**Resumo – Algoritmos de Ordenação**

**Algoritmos O(n2) – simples, mas lentos em listas grandes**

* **Selection Sort**
  + **Ideia**: seleciona o menor elemento e coloca na posição correta.
  + Complexidade: O(**n2**).
  + Instável ✘.
  + Uso: poucas trocas, mas ineficiente em listas grandes.
* **Bubble Sort**
  + **Ideia**: troca vizinhos até ordenar.
  + Complexidade: O(**n2**).
  + Estável ✔.
  + Uso: aprendizado, listas muito pequenas.
* **Insertion Sort**
  + **Ideia**: insere cada elemento na parte já ordenada (como cartas de baralho).
  + Complexidade: O(**n2**), 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(**n2**) 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** |
| --- | --- | --- | --- |
| Selection Sort | O(n2) | ✘ | Poucas trocas |
| Bubble Sort | O(n2) | ✔ | Ensino, listas pequenas |
| Insertion Sort | O(n2) | ✔ | 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 |

**Linguagens modernas**: Utilizam **algoritmos híbridos**, ex: **TimSort**.