1. Bubble Sort

- **Descrição:** Comparações e trocas consecutivas entre elementos adjacentes.
- **Complexidade:**
    - **Melhor Caso:** O(n) (array já ordenado).
    - **Pior Caso:** O(n^2) (array completamente desordenado).
    - **Médio Caso:** O(n^2)
- **Uso:** Didático; não é eficiente para grandes conjuntos de dados.

---

2. Selection Sort

- **Descrição:** Seleciona o menor elemento do array e o coloca na posição correta em cada iteração.
- **Complexidade:**
    - **Melhor Caso:** O(n^2)
    - **Pior Caso:** O(n^2
    - **Médio Caso:** O(n^2)
- **Uso:** Simples, mas raramente prático.

---

3. Insertion Sort

- **Descrição:** Insere elementos de forma ordenada em uma sublista.
- **Complexidade:**
    - **Melhor Caso:** O(n) (array já ordenado).
    - **Pior Caso:** O(n^2) (array em ordem reversa).
    - **Médio Caso:** O(n^2)
- **Uso:** Bom para arrays pequenos ou quase ordenados.

---

4. Merge Sort

- **Descrição:** Divide o array em partes menores, ordena-as e as combina.
- **Complexidade:**
    - **Melhor Caso:** O(n log n).
    - **Pior Caso:** O(n log n)
    - **Médio Caso:** O(n log n)
- **Uso:** Eficiente, especialmente para grandes conjuntos de dados.

---

5. QuickSort

- **Descrição:** Escolhe um pivô, divide os elementos em menores e maiores que o pivô e ordena recursivamente.
- **Complexidade:**
    - **Melhor Caso:** O(n log n)
    - **Pior Caso:** O(n^2) (escolha ruim do pivô, como array já ordenado).
    - **Médio Caso:** O(n log n)
- **Uso:** Muito eficiente na prática, especialmente com otimizações como pivô aleatório.

---

6. HeapSort

- **Descrição:** Constrói um heap (estrutura de árvore) e extrai o maior elemento repetidamente.
- **Complexidade:**
    - **Melhor Caso:** O(n log n)
    - **Pior Caso:** O(n log n)
    - **Médio Caso:** O(n log n)
- **Uso:** Bom para grandes conjuntos de dados; estável no uso de memória.