UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
Curso de Ciência da Computação
GEX609 - Pesquisa e Ordenação
Giancarlo Salton & Jefferson Caramori

Avaliação A1

12/04/2024, 13h30m

Estudante: _____

Instruções

i A prova pode ser feita a lápis, porém o professor se dará ao direito de não aceitar reclamações relativas à correção.

ii Coloque o seu nome nas folhas de respostas.

iii A duração da prova é de 2 horas.

iv Manter celulares desligados!

1. Calcule a complexidade dos algoritmos abaixo e argumente sua resposta, justificando o seu cálculo:

(a) (1.0 ponto)

```
int binary_search(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```

(b) (1.0 ponto)

```
int **matrix_multiply(int **A, int **B, int n) {
    int **C = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++) {
        C[i] = (int *)malloc(n * sizeof(int));
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            C[i][j] = 0;
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}
```

2. (1.0 ponto) Assinale VERDADEIRO (V) ou FALSO (F) para as afirmativas abaixo:

a. ( ) O algoritmo *MergeSort* divide o vetor a ser ordenado recursivamente em duas partes e intercala esses dois segmentos.

b. ( ) O algoritmo *BubbleSort* compara pares de elementos adjacentes; se o elemento da esquerda é maior do que o da direita, troca-os de posição.

c. ( ) O algoritmo *CountingSort* ordena o vetor através da contagem de ocorrências de cada valor.

d. ( ) O Algoritmo *HeapSort* utiliza uma estrutura de árvore para realizar a ordenação.

e. ( ) O algoritmo *InsertionSort* percorre o vetor da esquerda para a direita, inserindo cada elemento em sua posição correta na parte ordenada.

f. ( ) O algoritmo *QuickSort* utiliza um elemento pivô para dividir o vetor a ser ordenado.

3. (1.0 ponto) Associe a coluna da esquerda que contém critérios de avaliação relativos aos tipo de busca com a sua definição correspondente na coluna da direita e assinale abaixo a resposta que define a ordem correta de associação:

1. *SelectionSort*	( ) Divide o vetor a ser ordenado recursivamente em duas partes e intercala esses dois segmentos.
2. *BubbleSort*	( ) Compara pares de elementos adjacentes; se o elemento da esquerda é maior do que o da direita, troca-os de posição.
3. *InsertionSort*	( ) Ordena o vetor através da contagem de ocorrências de cada valor.
4. *QuickSort*	( ) Utiliza uma estrutura de árvore para realizar a ordenação.
5. *MergeSort*	( ) Percorre o vetor da esquerda para a direita, inserindo cada elemento em sua posição correta na parte ordenada.
6. *HeapSort*	( ) Utiliza um elemento pivô para dividir o vetor a ser ordenado.
7. *RadixSort*	( ) Consiste em selecionar o menor valor e trocá-lo com o elemento que está na primeira posição.
8. *CountingSort*	( ) Realiza a ordenação através dos dígitos de cada número iniciando do menos significativo para o mais significativo.

a. 5, 2, 8, 6, 3, 4, 1, 7.
b. 5, 2, 8, 6, 4, 3, 1, 7.
c. 5, 2, 7, 6, 3, 4, 1, 8.
d. 4, 1, 7, 6, 3, 5, 2, 8.
e. Nenhuma das anteriores.

4. Imagine que você está desenvolvendo um sistema embarcado com recursos limitados de hardware para um dispositivo IoT, que coleta e processa dados em tempo real. Este sistema possui apenas 128MB de RAM e um processador de baixa potência, limitando significativamente a capacidade de computação e memória disponível. Os dados coletados são armazenados em uma estrutura que mantém os itens parcialmente ordenados, e frequentemente recebe pequenas quantidades de novos dados que precisam ser integrados de forma ordenada na estrutura existente. O volume de novos dados a serem ordenados a cada atualização é pequeno, raramente excedendo 100 itens. Considerando as limitações do sistema e a necessidade de eficiência na ordenação de pequenas quantidades de dados novos ou ligeiramente desordenados:

(a) (1.0 ponto) Descreva qual algoritmo de ordenação seria escolhido na situação apresentada e argumente a sua escolha, justificando a sua resposta.

(b) (1.0 ponto) Apresente os passos de ordenação do algoritmo escolhido utilizando o vetor

```
v = {1, 2, 2, 8, 3, 10, 5, 9}
```

5. Um servidor destinado a operações de processamento de dados em grande volume necessita ordenar um vetor de 600 itens, onde cada ítem ocupa $0,5$MB, totalizando 300MB de dados a serem ordenados. A configuração do servidor inclui um processador quadcore de 2.5GHz e 8GB de RAM, dos quais apenas 1GB estão disponíveis para a operação de ordenação, devido a outros processos em execução. Além disso, estima-se que cada operação de ordenação leve aproximadamente 1 milissegundo por item. Para atender às demandas de desempenho de um sistema de análise de dados em tempo real, o vetor precisa ser ordenado e disponibilizado para consulta em menos de 3 segundos.

(a) (1.0 ponto) Descreva qual algoritmo de ordenação seria escolhido na situação apresentada e argumente a sua escolha, justificando a sua resposta.

(b) (1.0 ponto) Apresente os passos de ordenação do algoritmo escolhido utilizando o vetor

```
v = {5, 14, 9, 1, 29, 17, 6, 12}
```

6. Considere um ambiente de processamento de dados de alta performance que frequentemente lida com a ordenação de grandes conjuntos de dados. A configuração do sistema inclui um servidor com 32GB de RAM, um processador octa-core de 3.2GHz e discos SSD para armazenamento rápido. Este sistema é utilizado para processar grandes volumes de dados numéricos, que variam de milhões a bilhões de registros, cada um contendo uma chave numérica única utilizada para a ordenação. Um dos requisitos críticos é a capacidade de ordenar estes grandes volumes de dados de forma eficiente, maximizando o uso da capacidade de processamento paralelo do servidor e minimizando o tempo total de processamento. A memória disponível é suficiente para manter os conjuntos de dados a serem ordenados, mas o objetivo é otimizar o throughput de ordenação para suportar análises de dados em tempo real.

(a) (1.0 ponto) Qual seria o algoritmo mais adequado para cumprir as restrições da situação descrita acima? Justifique sua escolha, levando em consideração as características de cada algoritmo.

(b) (1.0 ponto) Apresente os passos de ordenação do algoritmo escolhido utilizando o vetor

```
v = {38, 28, 31, 22, 33, 6, 10, 19, 12, 47}
```