Quiz 05 - Fundamentos de Análise de Complexidade

- Entrega 26 fev em 9:12
- Pontos 100
- Perguntas 8
- Disponível 26 fev em 8:50 - 26 fev em 9:12 22 minutos
- Limite de tempo Nenhum

Instruções

Este quiz aborda noções de complexidade. Ele tem 7 questões de múltipla escolha, sendo que as quatro primeiras valem 10 pontos cada e as duas últimas 30 pontos cada. 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 26 fev em 9:12.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	13 minutos	100 de 100

Pontuação deste teste: 100 de 100
Enviado 26 fev em 9:11
Esta tentativa levou 13 minutos.

Pergunta 1
10 / 10 pts

O comando de repetição *for* permite que o projetista do algoritmo declare um contador e repita uma lista de comandos enquanto esse contador não alcançar um determinado valor. O *for* também incrementa ou decrementa seu contador para que ele alcance a condição de parada. O trecho de código abaixo contém o comando *for*.

```
for (int i = n - 2; i > 5; i--){
   b *= 3;
}
```

Considerando o código acima, assinale a opção que apresenta o número de vezes que realizamos a operação de multiplicação.

Correto!

- n-7
- n-8
- n-2
- n-6
- n-5

Quando o comando for é inicializado com um inicial e repetido enquanto esse valor é maior que um valor final e decrementado de uma em uma unidade, o número de repetições é igual a inicial - final. Por exemplo, no código acima, quando n = 10, efetuamos três multiplicações quando os valores de i são 8, 7 e 6. Se n = 20, temos treze multiplicações quando os valores de i são 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7 e 6.

Pergunta 2
10 / 10 pts

A contagem do número de operações realizadas por um algoritmo é uma tarefa fundamental para identificar seu custo computacional. O trecho de código abaixo realiza algumas operações.

```
Random gerador = new Random();
gerador.setSeed(4);
int x = Math.abs(gerador.nextInt());

for (int i = 0; i <= n-5; i++){
    if( x % 9 < 4) {
        a *= 2; b *= 3; l *= 2;
    } else if (x % 9 == 5) {
        a *= 2; l *= 3;
    } else  {
        a *= 2;
    }
}
```

Considerando o código acima, marque a opção que apresenta o pior e melhor caso para o número de multiplicações, respectivamente.

- 3(n-4), 1

Correto!

- 3(n-4), (n-4)
- n, n
- 3(n-5), 1

3(n-5), (n-5)

Pergunta 3
10 / 10 pts

O comando condicional *if-else* possibilita a escolha de um grupo de ações a serem executadas quando determinadas condições de entrada são ou não satisfeitas. O trecho de código abaixo contém uma estrutura condicional.

```
if (n < a + 3 || n > b + 4 || n > c + 1){
    l+= 5;
} else {
    l+= 2; k+=3; m+=7; x += 8;
}

if (n < a + 3){
    l+= 2; k+=3; m+=7; x += 8;
} else {
    l+= 5;
}
```

Considerando o código acima, marque a opção que apresenta o melhor e pior caso, respectivamente, para o número de adições.

5 e 12

6 e 12

5 e 9

Correto!

4 e 12

4 e 9

O pior caso tem 12 adições e acontece quando as três condições do primeiro if são falsas e, consequentemente, a condição única do segundo if é falsa dado que ela é igual a primeira condição do primeiro if. Dessa forma, o teste do primeiro if realiza 3 adições e sua lista de comandos, quatro. O teste do segundo if realiza uma adição e mais quatro na lista verdadeira. O melhor tem 4 adições e isso acontece quando a primeira condição é falsa e a segunda verdadeira. Nesse caso, o teste do primeiro if realiza duas adições e sua lista de comandos, uma. No segundo if, temos uma adição do teste e mais uma da lista do else.

Pergunta 4
10 / 10 pts

Um desafio no projeto de algoritmos é a obtenção de um custo computacional reduzido. Para

isso, uma habilidade do projetista é contar o número de operações realizadas pelo algoritmo. O trecho de código abaixo realiza algumas operações.

```
for (int i = n; i > 0; i /= 2){
  a *= 2;
}
```

Considerando o código acima, assinale a opção que apresenta a função de complexidade *f(n)* para o melhor e pior caso considerando a operação de multiplicação.

- $f(n) = lg(n) + 1$
- $f(n) = n + 1$

Correto!

- $f(n) = \lfloor lg(n) \rfloor + 1$
- $f(n) = \lceil lg(n) \rceil + 1$
- $f(n) = n$

Pergunta 5
15 / 15 pts

A análise de algoritmos é o estudo do desempenho de algoritmos, especialmente seu tempo de execução e necessidade de espaço. Um componente fundamental deste estudo é a caracterização de algoritmos em termos de seu tempo de execução ou espaço requerido como uma função do tamanho da entrada, conhecida como complexidade de tempo e complexidade de espaço, respectivamente. Além disso, os algoritmos são frequentemente classificados em categorias com base em sua complexidade, tais como algoritmos de tempo polinomial e algoritmos de tempo exponencial.

Com relação ao comentado, avalie as afirmações a seguir.

I. Um algoritmo com complexidade de tempo O(n log n) tem melhor performance que um algoritmo com complexidade de tempo O(n^2) para grandes entradas.

II. A complexidade de espaço de um algoritmo se refere ao tempo que ele leva para executar.

III. Um algoritmo de ordenação por seleção, na melhor das hipóteses, tem complexidade de tempo O(n log n).

É correto o que se afirma em

- I e II apenas.

II, apenas.

I e III, apenas.

Correto!

I, apenas.

I. Esta afirmação é verdadeira. O(n log n) tem melhor performance que O(n^2) para grandes entradas. À medida que n se torna grande, o tempo de execução do algoritmo de O(n log n) cresce muito mais devagar do que o do algoritmo O(n^2), tornando-o mais eficiente para grandes entradas.

II. Esta afirmação é falsa. A complexidade de espaço de um algoritmo se refere à quantidade de memória que ele necessita para executar, e não ao tempo que ele leva para executar. A quantidade de memória necessária pode ser em termos do tamanho da entrada, mas também pode ser em termos de variáveis auxiliares, estruturas de dados, etc.

III. Esta afirmação é falsa. O algoritmo de ordenação por seleção tem complexidade de tempo O(n^2) mesmo na melhor das hipóteses. Ele opera encontrando o menor (ou maior, dependendo da ordem de classificação) elemento da lista e trocando-o com o primeiro elemento não classificado. Este processo é repetido para o restante dos elementos, até que toda a lista esteja classificada. Cada passo envolve uma varredura completa dos elementos não classificados, resultando em uma complexidade de tempo quadrática.

Pergunta 6

15 / 15 pts

O processo de pesquisa em memória primária é fundamental para a eficiência na recuperação e manipulação de dados. Considere um jogo de *Role-Playing Game* (RPG) online em que os jogadores podem formar grupos ou guildas para enfrentar desafios e completar missões juntos. Nesse tipo de jogo, a comunicação eficiente entre os membros do grupo é essencial para o sucesso das atividades de equipe. Uma aplicação de pesquisa em memória principal pode ser encontrada na implementação de um sistema de bate-papo em tempo real para facilitar a comunicação entre os jogadores. Em vez de realizar consultas repetitivas em um banco de dados ou em um servidor externo para recuperar as mensagens do bate-papo, as mensagens mais recentes ou as mais relevantes podem ser armazenadas em uma estrutura de dados na memória principal do sistema. Dessa forma, quando um jogador envia uma mensagem no bate-papo, essa mensagem é adicionada à memória principal e fica disponível imediatamente para todos os membros do grupo. Ao receber uma nova mensagem, os jogadores não precisam esperar por consultas demoradas ou por atualizações lentas do servidor. Em vez disso, as mensagens são exibidas instantaneamente, criando uma experiência de bate-papo em tempo real. A pesquisa em

memória principal também permite que os jogadores acessem rapidamente o histórico recente de mensagens. Se um jogador entrar em um grupo já em andamento, ele pode recuperar as mensagens anteriores armazenadas na memória principal e se atualizar rapidamente sobre as conversas anteriores. Isso facilita a integração do jogador no grupo e evita a perda de informações importantes. Ao armazenar as mensagens em memória principal, o jogo também pode implementar recursos adicionais, como pesquisas dentro do bate-papo ou filtragem de mensagens por tipo ou tópico. Essas funcionalidades podem ser realizadas de forma rápida e eficiente, uma vez que os dados estão prontamente disponíveis na memória principal. Compreender as classes de complexidade é essencial para selecionar a melhor estratégia de pesquisa. Considerando a pesquisa em memória principal, assinale a opção correta.

A classe de complexidade O(1) indica que a pesquisa requer um número constante de operações independentemente do tamanho dos dados. Essa classe é comumente encontrada em algoritmos de pesquisa sequencial.

Correto!

A classe de complexidade O(n) indica que a pesquisa requer um número de operações linearmente proporcional ao tamanho dos dados. Algoritmos como a busca sequencial se enquadram nessa classe.

Esta alternativa está correta. A classe de complexidade O(n) indica que a pesquisa requer um número de operações linearmente proporcional ao tamanho dos dados. Algoritmos como a busca sequencial se enquadram nessa classe.

A classe de complexidade $O(n^2)$ indica que a pesquisa requer um número de operações quadrático em relação ao tamanho dos dados. Algoritmos de pesquisa como o Seleção se enquadram nessa classe.

A classe de complexidade $O(n \times log\ n)$ indica que a pesquisa requer um número de operações proporcional ao produto do tamanho dos dados pelo logaritmo do tamanho dos dados. Essa classe é comumente encontrada em algoritmos de pesquisa como o Quicksort.

A classe de complexidade O(log n) indica que a pesquisa requer um número de operações proporcional ao logaritmo do tamanho dos dados. Ela é comumente encontrada em algoritmos de pesquisa linear.

Pergunta 7
15 / 15 pts

Uma das principais estruturas de dados é a matriz que possui diversas aplicações na Computação como, por exemplo, em processamento de imagens e vídeos, otimização de sistemas e teoria dos grafos. O código abaixo realiza a multiplicação entre duas matrizes quadradas.

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
```

Sobre o código acima é correto afirmar:

I. Se alterarmos o primeiro laço para *for(i = n - 1; i >= 5; i--)*, a função de complexidade do algoritmo será mantida.
II. Se alterarmos o primeiro laço para *for(i = n - 1; i >= 5; i--)*, a ordem de complexidade do algoritmo será mantida.
III. Sua função de complexidade para o número de multiplicações envolvendo os elementos do vetor é $f(n) = n^3$.

É correto o que se afirma em

Correto!

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

A função de complexidade para o número de multiplicações envolvendo os elementos do vetor é $f(n) = n^3 = O(n^3)$.

I. ERRADA - A alteração fará com que a função de complexidade seja $f(n) = (n-5)n^2$

II. CORRETA - A alteração fará com que a função de complexidade seja $f(n) = (n-5)n^2 = O(n^3)$

III. CORRETA.

Pergunta 8
15 / 15 pts

Compreender as classes de complexidade é fundamental para a análise e resolução de problemas algorítmicos em sistemas para internet. Os algoritmos podem apresentar diferentes comportamentos e eficiências dependendo da sua complexidade.

Com relação ao tema abordado, assinale a opção correta.

Os algoritmos de complexidade O(n^3) são mais eficientes do que os de complexidade O(n^2) em um sistema para internet, pois possuem um tempo de execução cúbico em relação ao tamanho da entrada. Essa complexidade é comumente encontrada em algoritmos de processamento de imagens e vídeo em sistemas para internet.

Correto!

Os algoritmos de complexidade O(n) possuem um tempo de execução proporcional ao tamanho da entrada n, o que significa que seu desempenho aumenta linearmente com o aumento do tamanho do conjunto de dados em um sistema para internet.

Correta. Os algoritmos de complexidade O(n) têm um tempo de execução linear, o que significa que o tempo de execução aumenta proporcionalmente ao tamanho da entrada. Isso é relevante em sistemas para internet, onde é comum processar grandes conjuntos de dados.

Os algoritmos de complexidade O(log n) têm um tempo de execução constante em um sistema para internet. Esses algoritmos são especialmente eficientes em problemas de busca e indexação em bancos de dados.

Os algoritmos de complexidade O(n^2) apresentam um tempo de execução quadrático em relação ao tamanho da entrada em um sistema para internet. Isso significa que o tempo de execução aumenta de forma proporcional ao tamanho do conjunto de dados.

Os algoritmos de complexidade O(1) têm um tempo de execução constante, independentemente do tamanho da entrada em um sistema para internet. Isso indica que seu desempenho é afetado pelo crescimento do conjunto de dados.

Pontuação do teste: 100 de 100