AED II - 2023 - ATIVIDADE DE PROGRAMAÇÃO 03 - MERGESORT

Instruções:

1. E/S: tanto a entrada quanto a saída de dados devem ser "secas", ou seja, não devem apresentar frases explicativas. Siga o modelo fornecido e apenas complete as partes informadas (veja o exemplo abaixo).
2. Identificadores de variáveis: escolha nomes apropriados
3. Documentação: inclua cabeçalho, comentários e indentação no programa.
4. Submeta o programa no sistema Judge: `https://judge.unifesp.br/aediiS1A23/`

Descrição:

Chaves associadas a registros nem sempre são estruturadas em um vetor. Em aplicações do mundo real, essas chaves podem estar em listas simplesmente encadeadas (ou ligadas), as quais permitem que o conjunto de chaves cresça dinamicamente.

Para rápido acesso aos registros, muitas vezes é necessário que suas chaves estejam ordenadas. No entanto, ordenar chaves estruturadas em listas simplesmente ligadas não é uma tarefa simples. Isso porque o acesso aleatório às chaves nessas listas é mais lento ($O(n)$) que em vetores ($O(1)$), o que pode tornar a computação dispendiosa para algoritmos como o *QuickSort* (que veremos em breve) ou até mesmo impossível para o *HeapSort* (que também veremos em breve). Em situações como essa, o algoritmo *MergeSort* aparece como uma ótima opção.

Para a atividade desta semana, você deve implementar o algoritmo *MergeSort* para ordenar chaves em uma lista simplesmente ligada. Além disso, seu algoritmo *MergeSort* deve indicar o nível mais profundo de recursão atingido (quantidade de chamadas recursivas), considerando a primeira chamada a função/algoritmo *MergeSort* (normalmente a partir do programa principal) é considerada nível 0 (zero).

De modo geral, os passos computacionais do *MergeSort* são os mesmos usados para chaves estruturadas em um vetor: 1) encontra-se o meio do conjunto de chaves; 2) ordena-se a metade da esquerda; 3) ordena-se a metade da direita; e 4) mescla-se essas duas metades. Esses procedimentos são aplicados recursivamente até que todo o conjunto esteja ordenado. No caso de listas simplesmente ligadas, no entanto, é necessário estratégias diferentes para encontrar o meio da lista – a partir do qual se determina as metades da direita e da esquerda a serem ordenadas – e para mesclar listas previamente ordenadas.

Considere as seguintes condições:

1. Sua solução deve implementar a versão **recursiva** do MergeSort;
2. A complexidade do algoritmo deve se manter em $O(n \log n)$ (atenção ao algoritmo que encontra o meio de uma lista encadeada);

3. Em caso de lista com número ímpar de nós, a chave no nó do meio da lista deverá ser incluída na metade da esquerda após a divisão em duas metades.
4. Use lista **simplesmente** ligada;
5. O código-fonte **deve** ser escrito em C/C++;
6. **Toda** memória alocada dinamicamente deve ser desalocada;
7. **Nenhuma** variável global deve ser utilizada;

Soluções que violem as condições acima não serão aceitas.

Entrada do programa:

A entrada contém duas linhas. A primeira contém o número de chaves a ser ordenadas, enquanto a segunda linha contém as chaves a serem ordenadas separadas por um espaço em branco.

Saída do programa:

A primeira linha da saída deve apresentar as chaves ordenadas e separadas por um espaço em branco. A segunda linha mostra o nível mais profundo de recursão atingido.

Exemplos de entrada e saída:

Exemplos de entrada	Exemplos de saída
5 1 5 3 4 2	1 2 3 4 5 3
6 3 5 2 1 4 6	1 2 3 4 5 6 3
10 1 2 3 4 5 6 8 9 10 20	1 2 3 4 5 6 8 9 10 20 4

Tabela 1: Exemplos de entrada e saída