Quiz 08 - Algoritmos de Ordenação

- Entrega 31 mar em 9:15
- Pontos 100
- Perguntas 7
- Disponível 31 mar em 9:00 - 31 mar em 9:15 15 minutos
- Limite de tempo Nenhum

Instruções

Este quiz aborda os algoritmos de ordenação Quicksort, Mergesort, Heapsort, Shellsort, Countingsort, Seleção, Inserção e Bolha. Ele tem 7 questões de múltiplas escolha sendo que as seis primeiras valem 15 pontos cada e a última, 10 pontos. Após o preenchimento de uma questão, o aluno não tem a opção de retorno à mesma. Este trabalho deve ser efetuado sem consulta.

Este teste foi travado 31 mar em 9:15.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	11 minutos	85 de 100

Pontuação deste teste: 85 de 100
Enviado 31 mar em 9:11
Esta tentativa levou 11 minutos.

Pergunta 1
15 / 15 pts

A ordenação interna, também conhecida como ordenação em memória principal, é um processo de rearranjo de elementos em uma estrutura de dados na memória do computador, como um array, uma lista ou uma árvore, de acordo com uma determinada ordem ou critério. A ordenação interna é chamada assim porque os elementos a serem ordenados estão todos armazenados na memória principal do sistema, ao contrário da ordenação externa, que envolve a manipulação de dados armazenados em dispositivos de armazenamento secundário, como discos rígidos.

Existem vários algoritmos populares para realizar a ordenação interna, cada um com suas características e complexidades diferentes. Os algoritmos de ordenação interna desempenham um papel crucial no processamento eficiente de dados em diversas áreas da programação. Eles

permitem organizar elementos em uma determinada sequência, facilitando a busca, análise e manipulação dos dados.

Uma técnica tradicional de ordenação interna simples de se implementar e que utiliza o método de comparação de pares de elementos adjacentes e realiza trocas quando necessário trata-se do

- Insertion Sort.
- Merge Sort.
- Selection Sort.
- Quick Sort.

Correto!

- Bubble Sort.

Correta. O Bubble Sort é um algoritmo de ordenação que compara pares de elementos adjacentes e realiza trocas quando necessário, movendo o maior elemento gradualmente para o final da lista. É uma técnica simples de implementar.

Pergunta 2
15 / 15 pts

A eficiência dos algoritmos de ordenação e pesquisa é uma preocupação fundamental no campo da Ciência da Computação. Dentre os algoritmos estudados, o Quicksort se destaca por sua abordagem inovadora e eficiente na resolução desse tipo de problema. Esse algoritmo utiliza a estratégia de dividir e conquistar, o que o diferencia dos demais algoritmos de ordenação como Inserção, Seleção e Bubblesort.

Considerando esse contexto, avalie as asserções a seguir e a relação proposta entre elas.

I) O algoritmo Quicksort tem complexidade média $O(n \times log\ n)$, enquanto os algoritmos Inserção, Seleção e Bubblesort possuem complexidade $O(n^2)$.

PORQUE

II) A estratégia de dividir e conquistar do Quicksort permite reduzir consideravelmente o número de comparações e trocas necessárias para ordenar uma lista, resultando em um desempenho superior em relação aos demais algoritmos estudados.

A respeito dessas asserções, assinale a opção correta.

- As asserções I e II são proposições falsas.
- A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

Correto!

- As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.

A asserção I é verdadeira porque o algoritmo Quicksort tem uma complexidade média de O(n log n), enquanto os algoritmos Inserção, Seleção e Bubblesort possuem complexidade de O(n^2), o que significa que o Quicksort tende a ser mais eficiente em termos de tempo de execução.

A asserção II é verdadeira porque a estratégia de dividir e conquistar do Quicksort permite reduzir consideravelmente o número de comparações e trocas necessárias para ordenar uma lista. Isso ocorre porque o Quicksort seleciona um elemento pivô, divide a lista em duas partes com base nesse pivô e recursivamente aplica o mesmo processo em cada uma dessas partes, reduzindo assim o número de comparações e trocas em comparação aos algoritmos de ordenação com complexidade quadrática.

Portanto, as asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.

Pergunta 3
15 / 15 pts

A respeito do problema de ordenação avalie as seguintes afirmativas:

I. O problema é $O\left(n \times \lg(n)\right)$.
II. O algoritmo HeapSort é ótimo.
III. O algoritmo de Inserção possui complexidade quadrática.

Está correto o que se afirma em

Correto!

- I, II e III
- I apenas
- I e III apenas
- II e III apenas
- II apenas

I. CORRETA. Sabe-se que o limite inferior do problema de ordenação é $O(n \times \lg(n))$.

II. CORRETA. o Heapsort é dito ótimo porque ele ordena vetores com o custo de $O(n \times \lg(n))$ comparações entre registros.

II. CORRETA. o Inserção realiza um número quadrático de comparações entre registros nos seus pior e médio casos.

Pergunta 4
15 / 15 pts

A compreensão dos algoritmos de ordenação é fundamental para formar profissionais capazes de analisar problemas e desenvolver soluções algorítmicas inovadoras. Entre esses algoritmos, o Quicksort se destaca pela sua eficiência em uma ampla gama de cenários, diferenciando-se significativamente de algoritmos como Inserção, Seleção e Bubblesort em termos de complexidade temporal e abordagem de ordenação. Considere o seguinte código em C# que implementa o algoritmo Quicksort:

```
void Quicksort(int[] array, int inicio, int fim) {
    if (inicio < fim) {
        int posicaoPivo = Particionar(array, inicio,
fim);
        Quicksort(array, inicio, posicaoPivo - 1);
        Quicksort(array, posicaoPivo + 1, fim);
    }
}

int Particionar(int[] array, int inicio, int fim) {
    int pivo = array[fim];
    int i = (inicio - 1);
    for (int j = inicio; j < fim; j++) {
        if (array[j] <= pivo) {
            i++;
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }
    int tempPivo = array[i + 1];
```

```
    array[i + 1] = array[fim];
    array[fim] = tempPivo;
    return i + 1;
}
```

Este código exemplifica o uso de uma estratégia de divisão e conquista para ordenar dados, onde o conjunto é dividido em subconjuntos menores que são ordenados independentemente. A escolha do pivo é crucial para a eficiência do algoritmo, e diferentes estratégias podem ser adotadas para essa escolha, influenciando diretamente no desempenho do algoritmo em diferentes conjuntos de dados.

Considerando esse contexto, avalie as asserções a seguir e a relação proposta entre elas.

I) O algoritmo Quicksort, conforme implementado, é eficaz para grandes conjuntos de dados devido à sua abordagem de divisão e conquista.

PORQUE

II) A eficácia do Quicksort em grandes conjuntos de dados deve-se ao fato de que, ao dividir o conjunto em subconjuntos menores e ordená-los independentemente, reduz-se significativamente o número total de comparações e trocas necessárias em comparação com algoritmos como Inserção, Seleção e Bubblesort.

A respeito dessas asserções, assinale a opção correta.

Correto!

- As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- As asserções I e II são proposições falsas.

As duas asserções são verdadeiras, e a II é uma justificativa correta da I. A estratégia de divisão

e conquista do Quicksort permite que o algoritmo seja particularmente eficaz em grandes conjuntos de dados. Isso ocorre porque, ao dividir o conjunto em partes menores e resolver essas partes independentemente, o Quicksort consegue reduzir drasticamente o número de comparações e trocas necessárias para ordenar o conjunto inteiro. Esta abordagem contrasta com algoritmos de ordenação mais simples, como Inserção, Seleção e Bubblesort, que geralmente requerem mais comparações e trocas em conjuntos de dados grandes, tornando o Quicksort uma opção superior para esses cenários.

Pergunta 5
0 / 15 pts

O Heapsort é um algoritmo de ordenação clássico que utiliza uma estrutura de *heap* invertido para ordenar os elementos do vetor. A primeira etapa desse algoritmo organiza todos os elementos do vetor em um *heap* invertido, estrutura de dados na qual todos os elementos são maiores ou iguais aos seus filhos. A segunda etapa remove a cabeça do *heap* (maior elemento), reduzindo o tamanho do mesmo em uma unidade. Em seguida, a posição que ficou livre recebe o elemento removido. A segunda etapa repete esse processo até que o *heap* tenha somente um elemento, o menor de todos. Considerando a descrição anterior e seus conhecimentos sobre algoritmos de ordenação, avalie as asserções que se seguem:

I. A ordem de complexidade do Heapsort é $O(n \times lg\, n)$ para todos os casos.

PORQUE

II. A ordem de complexidade da primeira etapa é $O(n)$ no melhor caso e $O(n \times lg\, n)$ no pior caso e a da segunda etapa é $O(n \times lg\, n)$ para todos os casos.

A respeito dessas asserções, assinale a opção correta.

Resposta correta

- As asserções I e II são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
- As asserções I e II são proposições falsas.
- A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira.
- As asserções I e II são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira.

Você respondeu

- A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.

As duas afirmações são verdadeiras e a segunda, realmente, justifica a primeira.

Pergunta 6
15 / 15 pts

O melhor caso do algoritmo Quicksort ocorre quando a partição divide o array de maneira equilibrada, isto é, quando o pivô escolhido divide o array em duas partes aproximadamente iguais. Assim, cada partição subsequente continuará a dividir os dados restantes igualmente, resultando em um arranjo equilibrado, semelhante a uma árvore binária balanceada. Este cenário é ideal porque maximiza a eficiência do algoritmo, mantendo a profundidade da pilha de chamada recursiva ao mínimo e permitindo que o Quicksort opere na sua eficiência máxima de $\Theta(n \log n)$.

Contudo, na prática, para trabalhar próximo do melhor caso é importante escolher um bom pivô, como o uso do método "mediana de três", que consiste em pegar a mediana entre o primeiro, o último e o elemento central do array.

Sobre Quicksort, avalie as afirmações a seguir.

I. O melhor caso do Quicksort é $\Omega(n)$.

II. O melhor caso do Quicksort é $O(n^3)$.

Assinale a alternativa correta.

- A afirmação I é verdadeira e a II é falsa.
- A afirmação I é falsa e a II é verdadeira.

Correto!

- As afirmações I e II são verdadeiras.
- As afirmações I e II são falsas.

A afirmação I é verdadeira porque se o melhor caso do Quicksort é $\Theta(n \log n)$, então ele é $\Omega(n \log n)$ e se ele é $\Omega(n \log n)$, ele também é $\Omega(n)$.

A afirmação II é verdadeira porque se o melhor caso do Quicksort é $\Theta(n \log n)$, então ele é $O(n \log n)$ e se ele é $O(n \log n)$, ele também é $O(n^3)$.

Pergunta 7
10 / 10 pts

A ordenação interna é um problema clássico na Computação. Considerando-o, avalie as asserções que se seguem:

I. O algoritmo Countingsort ordena um vetor com custo linear.

PORQUE

II. O limite inferior do problema de ordenação interna é $\Theta(n \times lg\, n)$ para a comparação entre registros.

A respeito dessas asserções, assinale a opção correta.

Correto!

- As asserções I e II são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira
- A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa
- As asserções I e II são proposições falsas
- A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira
- As asserções I e II são proposições verdadeiras, e a segunda é uma justificativa correta da primeira

I - CORRETA: O algoritmo Countingsort efetua em tempo linear $\Theta\,(n)$ a ordenação dos elementos de um vetor. Ele considera três vetores: entrada, contagem e saída. O primeiro passo é em $\Theta\,(n)$ é criar o vetor de contagem de tal forma que cada posição tenha o número de elementos menores ou iguais aquela posição. O segundo passo é copiar cada elemento do vetor de entrada para o de saída mapeando de tal forma que a posição do elemento no vetor de saída será mapeada a partir do vetor de contagem.

II - CORRETA: É impossível ordenar um vetor com menos do que $\Theta(n \times lg\, n)$ comparações entre os elementos do vetor. O Countingsort não se aplica a tal regra porque ele triplica o espaço de memória e não funciona para qualquer tipo de elemento.

As duas afirmações são independentes.

Pontuação do teste: 85 de 100