estável ✓.
  - Uso: listas grandes, quando estabilidade é necessária.

- **Quick Sort**
  - Ideia: escolhe um pivô, particiona lista em menores/maiores, recursivo.
  - Complexidade: O(nlogn) médio, $O(n^2)$ pior caso.
  - Instável ✗.
  - Uso: muito rápido na prática, padrão em várias linguagens.

- **Heap Sort**
  - Ideia: constrói heap máximo e extrai o maior elemento.
  - Complexidade: O(nlogn).
  - Instável ✗.
  - Uso: robusto, não precisa de memória extra, mas menos eficiente que Quick/Merge.

Comparação Geral

Algoritmo	**Complexidade**	**Estável**	**Melhor uso**
Bubble Sort	$O(n^2)$	✓	Ensino, listas pequenas
Selection Sort	$O(n^2)$	✗	Poucas trocas
Insertion Sort	$O(n^2)$	✓	Listas pequenas/quase ordenadas
Merge Sort	O(nlogn)	✓	Listas grandes, estabilidade
Quick Sort	O(nlogn) médio	✗	Listas grandes, rapidez prática
Heap Sort	O(nlogn)	✗	Robusto, sem memória extra