FATEC IPIRANGA

PASTOR ENÉAS TOGNINI

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO ESTRUTURADA E MODULAR

PROFESSOR CARLOS HENRIQUE VERISSIMO PEREIRA

MILENA MITIE AOKI

SÃO PAULO, SP
2024

SUMÁRIO

N2-2-COMPARANDO ORDENAÇÃO "BUBBLE SORT" COM ORDENAÇÃO POR INSERÇÃO

1. MEDIÇÕES

Modificações Necessárias para Medir Tempo, Trocas e Ciclos

Para comparar ambos os algoritmos conforme os critérios especificados, precisamos modificar os códigos para:

-Medir o tempo de execução

-Contabilizar o número de trocas

-Contabilizar o número de ciclos

Bubble Sort Modificado:

```
#include <stdio.h>
#include <time.h>

// Função para trocar dois valores e contar trocas
void swap(int *a, int *b, int *trocas) {
    int temp = *a;
    *a = *b;
    *b = temp;
    (*trocas)++; // Incrementa o contador de trocas
}

// Função para mostrar o array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Função para realizar o Bubble Sort com contagem de trocas e ciclos
void bubbleSort(int arr[], int n, int *trocas, int *ciclos) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            (*ciclos)++; // Conta o número de ciclos
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1], trocas);
            }
        }
```

```
    }
}

int main() {
    int arr[] = {5, 3, 8, 4, 2,
        115, 113, 118, 114, 112,
        125, 123, 128, 124, 122,
        35, 33, 38, 34, 32,
        45, 43, 48, 44, 42,
        55, 53, 58, 54, 52,
        65, 63, 68, 64, 62,
        75, 73, 78, 74, 72,
        85, 83, 88, 84, 82,
        95, 93, 98, 94, 92,
        15, 13, 18, 14, 12,
        25, 23, 28, 24, 22
    };

    int n = sizeof(arr) / sizeof(arr[0]);
    int trocas = 0;
    int ciclos = 0;

    clock_t start, end;
    double cpu_time_used;

    // Inicia medição de tempo
    start = clock();

    // Executa Bubble Sort
    bubbleSort(arr, n, &trocas, &ciclos);

    // Finaliza medição de tempo
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Exibe resultados
    printf("Array ordenado: ");
    printArray(arr, n);
    printf("Tempo de execução: %f segundos\n", cpu_time_used);
    printf("Quantidade de trocas: %d\n", trocas);
    printf("Quantidade de ciclos: %d\n", ciclos);

    return 0;
}
```

Insertion Sort Modificado:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Ordena um array usando o algoritmo de ordenação por inserção com contagem de trocas e ciclos

void ordenacaoInsercao(int numeros[], int quantidade, int *trocas, int *ciclos) {

    int i, elemento, posicao;

    for (i = 1; i < quantidade; i++) {

        elemento = numeros[i];

        posicao = i - 1;

        while (posicao >= 0 && numeros[posicao] > elemento) {

            (*ciclos)++; // Conta o número de ciclos

            numeros[posicao + 1] = numeros[posicao];

            posicao--;

            (*trocas)++; // Conta uma "quase troca"

        }

        numeros[posicao + 1] = elemento;

    }

}



// Imprime o array
```

```
void imprimirArray(int numeros[], int quantidade) {
    for (int i = 0; i < quantidade; i++)
        printf("%d ", numeros[i]);
    printf("\n");
}
int main() {
    int arr[] = {5, 3, 8, 4, 2,
        115, 113, 118, 114, 112,
        125, 123, 128, 124, 122,
        35, 33, 38, 34, 32,
        45, 43, 48, 44, 42,
        55, 53, 58, 54, 52,
        65, 63, 68, 64, 62,
        75, 73, 78, 74, 72,
        85, 83, 88, 84, 82,
        95, 93, 98, 94, 92,
        15, 13, 18, 14, 12,
        25, 23, 28, 24, 22
    };
    int n = sizeof(arr) / sizeof(arr[0]);
    int trocas = 0;
```

```
    int ciclos = 0;
    clock_t start, end;
    double cpu_time_used;
    // Inicia medição de tempo
    start = clock();
    // Executa Insertion Sort
    ordenacaoInsercao(arr, n, &trocas, &ciclos);
    // Finaliza medição de tempo
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    // Exibe resultados
    printf("Array ordenado: ");
    imprimirArray(arr, n);
    printf("Tempo de execução: %f segundos\n", cpu_time_used);
    printf("Quantidade de trocas: %d\n", trocas);
    printf("Quantidade de ciclos: %d\n", ciclos);
    return 0;
}
```

COMPARAÇÃO

1. Complexidade Temporal

Bubble Sort: -Pior Caso: O(n²)

-Melhor Caso: O(n) (se otimizado com uma flag para verificar se houve trocas)

Insertion Sort: -Pior Caso: O(n²)

-Melhor Caso: O(n) (quando o array já está quase ordenado)

2. Eficiência em Arrays Parcialmente Ordenados

-Bubble Sort: Menos eficiente em arrays quase ordenados, pois continua realizando todas as iterações mesmo que o array já esteja ordenado.

-Insertion Sort: Mais eficiente em arrays quase ordenados, pois realiza menos comparações e movimentos.

3. Número de Trocas e Movimentações

-Bubble Sort: Realiza muitas trocas, o que pode ser custoso em termos de tempo, especialmente para grandes conjuntos de dados.

-Insertion Sort: Realiza menos movimentações, apenas movendo os elementos quando necessário para inserir o elemento na posição correta.

RESULTADOS

1. Tempo de Execução

-Bubble Sort teve um tempo de execução maior devido ao maior número de trocas.

-Insertion Sort foi mais rápido, especialmente quando o array estiver em partes já ordenadas.

2. Quantidade de Trocas

-Bubble Sort teve o mesmo número de trocas (movimentações).

-Insertion Sort teve o mesmo número de trocas (movimentações).

3. Quantidade de Ciclos

-Bubble Sort realizou mais ciclos devido aos loops aninhados.

-Insertion Sort realizou menos ciclos.

TABELA DE RESULTADOS

Algoritmo	Tempo de Execução (s)	Número de Trocas	Número de Comparações
Bubble Sort	0.000010	884	1770
Insertion Sort	0.000004	884	884

Análise dos Resultados

1. **Tempo de Execução**: O Insertion Sort teve um tempo de execução menor (0.000004 segundos) em comparação ao Bubble Sort (0.000010 segundos), indicando que, para o conjunto de dados testado, o Insertion Sort foi mais eficiente.
2. **Número de Trocas**: O Bubble Sort realizou o mesmo número de trocas (884) do que o Insertion Sort (884). Ambos os algoritmos realizaram o mesmo número de trocas. Isso pode ser resultado da forma como as trocas foram contabilizadas nas implementações.
3. **Número de Comparações**: O número de comparações também foi maior no Bubble Sort (1770) em comparação ao Insertion Sort (884). O Insertion Sort geralmente é mais eficiente em termos de comparações, especialmente em listas que já estão parcialmente ordenadas.