Quiz 07 - Ordenação - Seleção, Inserção e Bolha

- Entrega 24 mar em 9:15
- Pontos 100
- Perguntas 8
- Disponível 24 mar em 9:00 - 24 mar em 9:15 15 minutos
- Limite de tempo Nenhum

Instruções

Este quiz aborda os algoritmos de Seleção, Inserção e Bolha. 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 24 mar em 9:15.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	12 minutos	85 de 100

Pontuação deste teste: 85 de 100
Enviado 24 mar em 9:13
Esta tentativa levou 12 minutos.

Pergunta 1
5 / 5 pts
O Algoritmo de Seleção é mais eficiente na prática que o da Bolha para a maioria dos conjuntos de dados.

- Falso

Correto!

- Verdadeiro

Verdadeiro. Ambos têm complexidade quadrática para o número de comparações, contudo, o Seleção apresenta um número linear de trocas.

Pergunta 2
5 / 5 pts
O Algoritmo de Seleção é considerado ótimo em termos de complexidade de tempo para todos os

tamanhos de conjuntos de dados.
Correto!

- Falso
- Verdadeiro

Falsa. O Algoritmo de Seleção tem complexidade $\Theta(n^2)$, o que não é considerado ótimo para grandes conjuntos de dados.

Pergunta 3
15 / 15 pts

Nos jogos digitais, o desempenho e a eficiência do processamento de dados são fatores críticos que impactam diretamente a experiência do jogador. A organização dos dados em memória primária, especialmente em situações que envolvem buscas e manipulações frequentes, é uma prática essencial para otimizar o uso dos recursos computacionais. Para isso, algoritmos de ordenação desempenham um papel fundamental, sendo amplamente utilizados em operações como classificação de scores, organização de inventários ou até mesmo para melhorar o desempenho de sistemas de matchmaking. Entre os métodos disponíveis para ordenação em memória primária, as linguagens de programação oferecem recursos nativos que simplificam a implementação e garantem maior confiabilidade. Esses recursos são frequentemente ajustados para maximizar a eficiência em termos de complexidade computacional, aproveitando técnicas consagradas, como o uso de algoritmos de complexidade O(n lg n). Entender como e quando utilizar esses recursos é essencial para projetar sistemas eficazes e robustos. Considere o trecho de código em C#:

```
        int[] scores = { 500, 1200, 700, 1500, 800 };
        var sortedScores = scores.OrderByDescending(x => x).ToArray();

        Console.WriteLine(string.Join(", ", sortedScores));
```

Com base no código acima, podemos afirmar que o método utilizado para ordenar os elementos do array scores:

Emprega diretamente SelectionSort, por ser o método mais simples para implementação de ordenação nativa.

Correto!

O método busca uma complexidade O(n lg n), característica de algoritmos eficientes, sem especificar o algoritmo de ordenação.
Correta: O .NET utiliza algoritmos otimizados com complexidade O(n lg n) para ordenação em memória primária, mas não especifica qual é o algoritmo exato empregado.

Utiliza BubbleSort, pois este é o algoritmo mais eficiente para ordenação em memória primária em pequenas coleções.

Implementa o algoritmo MergeSort, dado que ele é o padrão do método OrderByDescending no .NET.

Realiza QuickSort, dado que o .NET privilegia algoritmos baseados em partições para ordenação.

Pergunta 4
15 / 15 pts

O sucesso de aplicativos e plataformas online da era digital dependem de sua velocidade e precisão. Isso faz com que o desenvolvimento de sistemas para internet se assemelhe à arte de compor uma sinfonia: cada nota, ou neste caso, cada linha de código, deve ser colocada com precisão para criar a harmonia desejada. Dentro dessa composição, os algoritmos de ordenação e pesquisa atuam como os instrumentos, cada um com sua voz e papel únicos, contribuindo para a eficácia e eficiência da obra completa. Entre os métodos disponíveis, Inserção, Seleção e Bubblesort destacam-se por sua simplicidade e eficácia em determinadas condições, servindo como ferramentas interessantes na caixa de ferramentas de um desenvolvedor. No entanto, como em qualquer escolha de instrumentação, a chave está em saber quando e como utilizar cada um destes algoritmos para otimizar tanto o desempenho quanto a utilização de recursos. Esta decisão é influenciada por diversos fatores, incluindo o tamanho do conjunto de dados e os objetivos específicos da aplicação, destacando a importância de uma compreensão profunda de suas características e melhor aplicabilidade. Com relação ao comentado, avalie as afirmações a seguir.

I. O algoritmo de Inserção tem um comportamento bimodal de tal forma que seu desempenho pode variar significativamente entre seu melhor e pior caso.

II. O algoritmo de Seleção é recomendado para garantir o menor tempo de ordenação em listas grandes, devido à sua complexidade constante em todos os casos.

III. O algoritmo de Bubblesort é ideal para aplicações que exigem a máxima eficiência em termos de tempo e espaço, independentemente do tamanho do conjunto de dados.

É correto o que se afirma em:

- I e III, apenas.

Correto!

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

Afirmação I: Correta. O algoritmo de Inserção exibe um comportamento bimodal em termos de performance, sendo muito eficiente para conjuntos de dados pequenos ou quase ordenados, onde pode se aproximar de uma complexidade linear, enquanto que para listas desordenadas ou grandes, sua eficiência diminui significativamente.

Afirmação II: Incorreta. Apesar de o algoritmo de Seleção apresentar uma complexidade de tempo $O(n^2)$ constante, independente do estado inicial dos dados, essa característica não o torna recomendado para listas grandes, uma vez que há algoritmos com melhor desempenho para esses casos.

Afirmação III: Incorreta. O algoritmo de Bubblesort, conhecido por sua simplicidade, é na verdade um dos métodos menos eficientes para ordenação, especialmente em conjuntos de dados grandes, devido à sua complexidade quadrática e ao alto custo das trocas realizadas.

A alternativa correta para esta questão é "I, apenas", refletindo adequadamente as propriedades e situações em que o algoritmo de Inserção pode ser considerado eficiente, contrastando com as limitações dos algoritmos de Seleção e Bubblesort.

Pergunta 5
0 / 15 pts

O desempenho e a eficiência na manipulação de dados são essenciais para criar experiências de jogo envolventes e sem interrupções. A escolha de algoritmos de ordenação e pesquisa adequados pode ter um impacto significativo na velocidade e na economia de recursos do jogo, especialmente quando se lida com grandes volumes de dados, como inventários de jogadores ou rankings de pontuações. Algoritmos como Inserção, Seleção e Bubble Sort oferecem diferentes abordagens para ordenar dados, cada um com suas próprias vantagens e desvantagens em

termos de complexidade computacional. A compreensão profunda dessas diferenças é crucial para desenvolvedores de jogos digitais, permitindo a escolha do algoritmo mais eficiente para cada situação específica. Por exemplo, enquanto o Bubble Sort pode ser intuitivo e simples de implementar, ele pode não ser a melhor escolha para listas muito grandes devido à sua natureza quadrática. Em contraste, algoritmos como o de Inserção podem ser mais adequados para pequenas listas ou listas parcialmente ordenadas, oferecendo uma abordagem mais direcionada e eficiente. Essa diferenciação estratégica pode ser a chave para otimizar o desempenho do jogo e melhorar a experiência geral do jogador. Considerando esse contexto, avalie as asserções a seguir e a relação proposta entre elas.

I) O algoritmo de Seleção é mais eficiente do que o Bubble Sort para ordenar grandes listas de dados em jogos digitais.

PORQUE

II) O algoritmo de Seleção possui uma complexidade de tempo constante, enquanto o Bubble Sort possui uma complexidade de tempo quadrático.

A respeito dessas asserções, assinale a opção correta.

Você respondeu

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

Resposta correta

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

A asserção I é verdadeira no sentido de que o algoritmo de Seleção pode ser, de fato, mais eficiente do que o Bubble Sort para ordenar grandes listas de dados devido ao menor número de trocas necessário.

A asserção II é falsa porque ambos os algoritmos, Seleção e Bubble Sort, possuem uma complexidade de tempo quadrática (O(n^2)) e não constante. Portanto, a correta justificação para a eficiência do algoritmo de Seleção em comparação com o Bubble Sort não se baseia em uma

complexidade de tempo constante, mas sim no fato de que o algoritmo de Seleção minimiza o número de trocas.

Isso torna a opção "A asserção I é uma proposição verdadeira, e a II é uma proposição falsa" a resposta correta.

Pergunta 6
15 / 15 pts

O método Bubble Sort, ou ordenação por bolha, é um algoritmo simples de ordenação. Esse método é denominado "Bubble Sort" porque os valores maiores "borbulham" para o final da lista através das sucessivas trocas, como bolhas de ar se elevam na água. Apesar de não ser o método mais eficiente para listas grandes, sua simplicidade o torna útil para listas pequenas ou para fins educacionais, onde o entendimento do processo de ordenação é o objetivo principal. Considerando o método Bubble Sort aplicado à lista inicial [4, 3, 2, 1], as representações das etapas de ordenação da lista até sua completa organização consiste em:

Correto!

- [4, 3, 2, 1], [3, 4, 2, 1], [3, 2, 4, 1], [3, 2, 1, 4], [2, 3, 1, 4], [2, 1, 3, 4], [1, 2, 3, 4]

Correta. Esta sequência ilustra precisamente o método Bubble Sort, mostrando todas as trocas passo a passo até que a lista [4, 3, 2, 1] esteja completamente ordenada.

- [4, 3, 2, 1], [3, 4, 2, 1], [2, 3, 4, 1], [1, 2, 3, 4]
- [4, 3, 2, 1], [1, 3, 2, 4], [1, 2, 3, 4], [1, 2, 3, 4]
- [4, 3, 2, 1], [3, 4, 2, 1], [3, 2, 4, 1], [2, 3, 4, 1], [2, 3, 1, 4], [2, 1, 3, 4], [1, 2, 3, 4]
- [4, 3, 2, 1], [3, 4, 2, 1], [3, 2, 1, 4], [2, 1, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]

Pergunta 7
15 / 15 pts

O algoritmo a seguir recebe um vetor v de números inteiros e rearranja esse vetor de tal forma que seus elementos, ao final, estejam ordenados de forma crescente.

```
void ordena(int *v, int n)

{
  int i, j, chave;
  for(i = 1; i < n; i++)
  {
    chave = v[i];
    j = i - 1;
```

```
    while(j >= 0 && v[j] < chave)
    {
      v[j-1] = v[j];
      j = j - 1;
    }
    v[j+1] = chave;
  }
}
```

Considerando que nesse algoritmo há erros de lógica que devem ser corrigidos para que os elementos sejam ordenados de forma crescente, assinale a opção correta no que se refere as correções adequadas (ENADE'17).

A linha 4 deve ser corrigida da seguinte forma: *for(i = 1; i < n-1; i++)* e a linha 13, do seguinte modo *v[j-1] = chave;*.

A linha 7 deve ser corrigida da seguinte forma: *j = i + 1;* e a linha 8, do seguinte modo: *while(j >= 0 && v[j] > chave)*.

No answer text provided.

Correto!

A linha 8 deve ser corrigida da seguinte forma: *while(j >= 0 && v[j] > chave)* e a linha 10, do seguinte modo: *v[j+1] = v[j];*.

A linha 10 deve ser corrigida da seguinte forma: *v[j+1] = v[j];* e a linha 13, do seguinte modo: *v[j-1] = chave;*.

O algoritmo em questão é o Inserção e os erros apresentados estão no laço interno. Na versão correta, o elemento a ser inserido na parte ordenada (chave) é comparado com os já ordenados (posições 0 à i). Enquanto o elemento chave é maior que os já ordenados, deslocamos cada elemento já comparado uma posição a mais do que sua posição atual, permitindo a inserção do elemento chave em sua posição correta. Essa inserção acontece após o laço interno.

Pergunta 8
15 / 15 pts

Um algoritmo clássico para a ordenação de elementos de um vetor é o Ordenação por Seleção. Abaixo, temos uma implementação desse algoritmo.

```
for (i = 0; i < (n - 1); i++) {
```

```
    indice = i;
    for (j = (i + 1); j < n; j++){
        if (vet[indice] > vet[j]){
            indice = j;
        }
    }
    tmp = vet[indice];
    vet[indice] = vet[i];
    vet[i] = tmp;
}
```

Considerando o código acima e seus conhecimentos sobre algoritmos de ordenação, avalie as asserções que se seguem:

I. O algoritmo de Seleção deve ser considerado como uma opção para ordenação quando tivermos registros "grandes".

PORQUE

II. O algoritmo de Seleção possui complexidade quadrática - $O(n^2)$ - para o número de comparações envolvendo os elementos do vetor.

A respeito dessas asserções, assinale a opção correta.

- As asserções I e II são proposições falsas.

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

A afirmação I é verdadeira porque o algoritmo de Seleção realiza Θ(n) trocas, o que é vantajoso quando registros "grandes" precisam ser movimentados, pois minimiza o custo associado à movimentação desses registros, apesar de sua complexidade quadrática em termos de comparações.

A afirmação II é verdadeira porque a complexidade do algoritmo de Seleção é Θ(n²), devido às comparações realizadas nos dois laços aninhados (o laço externo executa n−1 vezes, e o interno realiza comparações de forma decrescente).

A segunda asserção não justifica a primeira. O fato de o algoritmo possuir complexidade quadrática para comparações não é o motivo pelo qual ele é adequado para registros "grandes". A adequação é explicada pelo número reduzido de trocas Θ(n), que não está explicitamente mencionado na segunda asserção.

Pontuação do teste: 85 de 100