=========ALUNOS==========
Luiz Henrique Barbosa Dias RM:562399
Kenzo de Melo Sato RM:563648
Gregory Debom RM:562346
Nathan Lopes RM: 563507
==========
TURMA: 1CCPH
DATA:03/09/2025
—----------
—----------

O código em C realiza uma **análise comparativa de algoritmos de ordenação**. Ele testa e mede o desempenho de três algoritmos (Bubble Sort, Insertion Sort e QSort) em diferentes cenários de dados (aleatórios, ordenados, reversos e quase ordenados) e para diferentes tamanhos de arrays. O objetivo principal é coletar dados brutos sobre o tempo de execução e o número de comparações realizadas por cada algoritmo em cada cenário.

O código opera em um ciclo principal que itera sobre diferentes tamanhos de arrays e cenários de dados. Para cada combinação de tamanho e cenário, ele executa as seguintes funcionalidades:

1. **Geração de Dados:** Cria um array de inteiros de um tamanho específico e preenche-o com dados que podem ser **aleatórios**, **ordenados**, **reversos** ou **quase ordenados**.
2. **Testes de Ordenação:** Executa os algoritmos **Bubble Sort**, **Insertion Sort** e `qsort` no array gerado.
3. **Medição de Desempenho:** Para cada teste, o código mede o **tempo de execução** e **conta o número de comparações** que o algoritmo fez para ordenar o array.
4. **Impressão de Resultados:** Imprime na tela os resultados de cada teste, mostrando o cenário, o tamanho do array, o tempo levado e o número de comparações.

Essas funcionalidades combinadas servem para comparar a eficiência de diferentes algoritmos de ordenação em várias condições.

—----------

Observações sobre o **Bubble sort**:

O **bubble sort** não necessariamente devolverá uma nova lista ordenada, mas sim ele poderá apenas ordenar a lista já existente, trocando os “objetos de lugar”;

No melhor caso, o desempenho do bubble sort pode chegar a ser O(n), onde uma lista já está ordenada e o algoritmo só irá precisar percorrer uma vez para assim validar sua ordenação;

Em seu médio caso sua complexidade pode chegar a ser $O(n^2)$;

Observações sobre o **insertion sort**:

Ao contrário do bubble sort mencionado acima, o insertion sort realmente vai comparando os itens e criando uma tabela de cada vez, e as comparando novamente;

ele usa um sistema de “direita e esquerda”, sempre comparando um item com o item chave, se o próximo item for maior ou menor que o item chave ele irá mover para sua respectiva posição (para sua direita ou esquerda);

Observações sobre o **qsort**:

O qsort é uma função da biblioteca padrão do C que ordena um array no local, ou seja, ele modifica a lista já existente sem a necessidade de alocar memória para criar uma nova.

Ao contrário de outros algoritmos, o qsort é genérico e pode ordenar qualquer tipo de dado (inteiros, structs, strings, etc.). Para isso, o programador deve fornecer uma **função de comparação** que define a ordem de dois elementos.

Em seu caso médio, o desempenho do qsort é bastante eficiente, com uma complexidade de tempo de O(n log n).