AED II - 2025 - ATIVIDADE DE PROGRAMAÇÃO 03 - ALGORITMOS DE ORDENAÇÃO COM ESTRATÉGIA DIVIDIR PARA CONQUISTAR

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: Turma Integral `https://judge.unifesp.br/AED2IA012025`, Turma Noturna: `https://judge.unifesp.br/AED2N012025`
5. Plágios serão verificados pela plataforma MOSS (`https://theory.stanford.edu/~aiken/moss/`). Altas taxas de similaridade não serão tolerados.

Descrição:

Esta atividade de programação tem o objetivo de avaliar a eficiência de dois algoritmos de Ordenação, de ordem $O(N.logN)$, baseados na estratégia Dividir para Conquistar.

A eficiência do algoritmo *QuickSort* está ligada diretamente à escolha do elemento que servirá como pivô. Quando o último elemento do vetor é escolhido, vetores previamente ordenados (em ordem crescente ou decrescente) levam a uma complexidade quadrática ($O(n^2)$) do algoritmo, aumentando em muito o tempo de execução do código. Uma forma de diminuir a probabilidade de pior caso é escolher o pivô pelo método da mediana de três. Nesse método, são separados três elementos do vetor - por exemplo, o primeiro, o último, e o do meio - e o pivô será o elemento com valor intermediário, ou mediano, entre os três. Por exemplo, se os elementos separados são a, b, e c tais que:

$$a < b \leq c$$

O pivô deverá ser o elemento b.

Além disso, muitas vezes, ao restar apenas uma pequena quantidade de elementos a ordenar, os algoritmos de ordenação mais simples podem se sair mais eficientes. Dessa forma, sistemas de ordenação que combinam dois ou mais diferentes algoritmos de ordenação, podem apresentar melhor desempenho computacional.

Na atividade desta semana, compare três algoritmos de ordenação e conte, em cada um dos algoritmos, o número total de comparações realizadas envolvendo valores dos elementos a serem ordenados. As versões a serem implementadas são as seguintes:

1. *MergeSort* (MS).
2. *QuickSort* com pivô mediana-de-três (QS-M3).
3. *QuickSort* Híbrido (mediana-de-três + *Insertion Sort* para *subarrays* com até 5 elementos) (HÍBRIDO).

Você deverá implementar as versões recursivas do algoritmo *QuickSort* e *MergeSort*. Para a versão híbrida, deve-se usar o mesmo algoritmo *QuickSort*, porém, integrado a versão iterativa do algoritmo *InsertionSort*, a partir das versões mostradas em sala.

Seu programa deverá retornar, para cada algoritmo, as seguintes informações: em uma linha deve-se mostrar o *array*, de números inteiros, devidamente ordenado, e na linha seguinte informar um número inteiro referente ao total de comparações realizadas. Isso deve ser repetido para os 3 algoritmos solicitados, na sequência pré-determinada.

Considere as seguintes condições:

1. Sua solução deve implementar as versões **recursivas** do *MergeSort* e *QuickSort*;
2. O *array* de entrada deverá ser ordenado **três vezes**: na primeira vez pela execução do *MergeSort*, na segunda pela execução do *QuickSort* com mediana de três, e na última vez pela execução do algoritmo híbrido de ordenação;
3. O código-fonte **deve** ser escrito em C/C++;
4. **Toda** memória alocada dinamicamente (C/C++) deve ser desalocada;
5. **Nenhuma** variável global deve ser utilizada;

Regras para cálculo do Pivô e para divisão do *array* de entrada

Para o algoritmo *QuickSort* com Mediana-de-Três (QS-M3), o Pivô deverá ser definido como o valor mediano entre os elementos: $A[inicio]$, $A[meio]$, $A[fim]$.

Caso se tenha um *Subarray* com a quantidade ímpar de elementos, a posição do meio, para cálculo da mediana de três, bem como para divisão do domínio do *MergeSort*, deve ser definido da seguinte forma:

$$meio = inicio + (fim - inicio)/2$$

.

Caso se tenha um *Subarray* com a quantidade par de elementos, a posição do meio deve ser definido com arredondamento para baixo, exemplo: tamanho 4 implica que o índice $meio$ deve ser igual à 1. Com o *MergeSort* para $N = 5$, divide-se o arranjo em duas partes com 2 e 3 elementos, por exemplo.

O particionamento do *array* no *QuickSort* deve retornar o arranjo da seguinte forma: **Elementos $\leq$ pivô à esquerda, e elementos $\geq$ pivô à direita**. Caso não se tenha ele-

mentos menores que o pivô, não deverá haver elementos à esquerda, no qual o primeiro elemento do *subarray* deverá ser o próprio pivô.

Entrada:

A primeira linha contém o valor de um inteiro n que representa o tamanho do vetor a ser ordenado.

A segunda linha contém os elementos, como números inteiros, do vetor de entrada, separados por espaço.

Saída:

A saída deve conter 6 (seis) linhas, onde as primeiras duas linhas deve mostrar o *array*, de números inteiros, devidamente ordenado pelo algoritmo *MergeSort* e na segunda linha a quantidade de comparações efetuadas pelo algoritmo, entre valores do *array* a ordenar. A terceira e quarta linha, bem como a quinta e sexta, também deverão informar o mesmo, ou seja, o *array* ordenado e a quantidade de comparações, porém, para o algoritmo *QuickSort* com mediana de três, e para a versão híbrida, respectivamente.

Exemplos de entrada e saída:

- *input01*:

Entrada	**Saída**
3 1 2 3	1 2 3 5 1 2 3 5 1 2 3 2

Tabela 1: Exemplos de entrada e saída 01

- *input02*

Entrada	**Saída**
4 4 3 2 1	1 2 3 4 8 1 2 3 4 10 1 2 3 4 6

Tabela 2: Exemplos de entrada e saída 02

- *input03*

Entrada	**Saída**
6 5 2 9 1 4 3	1 2 3 4 5 9 16 1 2 3 4 5 9 18 1 2 3 4 5 9 12

Tabela 3: Exemplos de entrada e saída 03